PUBLIC
SAP HANA Platform 2.0 SPS 06
Document Version: 1.0 – 2021-12-03

SAP HANA Developer Guide for SAP HANA XS Advanced Model

© 2023 SAP SE or an SAP affiliate company. All rights reserved.

THE BEST RUN SAP

Content

1 SAP HANA Developer Guide for XS Advanced Model. 8

2 Introduction to Application Development and Deployment (XS Advanced Model). 9

2.1 SAP HANA Extended Application Services, Advanced Model. 10

2.2 SAP HANA XS Advanced Multi-Target Applications. 11

2.3 The XS Advanced Programming Model. 14

2.4 XS Advanced Application Development Tools. 22

2.5 Developer Information Map for XS Advanced. 25

3 Getting Started with SAP HANA Development. 27

3.1 Getting Started with Application Development in XS Advanced. 27

Working with the SAP Web IDE for SAP HANA. 28

Working with the XS Advanced Command-Line Client. 75

Working with Gerrit Version Control in XS Advanced. 95

Set up an XS Advanced Application Project. 101

3.2 Working with the SAP Cloud Application Programming Model. 108

Create a Basic HTML5 Module for your SAP Cloud Platform Business Application. 110

Create the Core Data Services for your SAP Cloud Platform Business Application. 114

Create the Node.js Back-end Services for your SAP Cloud Platform Business Application. 118

Create the SAPUI5 Web UI for your SAP Cloud Platform Business Application. 128

4 Maintaining the XS Advanced Application Development & Deployment Descriptors. 136

4.1 Create the MTA Description Files. 137

Multi-Target XS Advanced Applications. 139

The MTA Development Descriptor. 141

The MTA Deployment Descriptor. 147

The MTA Deployment Extension Descriptor. 210

4.2 Deploy a Multi-Target Application (with Command-Line Tools). 211

The MTA Archive. 215

The MANIFEST.MF File. 216

The MTA Archive (MTAR) Builder. 217

4.3 Deploy a Multi-Target Application with Blue-Green Deployment. 219

4.4 Deploy a Multi-Target Application with Zero-Downtime Maintenance. 221

Zero-Downtime Maintenance with Multi-Target Applications. 223

HDI Artifacts and Database Changes Supported by ZDM. 226

5 Defining the Data Model in XS Advanced. 233

5.1 Managing SAP HDI Containers and Artifacts. 234

Naming Conventions in SAP HDI. . . . 236
Set up an SAP HDI Container for XS Advanced Applications. . . . 237
Define Run-Time Name-Space Rules for SAP HDI. . . . 247
5.2 Configuring the HDI Deployer. . . . 253
Integrate the HDI Deployer into an XS Advanced MTA Database Module. . . . 254
5.3 Creating the Data Persistence Artifacts in XS Advanced. . . . 277
Design-Time Database Resources in XS Advanced. . . . 277
HDI Design-Time Resources and Build Plug-ins. . . . 279
Creating Data Persistence Artifacts with SQL DDL in XS Advanced. . . . 284
Creating Data Persistence Artifacts with CDS in XS Advanced. . . . 310
5.4 Maintaining JSON Collections in the SAP HANA Document Store. . . . 449
Create a JSON Document Store. . . . 450
Add and Remove JSON Collections in the Document Store. . . . 452
Import JSON Documents into a DocStore Collection. . . . 456
5.5 Creating Procedures and Functions in XS Advanced. . . . 459
Create the Underlying Data Artifacts for a Procedures. . . . 460
Create a Database Procedure in XS Advanced. . . . 470
Create a Database Function in XS advanced. . . . 473
5.6 Using Synonyms to Access External Schemas and Objects in XS Advanced. . . . 476
Enable Access to Objects in a Remote Classic Schema. . . . 476
Enable Access to Objects in Another HDI Container. . . . 505
5.7 Create Virtual Tables. . . . 510
5.8 Setting Up the Analytic Model. . . . 511
Create Calculation Views. . . . 512
Preview Calculation View Output. . . . 518
Defining Data Access Privileges. . . . 520

6 Defining Web-based Data Access in XS Advanced. . . . 522
6.1 Maintaining OData Services in XS Advanced. . . . 522
Defining OData v2 Services for XS Advanced JavaScript Applications. . . . 524
Defining OData v4 Services for XS Advanced Java Applications. . . . 580
6.2 Data Access with XMLA in SAP HANA XS. . . . 610
Define the Data to Expose with an XMLA Service in XS Advanced. . . . 611
Set up and Use the XMLA Interface in XS Advanced. . . . 612
6.3 Data Access with InA in XS Advanced. . . . 618
Maintain the Analytics Adapter in XS Advanced. . . . 618

7 Writing the XS Advanced Application Code. . . . 625
7.1 The SAP HANA XS Advanced JavaScript Run Time. . . . 627
Secure Programming with JavaScript. . . . 631
Download and Consume SAP Node.js Packages. . . . 633
Download and Install JavaScript Data Services. . . . 689

Tutorial: Setting up your JavaScript Application in XS Advanced. 699
7.2 The SAP HANA XS Advanced Java Run Time. 716
Download and Consume Java Libraries. 719
Tutorials: Setting Up the Java Run-Time Environment in XS Advanced. 735
7.3 The SAP HANA XS Advanced Python Run Time. 778
Set Up the Python Run Time for XS Advanced. 778
Download and Consume Python Libraries. 782
Tutorials: Setting Up the Python Run-Time Environment in XS Advanced. 792
7.4 Custom Run-Time Environments in SAP HANA XS Advanced. 804
Best Practices for Deploying Applications in a Custom Run-Time Environment. 805
Custom Buildpacks in XS Advanced. 808
Create a PHP Buildpack for XS Advanced. 811
Available Build Packs. 816
7.5 Create the XS Advanced Application Package Descriptor. 818
The XS Advanced Application Package Descriptor. 819
XS Advanced Application Package Descriptor Configuration Syntax. 822

8 Maintaining Application Services in XS Advanced. 826
8.1 Core Services in XS Advanced. 827
Audit Log Services in XS Advanced. 828
File-System Storage Services in XS Advanced. 831
SAP HANA and HDI Services in XS Advanced. 834
Instance Manager Services in XS Advanced. 841
User Account and Authentication Services in XS Advanced. 841
8.2 Additional Services in XS Advanced. 846
Messaging Services in XS Advanced. 847
Job Scheduler Services in XS Advanced. 850
SAP UI5 Services in XS Advanced. 853
8.3 Service Plans and Resources in XS Advanced. 856
8.4 Service Types in XS Advanced. 859
8.5 Create a Service Instance. 861
The Service Brokers. 862
The Service-Broker CLI. 863
8.6 Update a Service Instance. 865
8.7 Set up and Use an Application-Managed Instance. 867
Managed Service Instance API URLs and Properties. 868
8.8 Set up and Use the XS Advanced Message Service. 872
Consuming Messages with the XS Advanced Message Service. 874
8.9 Scheduling Jobs in XS Advanced. 881
Configure the Service Broker for Job Scheduler. 882
Create a Job Scheduler Service Instance 884
Bind an Application to the Job Scheduler Service. 885

Maintain Jobs and Job Schedules in XS Advanced. 886

9 Creating the Client User Interface. 909

9.1 Building User Interfaces with SAPUI5 for SAP HANA. 909

9.2 Consuming Data and Services with SAPUI5 for SAP HANA. 910

10 Maintaining XS Advanced Application Routes and Destinations. 912

10.1 Configure the XS Advanced Application Router. 913

XS Advanced Application-Router Resource Files. 916

The XS Advanced Application Descriptor. 917

XS Advanced Application Router Configuration Syntax. 922

XS Advanced Application Routes and Destinations. 936

XS Advanced Application Router Environment Variables 938

10.2 Extend the XS Advanced Application Router. 949

XS Advanced Application Router Extensions. 951

Custom Middleware Injection. 953

XS Advanced Application Router Extension API. 954

11 Maintaining Application Security in XS Advanced. 957

11.1 Identity, Authentication, and Authorization. 958

Authorization in SAP HANA XS Advanced. 961

OAuth 2.0 in XS Advanced. 962

Building an Authorization Concept for Users of an XS Advanced Application Instance. 963

Example Scenario for Leave Request Authorizations. 963

11.2 Set up Authentication for an XS Advanced Application. 965

11.3 Set up Generic Authorization for an XS Advanced Application. 969

Application Authorization Tasks, Roles, and Tools in XS Advanced. 972

11.4 Assign Functional Authorization Scopes for an XS Advanced Application. 974

Scopes for Functional Authorization Checks. 975

The Data Control Language. 976

11.5 Set up Application Security in XS Advanced. 978

XS Advanced Security Tools and Components. 984

Personalizing the Database Connection with JSON Web Tokens. 989

11.6 Create the Security Descriptor for an XS Advanced Application 995

The Application Security Descriptor. 996

Application Security Descriptor Configuration Syntax. 999

11.7 Create an Instance of the XS UAA Service. 1013

Modifications and Restrictions for XSUAA Service Updates. 1014

11.8 Bind the XS UAA Service Instance to the XS Advanced Application. 1016

11.9 Enable Applications to Perform Tasks with the Authority of Other Applications. 1017

Authorization Scopes and Authorities. 1018

11.10 Maintain Values in the SAP HANA Secure Store. 1020

SAP HANA Secure-Store Interface Procedures and Parameters. 1022
Access the SAP HANA Secure Store with Node.js. 1025
Access the SAP HANA Secure Store with Java. 1031
Access the SAP HANA Secure Store with Python. 1035
Access the SAP HANA Secure Store with SQL. 1041

12 Migrating XS Classic Applications to XS Advanced Model. 1043
12.1 The XS Advanced Application-Migration Process. 1044
12.2 The XS Advanced Migration Assistant. 1046
12.3 The XS Application Migration Report. 1048
12.4 Legacy Object Types not Supported in XS Advanced. 1049

13 HDI Artifact and Build Plug-in Configuration. 1051
13.1 HDI Artifact Types and Build Plug-ins 1052

14 The XS Command-Line Interface. 1057
14.1 The XS Command Overview. 1058

15 SAP Web IDE for SAP HANA Reference. 1060
15.1 Product Overview. 1060
15.2 Getting Started. 1062
Navigating SAP Web IDE. 1063
Setting User Preferences. 1065
Enable SAP Web IDE Extensions. 1066
Keyboard Shortcuts. 1067
Search Options. 1071
Working with Files and Folders. 1075
Resizing Panes. 1077
15.3 Developing Multitarget Applications. 1078
Inside an MTA Descriptor. 1079
Application Development Workflow 1086
Setting Up Application Projects. 1087
Developing SAP HANA Database (HDB) Modules. 1090
Developing Node.js Modules. 1129
Developing Java Modules. 1140
Developing HTML5 Modules. 1150
Developing Business Applications. 1222
Developing SAP Fiori Launchpad Modules. 1223
Packaging and Deploying Applications. 1234
15.4 Customizing Your Project. 1235
Customize JavaScript Beautifier Properties. 1235
Customize Code Checking Rules. 1236
Configure Mock Data Usage. 1237

15.5 Using Code Editors. 1238
Configure the Code Editor. 1238
Working in the Code Editor. 1239
Generate JSDoc Comment Snippets. 1243
Using Code Completion. 1243
Checking Code 1246
Locate Objects in Code. 1253
15.6 Using Source Control (Git). 1254
Set Up Git. 1257
Clone Repositories. 1258
Initialize a Local Git Repository. 1259
Set a Remote Repository. 1259
Fetch Changes. 1260
Rebase Changes. 1261
Merge Changes. 1262
Pull Changes. 1262
Stage Files. 1263
Commit Changes. 1265
Push Changes. 1266
Multiple Local Branches. 1266
Create Remote Branches. 1268
Git History. 1269
Set Up Git to Work with Gerrit. 1272
15.7 Protecting User Data. 1274
Export User Workspaces. 1274
Manage User Service Instances. 1275
Delete User Data. 1275
Log Changes of Personal Data. 1275
15.8 Troubleshooting. 1276
Archive Import Troubleshooting. 1276
Git Troubleshooting. 1277
Grunt Build Troubleshooting. 1279
Running Applications Troubleshooting. 1281
Remote Connectivity Troubleshooting. 1283

1 SAP HANA Developer Guide for XS Advanced Model

This guide explains how to build applications using SAP HANA, including how to model persistent and analytic data, how to write procedures, and how to build application logic in SAP HANA Extended Application Services advanced model (XS advanced).

The *SAP HANA Developer Guide for SAP HANA XS Advanced Model* explains the steps required to build and deploy applications that run in the SAP HANA XS advanced model run-time environment. It also describes the technical structure of applications that can be deployed to the XS advanced run-time platform using both SAP Web IDE for SAP HANA and XS command line tools. The information in the guide is organized as follows:

- Introduction and overview
  A high-level overview of the basic capabilities and architecture of SAP HANA XS advanced model. This section also includes an information map, which is designed to help you navigate the library of information currently available for SAP HANA developers.
- Getting started
  A collection of tutorials which are designed to demonstrate how to build and deploy a simple SAP HANA-based application on SAP HANA XS advanced model quickly and easily
- The development process
  Step-by-step information that shows in detail how to develop the various elements that make up an XS advanced application. The information provided uses tasks and tutorials to explain how to develop the SAP HANA development objects. Where appropriate, you can also find background information that explains the context of the task, and reference information that provides the detail you need to adapt the task-based examples to suit the requirements of your application environment.
- Reference sources
  A selection of reference guides that support the XS advanced application development process, for example: a command reference for the `xs` command-line interface; a complete list of the HDI plug-ins that you need to ensure that your XS advanced, design-time artifacts are deployed correctly in the XS advanced run-time environment; and a comprehensive guide to the tools provided with the browser-based, integrated development environment, SAP Web IDE for SAP HANA.

Related Information

Introduction to Application Development and Deployment (XS Advanced Model) [page 9]

2 Introduction to Application Development and Deployment (XS Advanced Model)

The SAP HANA Developer Guide for SAP HANA® XS, advanced model, presents a developer's view of the SAP HANA XS, advanced model, run-time platform.

The *SAP HANA Developer Guide* guide for SAP HANA XS advanced model explains how to build, deploy, and maintain applications that run in the SAP HANA XS advanced model run time. The information provided also describes the technical structure of applications that can be deployed to the XS advanced run-time platform. This guide is aimed at people performing the following developer roles:

- Database developers
  Often a business/data analyst or database expert, the database developer is concerned with the definition of the data model and schemas that will be used in SAP HANA XS advanced model, the specification and definition of tables, views, primary keys, indexes, partitions and other aspects of the layout and inter-relationship of the data in SAP HANA.
  The database developer is also concerned with designing and defining authorization and access control, through the specification of privileges, roles, and users and, in addition, authorization scopes.
- Application programmers
  The programmer is concerned with building SAP HANA XS advanced model applications: applications that are one part of so-called Multi-Target Applications (MTAs). Programmers develop the code for the business-logic component, for example, in JavaScript (Node.js or XS JavaScript) or Java.
- Client UI developers
  The user-interface (UI) client developer designs and creates client applications which bind business logic (from the application developer) to controls, events, and views in the client application user interface. In this way, data exposed by the database developer can be viewed in the client application's UI.

> ⚠ Caution
>
> The code examples included in this document for XS advanced are often syntactically incomplete; they are intended for illustration purposes only.

Related Information

SAP HANA Extended Application Services, Advanced Model [page 10]
SAP HANA XS Advanced Multi-Target Applications [page 11]
The XS Advanced Programming Model [page 14]
XS Advanced Application Development Tools [page 22]
Developer Information Map for XS Advanced [page 25]

2.1 SAP HANA Extended Application Services, Advanced Model

SAP HANA extended application services, advanced model, (XS advanced) provide a comprehensive platform for the development and execution of native data-intensive applications.

SAP HANA functions as a comprehensive platform for the development and execution of native data-intensive applications that run efficiently in SAP HANA, taking advantage of its in-memory architecture and parallel execution capabilities. Structured accordingly, applications can profit from the increased performance provided by SAP HANA due to the integration with the data source.

XS advanced is a polyglot application platform that supports several programming languages and execution environments, for example, Java and Node.js. The classic XS JavaScript (XSJS) is supported by a framework running in the Node.js run time.

In simple terms, XS advanced is basically the Cloud Foundry open-source Platform-as-a-Service (PaaS) with a number of tweaks and extensions provided by SAP. These SAP enhancements include amongst other things: an integration with the SAP HANA database, OData support, compatibility with XS classic model, and some additional features designed to improve application security. XS advanced also provides support for business applications that are composed of multiple micro-services, which are implemented as separate Cloud Foundry applications, which combined are also known as Multi-Target Applications (MTA). A multi-target application includes multiple so-called “modules” which are the equivalent of Cloud Foundry applications.

XS Advanced Application Development Scenarios

For developers considering building applications that run on SAP HANA XS advanced model, the following general rules apply:

- SAP HANA XS advanced applications
  Consider SAP HANA native development if you want to develop new applications, for example, Java or JavaScript (including Node.js) which will run solely on SAP HANA XS advanced. These new XS advanced applications are typically multi-target applications (MTA), which comprise multiple “modules” (software components), which all share a common life cycle for development and deployment.
- Integrated applications
  Combine SAP HANA native development with ABAP to enrich existing applications (for example, SAP HANA Live, Fiori).
- SAP Business Suite or BW
  In the context of SAP Business Suite or Business Warehouse (BW), use ABAP if the scope is simply to optimize existing programs.

Related Information

Introduction to Application Development and Deployment (XS Advanced Model) [page 9]
Developer Information Map for XS Advanced [page 25]

2.2 SAP HANA XS Advanced Multi-Target Applications

Multi-Target Applications comprise several modules that contain content for a distinct run-time environment.

Business applications are typically composed of several parts that are built with different technology for different run-time environments. For example, an application could contain static Web content that runs in the browser, server-side Java code that runs in a Java Enterprise server, OData service definitions for an OData provisioning run time, and also database content such as tables, views, and procedures. Since all these parts belong to the same business application, they are developed, delivered, configured, and deployed together. Often the various different parts have dependencies and, as a result, need to be deployed to the specified target in a given order. To manage such applications, SAP has introduced the concept of a multi-target application (MTA). XS Advanced supports the MTA concept, however, the approach is generic and not limited to SAP HANA Extended Application Services.

Each MTA consists of one or more modules. In XS advanced terminology, each MTA module is a separate XS advanced ("micro") application, which can be pushed to a target deployment platform, bound to any required services, and so on. MTA modules can expose certain attributes for use by other modules and they can have dependencies on other modules or resources, too. A module could, for example, depend on a REST API of some other module, or on the tables and views defined by a database module.

> → Tip
>
> Examples of resources, on which modules can depend are: XS services instances such as the SAP HANA User Account and Authentication (UAA) instance or an SAP HANA service instance.

The MTA application model with modules and dependencies is specified in a MTA deployment descriptor file. The MTA deployment descriptor lists the different MTA modules with their technical type, their dependencies, and any required parameters. This information is used to ensure that the modules are deployed in the right order, that the required dependencies (such as services) exist, and for setting up the connections required between the modules. The application files and metadata such as descriptors and manifests are collected into an MTA, which can be packaged in (and delivered as) an archive. The MTA archive also includes information for configuring services, for example the security configuration for the UAA instance.

In the folder structure of the MTA archive, each module has its own folder. For example, an XS advanced MTA could have the following base-level folders:

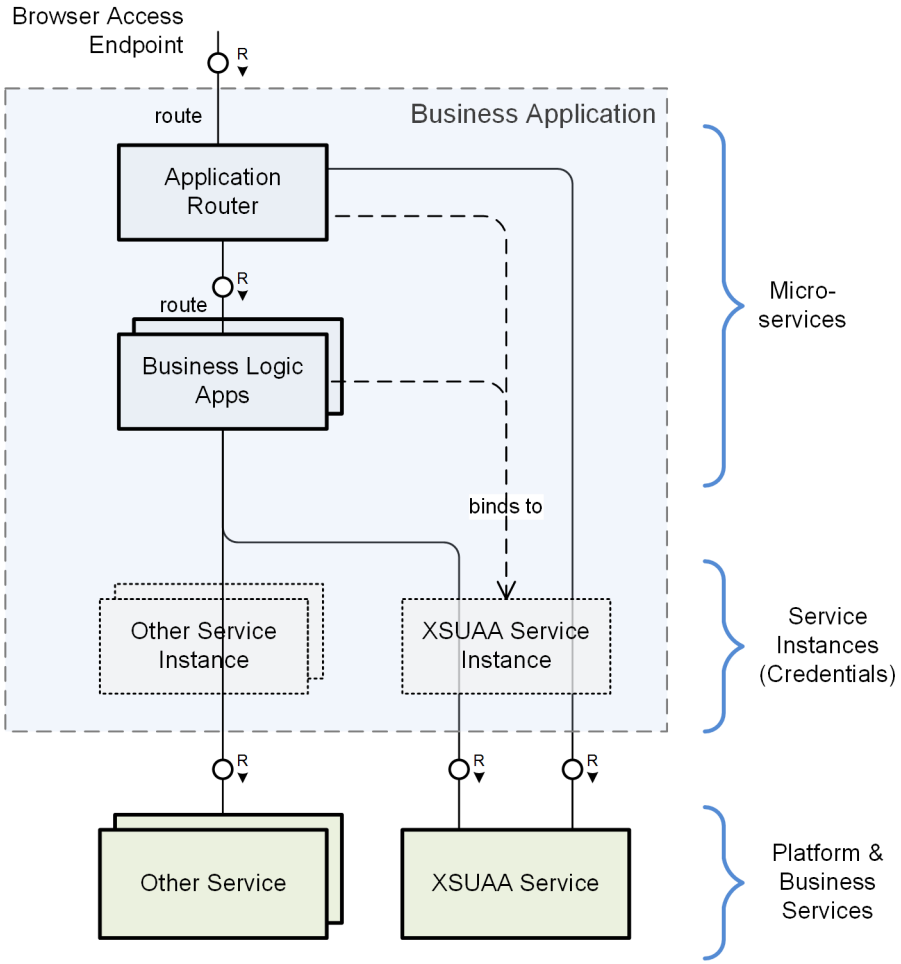
- `web/`
  A folder for static Web content and the application router configuration
- `java/`
  A folder for a Java application
- `js/`
  A folder for a Node.js application (including XS JavaScript, too)
- `db/`
  A folder for the SAP HANA database artifacts, for example: tables, views, stored procedures, Calculation Views, and so on...

> → Tip
>
> The mapping of folders to modules is described in an MTA manifest file.

Application Pattern

SAP is promoting a pattern for building XS advanced applications that can run either on-premise or in the Cloud. The logical view of this pattern shown in the diagram below represents the architecture of a business application, a term used in the context of Cloud Foundry to distinguish such applications from single applications. For an end-user's perspective, a business application is a collection of microservices, service instances, bindings, and routes which, taken as a whole, represent a usable Web-based application. These microservices, services instances, bindings, and routes are created by communicating with the Platform Controller (for example, using a command-line interface).

> → Tip
>
> SAP provides a set of libraries, services, and component communication principles which are used to implement (multi-tenant) business applications according to the application pattern described here.



- Microservices
  Microservices are created by "pushing" code to the target platform resulting in the deployment of a number of microservice instances each running in a separate container
- Services
  Services are exposed to applications by injecting access credentials into the application environment by means of service bindings. Applications are bound to a service instance which describes the configuration

and credentials required to consume a service. Services instances are managed by a service broker which must be provided for each service (or for a collection of services).

  - XS User Account and Authentication (UAA) service
    The UAA service is a multi-tenant identity management service used in Cloud Foundry. The UAA's primary role is as an OAuth2 provider, issuing tokens for client applications to use when they act on behalf of Cloud Foundry users. The UAA can also authenticate users with their Cloud Foundry credentials and can act as an SSO service using those credentials (or others). As well as various other management functions, the UAA provides endpoints for managing user accounts and for registering OAuth2 clients.
  - Platform and Business Services
    Along with backing services such as SAP HANA, MongoDB, PostgreSQL, Redis, RabbitMQ, Audit Log, Application Log, etc., the platform also provides various business services, for example, to enable you to retrieve currency exchange rates. Applications can also make use of so-called "user-provided" services that are not managed by the platform. In all these cases, applications can bind and consume the required services in a similar way.
- Routes
  Mapped to applications and define the URL to use to access the business application

Application Deployment

The XS deploy service is, as the name suggests, used to deploy MTAs on the XS advanced platform. The deployment service is an XS Advanced application for the deployment of MTA archives or directory structured in the same way as an MTA archive. The deploy service can be invoked from the XS advanced command-line client, for example, with the `xs deploy` command.

When the deploy service receives an MTA, it first validates it for consistency. Next it ensures that the required service instances exist and creates them if necessary. It deploys the different modules in the right order, based on the dependency information in the MTA descriptor. The deploy service also sets up environment variables with the information that is needed by the different modules to connect to required modules at run time. The deploy service executes those tasks required for the successful deloyment of the MTA, for example, creating service instances or pushing applications to the desired target run-time platform.

The deploy service supports transactional execution which includes the ability to restart a multi-step deployment that was interrupted. Multiple instances of the deploy service can be started for high availability. The deploy service also supports so-called “zero-down-time” updates, for example, with the blue-green deployment update described in the Cloud Foundry documentation.

Related Information

The XS Advanced Programming Model [page 14]
SAP HANA Extended Application Services, Advanced Model [page 10]
XS CLI: Plug-ins

2.3 The XS Advanced Programming Model

Writing applications for deployment to SAP HANA XS advanced.

SAP HANA Extended Application Services advanced model (XS advanced) adds an application platform to the SAP HANA in-memory database. In the Cloud, this platform is provided by Cloud Foundry. An SAP-developed run-time environment is bundled with SAP HANA on-premise which provides a compatible platform that enables applications to be deployed to both worlds: the Cloud and on-premise. XS advanced is optimized for simple deployment and the operation of business applications that need to be deployed in both worlds. For this reason, the XS advanced programming model fully embraces the Cloud Foundry model and leverages its concepts and technologies. In areas where Cloud Foundry as an intentionally generic platform for distributed Web applications does not address relevant topics or offers a choice of solutions, the XS advanced programming model provides guidance that is in line with the general Cloud programming model.

> **i** Note
>
> It is important to bear in mind that the availability of services can differ according to the target platform, for example, SAP HANA on-premise or SAP Cloud Platform. Some platform services are not always available in both environments, and the situation regarding the availability of services can change over time.

In this section, you can find information about the following topics:

- Cloud Foundry Concepts [page 14]
- System Architecture [page 16]
- Run-Time Platform [page 17]
- Authentication and Authorization [page 17]
- Component Model [page 18]
- Client User Interface [page 18]
- OData Services [page 19]
- SAP HANA Database [page 19]
  Includes information about the Service Broker, HDI containers, the HDI deployer, the SAP HANA Instance Broker, and database synonyms, as well as Core Data Services

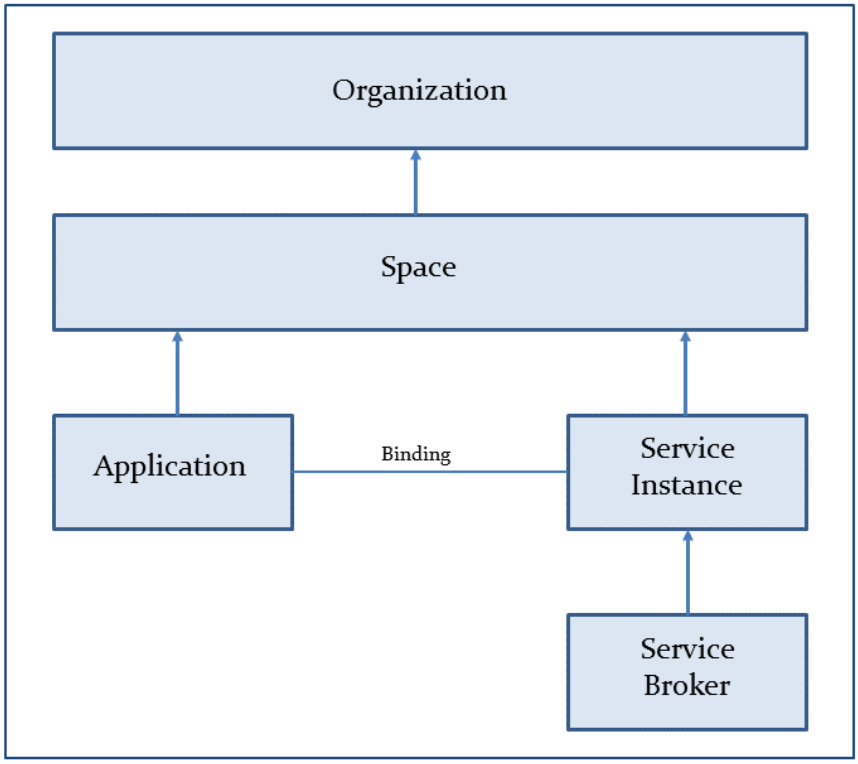
Cloud Foundry Concepts

The Cloud Foundry platform runs applications; these applications belong to “spaces”, which belong in turn to “organizations”.

In the context of an organization, the platform users can be assigned special permissions in the different spaces. For example, administrators and developers need permissions to deploy and start applications. Spaces can be used to provide a secure and isolated environment for the development teams working on different projects. Applications can use various runtime “services”, for example, in XS advanced: database services, the User Account and Authentication service (UAA), or a scheduler service. Each service can be offered in different configurations called “service plans”, which have different features and characteristics.

Before a service can be used by an application, it is necessary to create a service “instance” to which the application must be bound. For each type of service there is a “service broker”; the service broker executes

the actions required to create and delete service instances and bind application to services. The UAA service broker, for example, implements the API for managing UAA service instances. What a service instance means technically may differ between services and service plans. A service instance can be just a configuration for a shared implementation and a separate login and data area on a shared server. In certain other cases, each service instance can have its own server.



Basic Cloud Foundry Concepts

Applications are deployed to the target platform by using the `push` operation of the platform API. For this reason, in Cloud Foundry parlance, applications are "pushed" to the platform. Pushing an application works as follows:

1. First the application files are uploaded to the platform.
2. Next, programs called "buildpacks" are executed to create archives that create the self-contained and ready-to-run executable applications.
   Some of the most typical activities executed by buildpacks are: downloading any required libraries and other dependencies, and configuring the application. Different buildpacks exist for the different target run-time environments such as Java or JavaScript/Node.js.

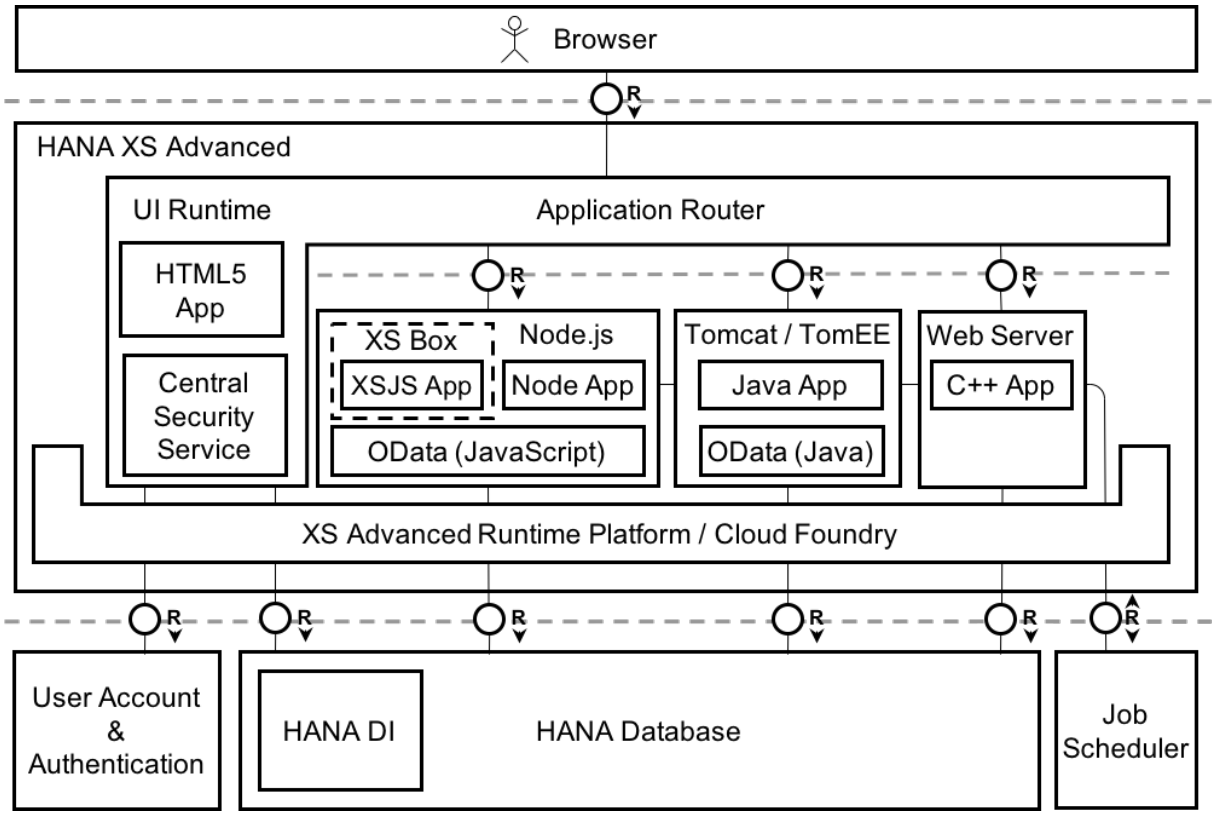
Applications are started as separate processes. At run time, the applications need the connection information for the services instances to which they are bound. The applications obtain this information from process-specific environment variables, which are resolved by the platform in a process known as "service wiring". Bindings can only be created between applications and service instances in the same "space". For scale-out scenarios, the platform can be instructed to start multiple instances of an application (several processes). XS advanced on-premise also implements the concepts of spaces, applications, services, buildpacks and droplets.

Like Cloud Foundry applications, XS advanced applications are units that can be pushed, started, stopped, and bound to services. You can display a list of all the applications running in the current space with the `xs apps` command.

> **i** Note
>
> Each XS advanced application runs in a specific run-time container.

System Architecture

Like Cloud Foundry, XS advanced is optimized for 12-Factor applications: the new methodology for building scalable, easy-to-maintain, software-as-a-service applications. Unlike in XS classic, where design-time artifacts are stored in the SAP HANA Repository, the code base for an XS advanced application is stored in an external Git repository. Applications are deployed (pushed) from the Git repository to the target platform and run in separate processes that can be scaled independently. The applications expose services by means of port bindings, store their configuration in the environment, and build upon backing services that are consumed over the network. XS advanced applications build upon the micro services paradigm and leverage Cloud Foundry tools and services to create, bind, and dispose of services.



XS Advanced System Architecture

Typically, the following service layers are available:

- Backing Services

Provide the technology on which XS advanced application are built, for example:

  - Persistence services (databases, key-value stores, ...)
  - Communication services (message queues, email gateways, ...)
  - Authentication services (User Account and Authentication service (UAA))

- Application Services
  Implement business logic and build upon backing services. Application services are implemented in different languages (for example, Java or JavaScript) for which the platform provides appropriate run-time environment (for example, Tomcat or Node.js). Communication between application services is always remote since the application services run in isolated containers that are managed by the platform.
- Mashup Services
  Combining application services, mashup services expose central end points to the user interface that runs as a rich client in a Web browser. In their simplest form, mashup services perform simple URL-based routing, for example, provided by the XS advanced Application Router (`approuter`) or the Fiori Launchpad. However, mashup services can also implement more complex scenarios, for example, the service calls seen in next-generation micro-service based platforms such as “hybris-as-a-service” (YaaS).

Run-Time Platform

In XS advanced, an application run-time environment comprises lightweight processes that are invoked over HTTP and communicate remotely with the database through a database client (for example, SQL or MDX). Each run-time environment provides an execution environment for a specific language and platform, for example, JavaScript (XS JS or Node.js), Java (JavaEE). and even native code written in C or C++. Higher-level functionality provided by XS classic (for example, the XMLA service and the job scheduler) are implemented as applications on top of those XS advanced run times.

Generic functionality like load balancing, elastic scaling, routing, identity and access management, API management, and application lifecycle management are provided by an XS run-time platform underneath the XS advanced run-time environments. The XS advanced run-time platform provides a number of services for managing the various container instances and their run-time environments. Containers manage run-time environments and allow for isolation, resource management and shared service injection.

Besides the functionality for starting and stopping containers, the XS advanced run-time platform provides a monitoring service for obtaining an overview of the started container instances, run-time environments, and applications. This service also includes information about consumed resources, for example: memory, CPU, network, and file resources.

Authentication and Authorization

When a user sends an unauthenticated request to the central entry point of an XS advanced application such as the XS advanced Application Router (or any other mashup service), the user is redirected to the XS advanced User Account and Authentication (UAA) service. The UAA can handle different types of authentication methods, for example: basic (user name and password) and SAML assertions. The UAA does not store user credentials itself; it relies on an external user store, for example, SAP Cloud Identity or the SAP HANA database. Upon successful login, the UAA issues an OAuth token that can then be forwarded in all calls to further application services to authenticate that user. This way, application services only need to

support one single authentication method: OAuth tokens. By contrast, authentication at backing services like databases is based on technical users that are authenticated with user credentials (name and password) provided to the application in its environment. The user's OAuth token can also be forwarded to a backing service such as the SAP HANA database for user-based authorizations.

OAuth tokens not only provide a unified authentication method, they also contain the user's authorization in the form of a set of permissions, called "scopes". Authorization scopes are managed in the UAA and can be derived for example from SAML roles or attributes. Based on these scopes, applications perform functional authorization checks on all service layers, either modeled or implemented in code. Instance-based authorization is based on attributes that can be included in the OAuth token but, more often, are managed by the XS advanced application itself.

Component Model

In a microservice architecture such as the one provided with XS advanced, the following rules apply:

- Service life cycle
  Each service has its own independent life cycle and must provide stable service interfaces for robust integration with other services that may be upgraded or even completely replaced at different times.
- Service isolation and independence
  Services are isolated from each other and share nothing. If common frameworks or libraries need to be shared, they are embedded redundantly (and possible in different versions) in each service.
- Service language and run time
  Services can be implemented in any available language and run in any available run time environment, independently of the languages and run-time environments used to implement other services.
- Service extensions
  Services can provide extensions to other services, either by wrapping or replacing existing services with extended implementations or implementing predefined extension point service interfaces. Additionally, new applications can be aggregated from existing and new services by combining them in a "mashup" service.

To facilitate the deployment of groups of services which in their specific combination constitute a business application that is managed as a whole, XS advanced introduces the concept of Multi-Target-Applications (MTAs) that can be deployed in one step and share the same life cycle but otherwise retain their independence. This approach brings an important operations benefit that is also supported in the Cloud.

Client User Interface

The UI model assumes a rich client that communicates via HTTP with application services in the Cloud (for example using the OData protocol for UI5 based user interfaces). The SAP standard UI technologies, Fiori and UI5, follow this model. To overcome the same-origin policy restriction implemented in Web browsers, the UI client connects to a central entry point such as the XS advanced Application Router or a mashup service that acts as reverse proxy and routes inbound requests to the appropriate XS advanced application service. Native clients that are not subject to the restrictions of a Web browser should follow this model to simplify configuration, hide internal implementation details, and provide central login (and logout) services.

The same reasons (of simplicity, implementation details, and central login services) also apply when deciding whether to use Cross-Origin Resource Sharing (CORS) in the browser as an alternative solution to the same-

origin policy restriction. Fiori applications can be served by a central XS advanced Application Router that reads UI content and routing configuration from the layered repository (LRep) backing service. As a result, Fiori applications do not need to deploy a dedicated XS advanced Application Router; they just have to deploy their Fiori content to the shared LRep. By contrast, plain (non-Fiori) UI5 applications embed their UI content in their own XS advanced Application Router which also acts as a Web server for content packaged with its own container; these UI5 applications do not rely on a backing service as an external content repository.

OData Services

To support OData, the SAP-recommended protocol for REST-based data access, XS advanced provides a selection of OData server libraries for the supported run-time environments, for example, Java and JavaScript (Node.js). XS advanced applications can implement own data providers based on these OData libraries. However, more typically, XS advanced applications simply model the database objects that must be exposed to client requests by annotating CDS views or defining `xsodata` metadata artifacts; the applications can then make use of the generic data providers that interpret these models. It is also possible to use a combination of these two approaches, for example, where the generic data providers perform the data-intensive tasks, and application data providers exit to perform for example data validations or transformations.

SAP HANA Database

As Extended Application Services (XS) to the HANA database, XS advanced provides support for SAP HANA as the database backing service.

- SAP HANA Service Broker [page 19]
- HDI Container [page 20]
- HDI Deployer [page 20]
- SAP HANA Instance Broker [page 20]
- Synonyms [page 20]
- Core Data Services [page 21]

SAP HANA Service Broker

Service Brokers are the components that manage backing services. An application that requires a particular instance of a backing service defines this dependency, and the platform interacts with the service-specific Service Broker to create a dedicated instance for the application and binds the application to it.

The SAP HANA Service Broker provides various types of service instances according to the service plan the application requires. First and foremost, these are HDI containers; however, plain schemata are also possible. Both types of service plan provide isolation in a shared database so that XS advanced applications do not get exclusive access to a complete SAP HANA system; the applications have access to a container which allows multiple applications (or different versions of one application) to share an SAP HANA system.

When an application is started, the platform passes the connection parameters and credentials of the container to the XS advanced application in its environment. For this reason, applications do not need to hard-code the names of any schema or provide access to global tables; XS advanced applications are isolated within their assigned container.

Multiple applications within one MTA may share a container if they share the same life cycle. However, different MTAs that have different life cycles are assigned different containers.

HDI Container

HDI Containers are a special type of database schemata that manage their contained database objects. These objects are described in design-time artifacts that are deployed by the HDI Deployer application. HDI takes care of dependency management and determines the order of activation; HDI also provides support for upgrading existing run-time artifacts when their corresponding design-time artifacts are modified.

Applications that use other persistency frameworks are obliged to fall back on the use of plain schemata. These frameworks can then directly execute DDL statements, something that is not allowed in HDI Containers, where the management of database objects is performed exclusively by HDI.

HDI Deployer

The HDI Deployer is an application that is included in an MTA and is used to deploy HDI design-time artifacts to the respective HDI containers. When an MTA is deployed, the HDI Deployer application is pushed first so that it can "prepare" the SAP HANA persistence; by the time defined services are started in XS advanced, the HDI container is ready for use.

The same pattern used for the HDI Deployer is also applied with other backing services in XS advanced. For example, the SDS Deployer deploys Streaming Analytics artifacts, and the LRep Deployer deploys FLP content to the Layered Repository. The common approach is that dedicated XS advanced applications initialize backing service instances for MTAs and are coordinated by the deployment process as defined in the MTA deployment descriptor.

In addition to the stand-alone deployer application, HDI also offers an API that enables applications to create database objects at run time by generating design-time artifacts within the application and deploying them into their container with the HDI. In this way, it is possible to extend the persistence in run-time authoring scenarios, for example, field extensibility or dynamic rule generation.

SAP HANA Instance Broker

The SAP HANA Instance Broker provides dynamic access to multiple containers (HDI Containers or plain schemata) for applications that implement their multi-tenancy data separation by tenant-specific containers. Whereas the SAP HANA Service Broker provides a static binding of an application to its container, the SAP HANA Instance Broker provides an API for applications to request and obtain the connection parameters and credentials required to access a tenant-specific container.

Synonyms

In XS advanced, applications only connect to their own database container (or in the case of multi-tenancy: their containers); an XS advanced application but cannot connect directly to containers bound to other applications. Explicit schema references are not allowed or even possible because they are dynamically created and managed by the SAP HANA Service Broker. This is in line with the decentralized data management concept in a micro-services architecture.

However, from a performance perspective and also from a data-consistency perspective, replication is not always the best answer. So, the SAP HANA database provides synonyms that enable an XS advanced application to perform a controlled "break out" from its own container to related containers. A database synonym is similar to a symbolic link; from an application's perspective, the access always happens within its own container, but the synonym may link to an object in another container. A container that exposes objects to other containers must explicitly grant access to consuming containers in order to create synonyms. The container exposing objects with synonyms has control over the set of objects which can be accessed.

As a result, the exposing container has the means to define an application programming interface (API) at the database level, for example, by allow-listing only public objects in a database role that is granted to the consuming container.

> **i** Note
>
> The complete setup and deployment of synonyms is done by the HDI Deployer; it is not necessary for application developers to configure this manually.

Core Data Services

For SAP HANA, Core Data Services (CDS) is the preferred language for the modeling of your data persistence objects and, as a result, fully supported by XS advanced. The following rules apply:

- Deployment infrastructure
  With SAP HANA Deployment Infrastructure (HDI) and the HDI Deployer, the CDS models are deployed to HDI "containers"
- Run-time environments
  The language run-time environments included with XS advanced provide client libraries for easy consumption of CDS at run time, for example.
  - JavaScript
    JavaScript Data Services (`node-cds` API) include a native JavaScript client and query builder for Core Data Services (CDS) for node.js on SAP HANA XS Advanced Model; `node-cds` enables programs to consume SAP-HANA-based records as native JavaScript objects. The `node-cds` library supports basic CRUD operations and complex queries on all data types (native SAP HANA and defined in CDS), and covers associations, transactions, and lazy navigation.
  - The Java Data Services (JDS) for XS advanced provides a set of tools that facilitate the consumption of Core Data Services (CDS) artifacts in Java applications. JDS enables you to map existing CDS data models to the corresponding Java Persistence API (JPA) 2.1 data models so that they can be consumed by Java applications.
- Data access
  You can use the CDS Data Control Language (DCL) to model instance-based authorizations XS advanced.

Related Information

SAP HANA Extended Application Services, Advanced Model [page 10]
The XS Advanced Programming Model [page 14]

2.4 XS Advanced Application Development Tools

SAP HANA XS advanced model comes with a selection of development and administration tools.

SAP HANA provides a selection of tools to help in the various phases of the design-time development and run-time administration of Multi-Target Applications (MTA) on XS advanced:

- Development Tools [page 22]
- Administration Tools [page 23]

XS Advanced Development Tools

SAP Web IDE for SAP HANA is the browser-based development environment for applications developed for deployment on SAP HANA XS advanced run-time environments; SAP Web IDE for SAP HANA can be used to develop all layers of an application, including the client user interface (UI), XS advanced server applications, and SAP HANA database content. With SAP Web IDE for SAP HANA you have the tools to help you carry out all parts of the XS advanced application-development process, for example: editing, debugging, testing, modeling, version management, building, and deploying. SAP Web IDE is based on XS advanced and the SAP HANA Deployment Infrastructure (HDI) and uses Git for source code management.

- Code editors
  With syntax suggestions/highlights, syntax and format checking, and graphical visualization
- Debugging tools
  Break points, pause and step-over function, call stack visualization, variable lists and values, etc.
- Unit Test tools
  Visualization of test results, stack trace, and test coverage
- Version Control tools
  Easy lifecycle management of design-time artifacts in XS advanced mode with a bundled Git client and a special integration with Gerrit
- Code visualization tools
  Display an outline of your code for easier navigation

Gerrit for XS Advanced

A Gerrit server is integrated with the XS advanced run-time environment. Gerrit for XS advanced is an optional component of the SAP HANA extended application services, advanced model platform, which can be used to store and manage versions of the source code for XS advanced applications, for example SAPUI5 and JavaScript or Java applications, in Git repositories.

> **i Note**
>
> SAP Gerrit for XS advanced is deprecated as indicated in SAP Note 1589311. However, SAP Note 2889508 describes how you can migrate an existing installation of Gerrit for XS advanced to the open-source version of Gerrit while retaining all repositories and review data, and continuing to use XS advanced user-account-and-authentication service (XSUAA) to manage SAP HANA XS advanced user accounts. For information about how to perform a fresh installation of the open-source version 3.1 Gerrit server for use with XS advanced, see SAP Note 2890964.

Gerrit for XS advanced is attached to the User Account and Authentication (UAA) service in the XS advanced platform. If you are installing XS advanced application servers along with the SAP HANA databases, you need at least one central Git instance running in the SAP HANA landscape and connected to XS advanced. Since Gerrit is intended to run on a separate, dedicated host, it is the only XS advanced component that is not installed with the life-cycle management tool `hdblcm`. Gerrit for XS advanced must be installed separately, and preferably after installing SAP HANA.

> → Tip
>
> You can find the Gerrit for XS advanced documentation at the following URL on the system where you installed Gerrit for XS advanced: `https://<GerritServer>.acme.com:8080/Documentation/index.html`

Database Explorer

The Web-based SAP Web IDE for SAP HANA includes the SAP HANA database explorer that contains features and functions required by both developers and administrators, for example:

- Database browser
  Browse, view, run, and visualize the content of all types of catalog objects, for example: tables, views, stored procedures, functions, synonyms, and so on.
- SQL Console
  Create SQLScript, run SQL commands, visualize objects in text form (procedures, functions, and so on).
- SQL Debugger
  Debug procedures and views in HDI containers. View the call stack, set break points, view and evaluate expressions and variables, and so on.
- Job Log Viewer
  View the logs written by the XS jobs.

XS Advanced Administration Tools

Web-based administration tools for SAP HANA XS advanced are available in the SAP HANA cockpit. The cockpit is an Web-based tool that integrates various tools for administration, monitoring, and software life-cycle management. For XS advanced applications, administrators can make use of the following tools:

- XS Advanced Administration Graphical Tools
- XS Advanced Administration Command-Line Tools

XS Advanced Administration Graphical Tools

The following administration tools are available in SAP HANA Cockpit to help you to maintain and manage the various components of the SAP HANA XS advanced model run-time environment:

- *Application Monitor*
  Monitor the system usage of the applications running in the XS advanced model run-time
- *Application Role Builder*
  Maintain and manage user roles and role collections in SAP HANA.
- *Host Management*
  Manage the SAP HANA hosts that are pinned to SAP HANA XS advanced applications or spaces.

- *Organization and Space Management*
  Create, list, or delete user organizations and spaces in the XS advanced model run time.
- *SAML Identity Providers Configuration*
  Configure SAML Identity providers (IDP) for SAP HANA XS advanced model applications that use SAML assertions as the logon authentication method.
- *SAP HANA Logical Database Configuration*
  Manage SAP HANA tenant database instances for SAP HANA XS advanced model applications.
- *SAP HANA Service Brokers*
  Manage and monitor the SAP HANA service broker used by SAP HANA XS advanced model applications.
- *Trusted Certificates*
  Manage and maintain the certificates used to establish secure and trusted connections between SAP HANA systems and SAP HANA XS advanced applications.
- *User Management*
  Create and manage business users for SAP HANA XS advanced model applications.
- *XS Advanced Audit Logs*
  View all audit logs for any previous operation. Group or filter the audit logs by different criteria such as org, space, user, and application.
- *Job Scheduler Service Dashboard*
  Create, schedule, and manage long running operations jobs in the SAP HANA XS advanced model run-time environment.

XS Advanced Administration Command-Line Tools

SAP HANA provides the following tools to enable you to maintain your XS advanced run-time environment from the command line:

- The `XSA` command

  ```
  XSA {command} [--<options>]
  ```

  A selection of utilities to help you maintain XS advanced run-time instances, for example, enable, disable, or restart XS advanced instances, manage domain certificates, and maintain data privacy.
- The `xs` command

  ```
  xs {command} [<arguments>] [<options>]
  ```

  A selection of utilities to help you update, manage, and maintain not only the applications that are deployed to the XS advanced run-time environment, but also the run-time environment itself, for example, the organizations and spaces, the available services, as well as the users who need access to the XS advanced run-time environment and the permissions that are required to do so.

  > **i** Note
  >
  > For more information about the graphical and command-line tools available for administrators of the XS Advanced model platform, see the *Application Run-Time Services* section of the *SAP HANA Administration Guide* in *Related Information* below.

Related Information

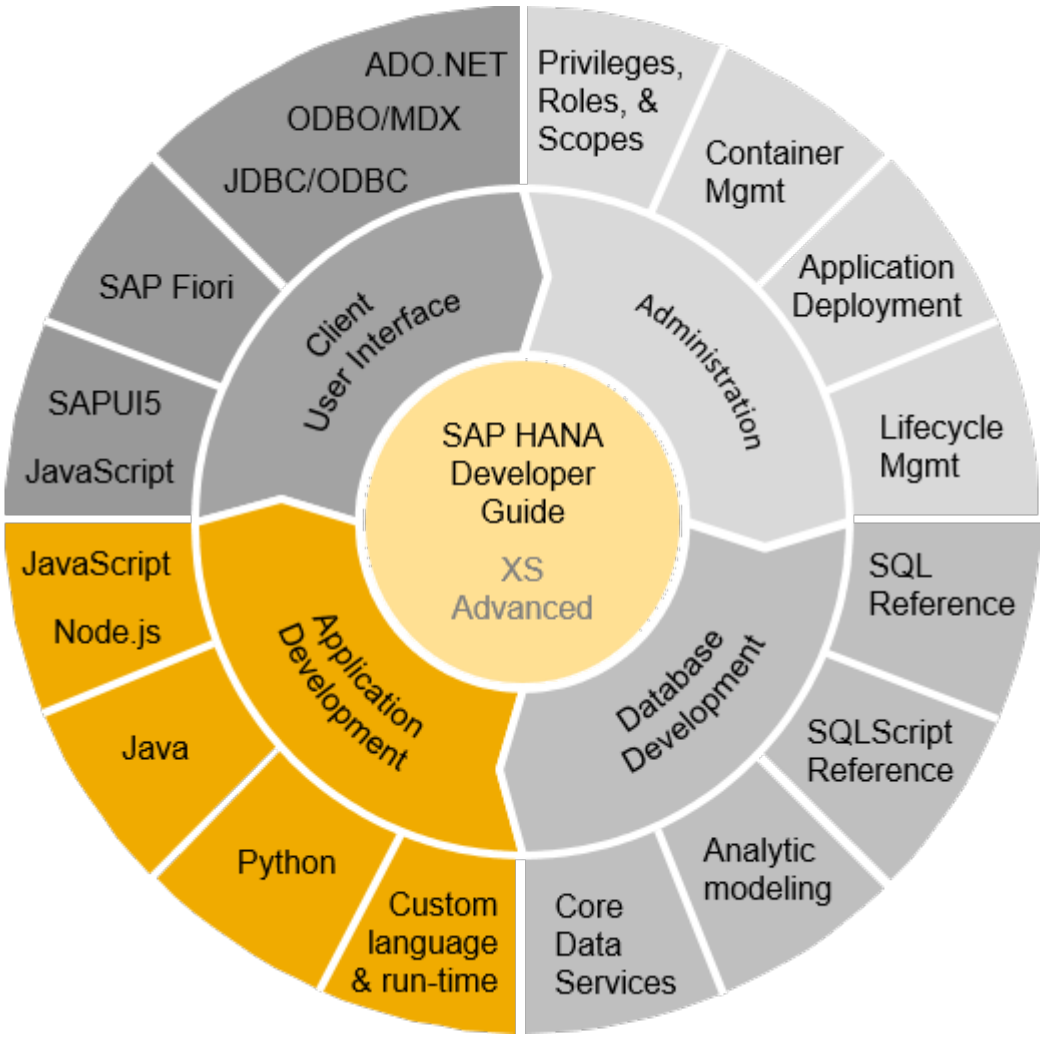
Introduction to Application Development and Deployment (XS Advanced Model) [page 9]
Developer Information Map for XS Advanced [page 25]
SAP HANA Administration Guide

2.5 Developer Information Map for XS Advanced

The developer information road map is designed to help developers find the information they need in the library of user and reference documentation currently available for SAP HANA development projects.

The development environment for SAP HANA supports a wide variety of application-development scenarios. For example, database developers need to be able to build a persistence model or design an analytic model; professional developers want to build enterprise-ready applications; business experts with a development background might like to build a simple server-side, line-of-business application; and application developers need to be able to design and build a client user interface (UI) that displays the data exposed by the data model and business logic. It is also essential to set up the development environment correctly and securely and ensure the efficient management of the various phases of the development lifecycle.

> → Tip
>
> The *SAP HANA Developer Information Map* is available in the section *SAP HANA Platform* on the SAP Help Portal.



Application Development in SAP HANA XS Advanced Model

Related Information

SAP HANA Developer Information Map

Introduction to Application Development and Deployment (XS Advanced Model) [page 9]

3 Getting Started with SAP HANA Development

Deploy a simple application to the XS advanced run-time environment.

The information in this section includes step-by-step tutorials that are designed to show you how to deploy a simple XS advanced or CAP application to the target run-time environment, for example:

- Getting started with SAP HANA XS, advanced model
- Working with the SAP Cloud Application Programming Model (CAP)

Related Information

Getting Started with Application Development in XS Advanced [page 27]
Working with the SAP Cloud Application Programming Model [page 108]

3.1 Getting Started with Application Development in XS Advanced

SAP HANA applications for SAP HANA XS advanced must include a number of mandatory files that are used for configuration and deployment.

To set up your SAP HANA application as a multi-target application, you need to create the necessary source files and place them in the appropriate folders. On deployment, the SAP HANA deployment service reads the information contained in the various configuration files and uses it to deploy the various application components to the corresponding targets.

The following high-level rules need to be followed when setting up your SAP HANA application:

- Separate all database content (design-time representations of tables, views, procedures, etc.)
- Isolate all static content (html, images, other UI related files) in its own folder.
- Place language-specific application source files in their own folder.

> **i Note**
>
> Since xsodata files must be deployed to a JavaScript or Java run-time, place OData service definitions in the application source folders, for example, `javascript/odata/` or `java/odata/`.

- `db/`
  Contains only database artifact, for example: tables, views stored procedures, etc.
- `web/`
  Static HTML resources

- `java/`
  Java source code and any Odata service definitions
- `javascript/`, `js/`, or `xsjs/`
  JavaScript source code and any Odata service definitions; this can include either JavaScript or node.js (or both) in sub-folders

The following example include modules for the following areas: database, Web, JavaScript application code, Odata (optional) services, and security (OAuth2 client configuration).

Example

Sample Code

```
<myAppName>
|- db/                          # Database deployment artifacts
|  \- src/                      # Database artifacts: tables, views, etc.
|- web/                         # Application router/descriptors
|  \- resources/                # Static Web resources
|- js/                          # JavaScript artifacts
|  \- src/                      # JavaScript source code
|     \- odata/                 # OData resources (optional)
|        \- srv/                # OData services
\- security/                    # Security deployment artifacts/scopes/auths
```

Related Information

Set up an XS Advanced Application Project [page 101]
XS Advanced Application Resource Files [page 107]
Working with the XS Advanced Command-Line Client [page 75]
Working with the SAP Web IDE for SAP HANA [page 28]

3.1.1 Working with the SAP Web IDE for SAP HANA

SAP Web IDE for SAP HANA is a browser-based integrated development environment (IDE) that can be used for the development of complex SAP HANA applications.

SAP Web IDE for SAP HANA provides a comprehensive suite of tools which enable the development of complex multi-target applications (MTA) comprising a Web-based or mobile user interface (UI), business logic, and extensive SAP HANA data models. Designed to support developers who use SAP HANA and XS Advanced, SAP Web IDE for SAP HANA provides a variety of tools, for example: syntax-aware editors for the creation of application code and many other SAP HANA artifacts, graphical editors for the design and development of CDS data models and calculation views, as well as a suite of additional tools for inspection, testing and debugging. SAP Web IDE for SAP HANA includes the SAP HANA database explorer, and is tightly integrated with the SAP HANA run-time tools, the SAP HANA deployment infrastructure (HDI) for XS advanced, the tools used for application life-cycle management (ALM), and the XS advanced run-time platform

To facilitate the process of developing multi-target applications for XS advanced, SAP Web IDE for SAP HANA provides the following features and tools:

- An integrated workspace
- A comprehensive suite of the most up-to-date development tools
- Wizards and code templates to help you get started more quickly and easily
- The automatic update in real time of dependencies between the development and build artifacts
- Build, run, and deployment tools

The SAP Web IDE for SAP HANA is designed to guide you through the development process required to deploy multi-target applications; the process includes the following high-level steps:

1. Set up access to the XS advanced development tools, for example, SAP Web IDE for SAP HANA.

   > → Tip
   >
   > Access to the SAP Web IDE for SAP HANA is restricted to those users who have the required privileges; the privileges are defined in a “role” which is included in a “role collection” that is assigned to users.

   To enable access to the development and administration tools included in SAP Web IDE for SAP HANA, some administration setup tasks are necessary after installation of SAP Web IDE for SAP HANA. For more information, see *Related Information* below.
2. Set up an application project.
3. Develop your application modules.
   For example, for your data model, your application business logic, and your client user interface.
4. Build your application modules.
5. Run and debug your application modules.
6. Package the modules into a multi-target application.
7. Deploy the application to SAP HANA XS advanced.

Related Information

Create a Simple "Tiny-World" Application [page 30]
Extend the "Tiny World" Application to Use an OData Service and a Client UI [page 36]
Working with the SAP HANA Database Explorer [page 75]
SAP Web IDE for SAP HANA Reference [page 1060]

3.1.1.1 Create a Simple "Tiny-World" Application

The "tiny-world" application demonstrates a quick way to get an application running in the XS advanced run time.

Prerequisites

- Access to a running SAP HANA system where the XS advanced run time is installed and configured
- Access to the developer tools included in the SAP Web IDE for SAP HANA
  If the SAP Web IDE for SAP HANA is not available at the default URL `https://host.acme.com:53075`, log in to the XS advanced run time with the XS command-line interface (CLI) and use the command `xs app webide --urls` to display the URL required to access the tool.

> ⚠ Caution
>
> The code examples included in this document for XS advanced are often syntactically incomplete; they are intended for illustration purposes only.

Context

For the XS advanced run time, you develop multi-target applications (MTA), which contain modules, for example: a database module, a module for your business logic (Node.js), and a UI module for your client interface (HTML5). The modules enable you to group together in logical subpackages the artifacts that you need for the various elements of your multi-target application. You can deploy the whole package or the individual subpackages.

Procedure

1. Start the SAP Web IDE for SAP HANA.
2. Create a project to develop your tiny-world application.

   An application is created within a context of a project. There are a number of ways to create a project, for example, by importing it or cloning it. For this tutorial, we will create a new project from scratch using a project template.

   a. In the SAP Web IDE for SAP HANA, choose *File* > *New* > *Project from Template*.
   b. Choose the project template type.

      Choose the project template type *Multi-Target Application Project* and choose *Next*.
   c. Type the name **`tinyworld`** for the new MTA project and choose *Next* to confirm.
   d. Specify details of the new MTA project and choose *Next* to confirm.
   e. Create the new MTA project; choose *Finish*.

A project is created with the name "`tinyworld`"; the root folder of the new project contains one essential file, the application's development descriptor `mta.yaml`. The MTA development descriptor defines the prerequisites and dependencies for the deployment of a multi-target application. At this point in time, the file contains only the name and version of the project and an optional short description.

> **→ Tip**
>
> As you add modules and code to your MTA, the SAP Web IDE adds the corresponding information to the relevant part of the `mta.yaml` file.

> **Sample Code**
>
> MTA Development Descriptor (`mta.yaml`)

```
_schema-version: 2.0.0
ID: tinyworld
description: Tinyworld project for GFN
version: 0.0.1
```

3. Add a module for your new MTA's database components.

   The SAP Web IDE for SAP HANA provides Wizards to simplify the process of creating modules for your MTA.

   a. Create a database module.

      Right-click the root folder of your new MTA project and, in the context menu, choose *New* > *SAP HANA Database Module*.

   b. Provide a name for the new database module; type **`tinydb`**

   c. Customize the details of the new database module, if required. In this case, you can accept the default settings and choose *Next* to confirm your settings.

   d. Confirm the details of the new module; choose *Finish*.

   The Wizard creates the new database module with the specified name: `tinydb`. The new module contains the folder `src` for your database artifacts and, in addition, two configuration files: `.hdiconfig` and `.hdinamespace`. The `.hdiconfig` file is mandatory and defines the plug-ins that SAP HANA uses to create a catalog object from a design-time resource; the `.hdinamespace` is optional and defines the rules to use for run-time name spaces.

> **i Note**
>
> By default, the `.hdiconfig` and `.hdinamespace` configuration files are hidden. To display hidden files in the SAP Web IDE for SAP HANA, choose *View* > *Show Hidden Files*.

   In addition, the SAP Web IDE adds details of the new module to the application's central deployment descriptor `mta.yaml`, as illustrated in the following example:

> **Sample Code**
>
> Application Deployment Descriptor `mta.yaml`

```
_schema-version: 2.0.0
ID: tinyworld
description: Tiny world tutorial project
version: 0.0.1
modules:
```

```
- name: tinydb
  type: hdb
  path: tinydb
  requires:
  - name: hdi_tinydb
resources:
- name: hdi_tinydb
  properties:
    hdi-container-name: '${service-name}'
  type: com.sap.xs.hdi-container
```

4. Define the data model for your new MTA; define some simple database content in the new database module.

   In this step we use Core Data Services (CDS) to define a simple database table. CDS is SAP HANA's enhancement to SQL for defining and consuming semantically rich data models.

   a. Navigate to the `src/` folder in your database module `tinydb/`.
   b. Right-click the folder `tinydb/src/` and choose *New* > *HDB CDS Artifact* in the context menu.
   c. Name the new CDS artifact **`tinyf`**.

      The setup Wizard adds the mandatory suffix for CDS artifacts (`.hdbcds`) to the new file name automatically. This suffix is important; it determines how the file is interpreted in the SAP HANA database. In SAP HANA, a file suffix is linked to a specific plug-in, which is used to generate the corresponding catalog object from a design-time artifact.

      > → Tip
      >
      > By default, the new CDS artifact is displayed in the CDS *Graphical Editor*; to open the new artifact in the text-based code editor, right-click the artifact and choose *Open with* > *Code Editor* in the context-sensitive menu.

   d. Enter the CDS code for a simple table (called an “entity” in CDS).

      > Sample Code
      >
      ```
      namespace tinyworld.tinydb;
      context tinyf {
        entity world {
         key continent: String(100);
        };
      };
      ```

      In this step, we defined a simple data-model with a single entity named “world”, with a single field named “continent”.

5. Build the HDB module `tinydb`.

   Building a database module activates the data model and creates corresponding object in the database catalog for each artifact defined in the CDS document. In this case, the build creates one database table called “world”.

   a. In the SAP Web IDE for SAP HANA, select the `tinydb` module and choose *Build* > *Build*.

   If the builder displays the message `(Builder) Build of /tinyworld/tinydb completed` in the SAP Web IDE console, the data-model was successfully activated in a HANA database container, and can now be used to store and retrieve data.

6. Create a new application module to contain the code for your “tiny world” Node.js application.

   The Node.js application module contains the business logic for your “tiny-world” application.

   a. Create a new MTA module for your Node.js code.

      Right-click the `tinyworld/` project folder and choose *New* > *Node.js Module* in the context menu.

   b. Name the new Node.js module **`tinyjs`** and choose *Finish* to complete the setup and close the Wizard.

      > → Remember
      >
      > Check the *Enable xsjs support* check box.

      The setup Wizard creates the new Node.js module `tinyjs` which contains a number of default folders and files, for example, the file `lib/index.xsjs`.

   c. In the Node.js module `tinyjs`, open the file `lib/index.xsjs` and check that it contains the following code:

      > Sample Code
      >
      ```
      $.response.contentType = "text/html";
      $.response.setBody("It's a tiny JS World!");
      ```

   d. Save your changes.

   The SAP Web IDE adds details of the new Node.js module (`tinyjs`) to the application's central deployment descriptor `mta.yaml`, as illustrated in the following example:

   > Sample Code
   >
   > Application Deployment Descriptor `mta.yaml`
   >
   ```
   _schema-version: 2.0.0
   ID: tinyworld
   description: Tiny world tutorial project
   version: 0.0.1
   modules:
   - name: tinydb
     type: hdb
     path: tinydb
     requires:
     - name: hdi_tinydb
   - name: tinyjs
     type: nodejs
     path: tinyjs
     requires:
       - name: tinydb
     provides:
       - name: tinyjs_api
         properties:
           url: '${default-url}'
   resources:
   - name: hdi_tinydb
     type: com.sap.xs.hdi-container
   ```

7. Build the new Node.js application module.

   Right-click the module folder `tinyjs/` and, in the context menu, choose *Run* > *Run as* > *Node.js Application*.

The *Run as Node.js Application* command automatically builds the application, too. You can view the build-run progress in the run console at the bottom of the SAP Web IDE screen; it should look like the following system output:

```
Application: https://host.acme.com:51013 Status: Running [] Stop
```

In addition, if a Web browser does not open automatically and display the page `It's a tiny JS world!`, paste the URL indicated in the system output into your Web browser, for example: **`https://host.acme.com:51013`**.

8. Create an HTML5 module for the client user interface.

   The HTML5 module contains the files for the client UI for your "tiny-world" application.

   a. Create a new MTA module for your HTML5 client code.

      Right-click the `tinyworld/` project folder and choose *New* > *Basic HTML5 Module* in the context menu.

   b. Name the new HTML5 module **`tinyui`** and choose *Finish* to complete the setup and close the Wizard.

      The setup Wizard creates the new HTML5 module `tinyui` which contains a number of default folders and files, for example, the file `resources/index.html`.

   c. In the HTML5 module, open the file `resources/index.html` and check that it contains the following code:

      Sample Code

```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta http-equiv='Content-Type' content='text/html;charset=UTF-8' />
<title>Tiny UI</title>
<!-- 1.) Load SAPUI5 (from a remote server), select theme and control
library -->
<script src="https://sapui5.hana.ondemand.com/resources/sap-ui-core.js"
    id="sap-ui-bootstrap" data-sap-ui-libs="sap.ui.commons,
sap.ui.table, sap.m"
    data-sap-ui-theme="sap_bluecrystal">
</script>
<!-- 2.) Create a UI5 button and place it onto the page -->
<script>
    // create the button instance
    var myButton = new sap.m.Button("btn");
    // set properties, e.g. the text
    myButton.setText("Hello Tiny HTML World!");
    // attach an action to the button's "press" event (use jQuery to
fade out the button)
    myButton.attachPress(function(){$("#btn").fadeOut()});
    // place the button into the HTML element defined below
    myButton.placeAt("uiArea");
</script>
</head>
<body class="sapUiBody">
    <!-- This is where you place the UI5 button -->
    <div id="uiArea"></div>
</body>
</html>
```

The SAP Web IDE adds details of the new HTML 5 module (`tinyui`) to the application's central deployment descriptor `mta.yaml`, as illustrated in the following example:

Sample Code

Application Deployment Descriptor `mta.yaml`

```
_schema-version: 2.0.0
ID: tinyworld
description: Tiny world tutorial project
version: 0.0.1
modules:
- name: tinydb
  type: hdb
  path: tinydb
  requires:
    - name: hdi_tinydb
- name: tinyjs
  type: nodejs
  path: tinyjs
  provides:
    - name: tinyjs_api
      properties:
        url: ${default-url}
- name: tinyui
  type: html5
  path: tinyui
resources:
- name: hdi_tinydb
  properties:
    hdi-container-name: '${service-name}'
  type: com.sap.xs.hdi-container
```

9. Run the new HTML5 application module.

   Right-click the module folder `tinyui/` and, in the context menu, choose *Run* > *Run as* > *Web Application*.

   A Web browser displays the text `Tiny HTML world`. You can view the build-run progress in the run console at the bottom of the SAP Web IDE screen; it should look like the following system output:

   ```
   Application: https://host.acme.com:51013 Status: Running [] Stop
   ```

   In addition, if a Web browser does not open automatically and display the page `Tiny HTML world!`, paste the URL indicated in the console output into your Web browser, for example: **`https://host.acme.com:51013`**.

Results

In this tutorial, we learned how to use the SAP Web IDE for SAP HANA to develop a small (but not very useful) application, consisting of three separate modules: database (`tinydb/`), Node.js (`tinyjs/`), and HTML5 (`tinyui/`). It is important to note that, in this example, each module is a stand-alone component: the business logic does not access the data model, and the UI application does not access the business logic. You can learn how to get MTA modules to interact with each other in other tutorials.

Related Information

Working with the XS Advanced Command-Line Client [page 75]
Extend the "Tiny World" Application to Use an OData Service and a Client UI [page 36]

3.1.1.2 Extend the "Tiny World" Application to Use an OData Service and a Client UI

Extend the simple “Tiny-World” application to enable it to access the SAP HANA database.

Prerequisites

- You have access to a running SAP HANA system where the XS advanced run time is installed and configured
- You have access to the developer tools included in the SAP Web IDE for SAP HANA
  If the SAP Web IDE for SAP HANA is not available at the default URL , for example,`https://host.acme.com:53075`, log in to the XS advanced run time with the XS command-line client (CLI) and use the command `xs app webide --urls` to display the URL required to access the tool.
- You have successfully completed the instructions described in the tutorial *Create a Simple “Tiny-World” Application in XS Advanced*.

> ⚠ Caution
>
> The code examples included in this document for XS advanced are sometimes syntactically incomplete; as a general rule, code examples are intended for illustration purposes only.

Context

In this tutorial, we learn how to use the SAP Web IDE for SAP HANA to build a real SAP-HANA-based business application, with a UI module that accesses the database by means of an Odata service.

Procedure

1. Start the SAP Web IDE for SAP HANA.
2. Open the MTA project “tinyworld” that you created in the tutorial *Create a Simple “Tiny-World” Application*.
3. Open the application module `tinyworld/tinydb` and display the contents of the CDS database artifact `tinyworld/tinydb/src/tinyf`.

Sample Code

```
namespace tinyworld.tinydb;
context tinyf {
  entity world {
   key continent: String(100);
  };
};
```

4. Add an entity (table) to the CDS artifact `tinydb/tinyf`.

   Add the entity definition `country` to the CDS context `tinyf`, as illustrated in **bold text** in the following example:

Sample Code

```
namespace tinyworld.tinydb;
context tinyf {
  entity world {
    key continent: String(100);
  };
  entity country {
    name: String(100);
  };
};
```

5. Add an association between the entities defined in the CDS artifact `tinyf`.

   Add the association `partof` between entity definitions `world` and `country`, as illustrated in **bold text** in the following example:

Sample Code

```
namespace tinyworld.tinydb;
context tinyf {
  entity world {
    key continent: String(100);
  };
  entity country {
    name: String(100);
    partof: Association[0..1] to world;
  };
};
```

6. Build the HDB module `tinydb`.

   Building a database module activates the data model and creates corresponding object in the database catalog for each artifact defined in the CDS document. In this case, the build creates two database tables called “world” and “country”.

   a. In the SAP Web IDE for SAP HANA, right-click `tinydb` and choose *Build*.

   If the builder displays the message `(Builder) Build of /tinyworld/tinydb completed` in the SAP Web IDE console, the data-model was successfully activated in a HANA database container, and can now be used to store and retrieve data.

7. Add the HDI container to the database explorer so that you can view the newly created run-time tables.

   a. Open the database explore, by choosing *Tools* > *Database Explorer*.

b. If prompted, choose to add a database to the database explorer. Otherwise, at the top of the database browser, click *Add database to the Database Explorer*.
c. In the *Add Database* window, choose *HDI Container* for the *Database Type*.
d. In the list of HDI containers, choose the HDI container in which the tables were created.

   The name of the database container with the database artifacts from the `tinyworld` MTA's database module will be something like the following: *your-user-name...tinyworld-hdi-container*.

   Your HDI container is added to the database browser.
e. View the two tables (tinyf.country and tinyf.world), by expanding the HDI container in the tree and clicking *Tables*.
f. Open the SQL console by right-clicking the *Tables* folder in your HDI container tree and choosing *Open SQL Console*.
g. Add the following statements to the SQL console and then choose *Run* from the global tool bar to execute them.

```
INSERT INTO "tinyworld.tinydb::tinyf.world" VALUES ('Africa');
INSERT INTO "tinyworld.tinydb::tinyf.world" VALUES ('Asia');
INSERT INTO "tinyworld.tinydb::tinyf.world" VALUES ('Europe');
INSERT INTO "tinyworld.tinydb::tinyf.country" VALUES ('Ghana', 'Africa');
INSERT INTO "tinyworld.tinydb::tinyf.country" VALUES ('Nigeria',
'Africa');
INSERT INTO "tinyworld.tinydb::tinyf.country" VALUES ('China', 'Asia');
INSERT INTO "tinyworld.tinydb::tinyf.country" VALUES ('Japan', 'Asia');
INSERT INTO "tinyworld.tinydb::tinyf.country" VALUES ('Denmark', 'Europe');
INSERT INTO "tinyworld.tinydb::tinyf.country" VALUES ('Spain', 'Europe');
```

h. Verify that the data was written into the new tables, by executing the following SQL statement.

```
SELECT * FROM "tinyworld.tinydb::tinyf.country" WHERE "partof.continent" =
'Africa'
```

   or...

```
SELECT * FROM "tinyworld.tinydb::tinyf.country" WHERE "partof.continent" =
'Europe'
```

8. Return to the development workspace, by clicking *Development* from the left side bar, and then add a calculation view to the data model.

   Use SAP HANA's graphical modeling tools to add a calculation view; the view should display only European countries.

   a. Navigate to the folder named `src` in the MTA database module `tinydb`.
   b. Create the new calculation view.

      Right-click the database module folder `tinydb/src` and choose *New* > *Calculation View* in the context menu. Name the new calculation view “myview” and choose the *Data Category* “DEFAULT” (or leave this selection empty).
9. Set up the details of the calculation view.

   The graphical calculation view should have the following nodes: *Semantics* > *Aggregation* > *Projection*

→ Tip

In the calculation-view graphical editor, you can change the **type** of node, for example, from *Projection* to *Aggregation* by right-clicking the node and choosing the corresponding option from the context-sensitive menu. For example, right-click a projection node and select *Switch to Aggregation*.

a. Create a projection using the *Create Projection* tool in the graphical editor for calculation views.

   Drag the new projection (*Projection_1*) down into the calculation view palette and drop it below the *Aggregation* node.

b. Connect the *Projection_1* node to the *Aggregation* node.

   Select the (->) icon next to the *Projection_1* node, then drag it (and drop it on) to the *Aggregation* node.

c. Configure the projection.

   Select the *Projection_1* node. In the *Mapping* tab, choose *[+]* to display a search wizard. Type "**`tiny`**" to search for the tables that we created, select *tinyworld.tinydb::tinyf.country* in the search results, and click *Finish*.

   The selected table *tinyf.country* is displayed in the list of *Data sources*.

d. Connect data sources to output columns.

   In the *Mapping* tab, select the table *tinyf.country* and, in the tab toolbar, choose *Add to Output*.

   The selected data sources (*name* and *partof_continent*) are displayed in the list of *Output Columns*.

e. Define a filter expression for your view.

   The filter expresses the rule that defines what the view displays.

   Choose the *FILTER EXPRESSION*" tab, expand the *Columns* element, and double-click the element *partof_continent*. In the *Operators* screen, select the equals icon (=). The expression appears at the top of the dialog screen; complete it by typing the text **`'Europe'`**.

   The complete filter expression should look like the following example:

   Sample Code

   ```
   "partof_continent"='Europe'
   ```

f. Set up the *Aggregation* node.

   Select the *Aggregation* node and, in the *MAPPING* tab, chose *name* in the list of *Data Sources* and drag it (and drop it on) to the *Output Columns* list.

   → Tip

   If no aggregation node exists, the calculation-view graphical editor allows you to change the **type** of node, for example, from *Projection* to *Aggregation* by right-clicking the node and choosing the corresponding option from the context-sensitive menu. Right-click a projection node and select *Switch to Aggregation*.

g. Change the privileges for the view to enable anonymous access.

   Select the *Semantics* node at the top of the view graph, display the *VIEW PROPERTIES* tab, and change the value shown in the *Apply Privileges* field from *SQL Analytical Privileges* to *empty*.

h. Save the changes you made to the calculation view.

i. Rebuild the MTA database module *tinyworld.tinydb*.

In the SAP Web IDE for SAP HANA, right-click the MTA module*tinyworld/tinydb/* and choose *Build* in the context menu.

This creates a view that displays only the names of European countries.

10. Create an OData service to display exposed parts of your data model.

The OData service is defined in one file (with the file extension `.xsodata`) that we will save in the Node.js module (`tinyjs/`), for example, `euro.xsodata`).

a. Create the new OData service.

You define an XS OData service in a design-time file with the file extension "`xsodata`", for example, `service.xsodata`.

In the navigation pane of the SAP Web IDE for SAP HANA, locate the *tinyworld* MTA project and, in the *tinyjs* module, right-click the folder `tinyjs/lib/`, and choose *New* *File*". Name the new OData service file "**`euro.xsodata`**".

Enter the following code into the `euro.xsodata` file and save the file:

Source Code

```
service {
  "tinyworld.tinydb::myview" as "euro" keys generate local "ID";
}
```

The OData service `euro.xsodata` defines an entity "euro" that is based on the calculation view "myview". The "euro" entity uses an auto-generated key named "ID".

b. Create a dependency between the Node.js module `tinyjs/` that exposes the OData service `euro.xsodata` and the underlying database module `tinydb/` on which the OData service depends.

Open the MTA descriptor `mta.yaml` and add the following code (in **bold text** in the following example) to the `tinyjs` module:

Sample Code

```
- name: tinyjs
  type: nodejs
  path: tinyjs/
  requires:
  - name: tinydb
  - name: hdi_container
  provides:
  - name: tinyjs_api
    properties:
      url: ${default-url}
```

c. Enable the application to create a connection to the SAP HANA database.

Open the file `tinyjs/server.js`, and uncomment the line that establishes the connection to the SAP HANA database, as illustrated in the following code example:

Sample Code

```
// configure HANA
try {
```

```
    options = Object.assign(options, xsenv.getServices({ hana: {tag:
"hana"} }));
} catch (err) {
    console.log("[WARN]", err.message);
}
```

Hide (comment out) the section that configures the User Account and Authentication (UAA) service, which is not required yet for this tutorial, as follows:

Sample Code

```
// configure UAA
//try {
// options = Object.assign(options, xsenv.getServices({ uaa: {tag:
"xsuaa"} }));
//} catch (err) {
// console.log("[WARN]", err.message);
//}
```

d. Save all files and run the Node.js module again.

Right-click the module *tinyjs/* and choose *Run* > *Run as Node.js Application* in the context menu.

The *Run as Node.js Application* command automatically builds the application, too. You can view the build-run progress in the run console at the bottom of the SAP Web IDE screen; it should look like the following system output:

```
Application: https://host.acme.com:51013 Status: Running [] Stop
```

In addition, if a Web browser does not open automatically and display the page `It's a tiny JS world!`, paste the URL indicated in the system output into your Web browser, for example: **`https://host.acme.com:51013`**.

11. Test the OData service is available.

Since we did not change the code in the file application start-up file `index.xsjs`, the Web browser still displays the text "It's a tiny JS World!". However, we can use a modified URL to test if the OData service is now running.

In your Web browser, replace the URL element "`/index.xsjs`" with the string "`/euro.xsodata/euro?$format=json`"; the resulting URL should look like the following example;

`https://host.acme.com:51013/euro.xsodata/euro?$format=json`

The OData output formatted as JSON should look like the following example:

Output Code

```
{
  "d": {
       "results": [
          {
            "__metadata": {
            "uri": "https:/host.acme.com:51013/euro.xsodata/
euro('14518283563411')",
            "type": "default.euroType"
            },
            "ID": "14518283563411",
            "name": "Spain"
          },
          {
```

```
            "__metadata": {
            "uri": "https:/host.acme.com:51013/euro.xsodata/
euro('14518283563422')",
            "type": "default.euroType"
            },
            "ID": "14518283563422",
            "name": "Denmark"
          }
       ]
  }
}
```

12. Create a simple SAPUI5 application that uses the OData service.

    The simple SAPUI5 application you create here uses the OData service `euro.xsodata` to extract data from the database and display it in the client interface.

    a. Open the file UI resource file `tinyui/resources/index.html` and replace its contents with the following UI5 code.

    > **i Note**
    >
    > Check that the `data-sap-ui-libs=` list includes the SAPUI5 libraries `sap.ui.commons, sap.ui.table` and, if not, add them.

    > **Sample Code**

    ```
    <!DOCTYPE HTML>
    <head><meta http-equiv="X-UA-Compatible" content="IE=Edge" />
    <title>DevXdemo</title>
    <script src="https://sapui5.hana.ondemand.com/resources/sap-ui-core.js"
    id="sap-ui-bootstrap" data-sap-ui-libs="sap.ui.commons, sap.ui.table,
    sap.m"
     data-sap-ui-theme="sap_bluecrystal">
    </script>
    <script>
     var oModel = new sap.ui.model.odata.ODataModel("/euro.xsodata", true);
     var oTable = new sap.ui.table.Table({ title: "European Countries" });
     oTable.addColumn(new sap.ui.table.Column({ label: "Country Name",
    template: "name" }));
     oTable.setModel(oModel); oTable.bindRows("/euro");
    oTable.placeAt("content");
    </script>
    </head>
    <body class='sapUiBody'>
     <!-- This is where you place the UI5 button
     <div id="uiArea"></div>-->
     <!-- This is where you place the UI5 content -->
     <div id='content'></div>
    </body>
    </html>
    ```

    The code snippet is designed to do the following actions:

    - Define a model (`oModel`)
    - Bind the model `oModel` to the OData service `/euro.xsodata`
    - Create a table with a single column, bound to the "`name`" field of the entity "`/euro`"
    - Initialize the table and display it in an HTML context.

13. Configure the application router to handle cross-module URL requests.

To run the UI5 application in the XS advanced run time, we must set up a destination for the XS advanced application router (“approuter”), which handles cross-references to other URLs. In this case, we define a dependency between the `tinyui` and `tinyjs` application modules and specify a destination route to the OData service `euro.xsodata` defined in the Node.js module “`tinyjs`”.

a. Open the `mta.yaml` file and add the following URL destination to the `tinyui` module.

Sample Code

```
# -- requires tinyjs service URL
  requires:
    - name: tinyjs_api
      group: destinations
      properties:
        name: tinyjs_be
        url: ~{url}
```

b. Add the URL destination for the `tinyjs_api` to the application-router configuration file.

Open the application-router configuration file `tinyui/xs-app.json` and add the following entry to the "`routes`" array (inside the square brackets):

Sample Code

```
{
  "welcomeFile": "index.html",
  "authenticationMethod": "none",
  "routes": [{
     "source": "^/euro.xsodata/.*$",
     "destination": "tinyjs_be"
   }]
}
```

c. Save both files and run the `tinyui` module again.

Right-click the module `tinyui` and choose *Run* > *Run as* > *Web Application* > .

The table *European Countries* is displayed in your Web browser.

Related Information

Working with the SAP Web IDE for SAP HANA [page 28]
Create a Simple "Tiny-World" Application [page 30]
Add Business Logic to the "Tiny World" Application [page 44]

3.1.1.3 Add Business Logic to the "Tiny World" Application

Add business logic to your application so that it is possible to use the application to write, modify, and delete data in the database.

Prerequisites

- You have access to a running SAP HANA system where the XS advanced run time is installed and configured.
- You have access to the XS advanced command-line client.
- You have access to the developer tools included in SAP Web IDE for SAP HANA.
  If SAP Web IDE for SAP HANA is not available at the default URL `https://host.acme.com:53075`, log in to the XS advanced run time and use the command `xs app webide` to display the URL required to access the tool.
- You have successfully completed the instructions described in the following tutorials:
  - *Create a Simple "Tiny World" Application in XS Advanced*
  - *Extend the "Tiny World" Application to use an OData Service and Client UI*

> ⚠ Caution
>
> The code examples included in this document for XS advanced are sometimes syntactically incomplete; as a general rule, code examples are intended for illustration purposes only.

Context

In real-life, productive applications, you can use application logic to complement and extend the built-in database features such as calculation views and SQL procedures. This tutorial shows you how to use stored procedure to add data to the database and, in addition, Node.js and its XS JavaScript compatibility layer to build SAP-HANA-based business logic that extracts and validates data from the database.

Procedure

1. From the SAP Web IDE for SAP HANA, open the MTA project "tinyworld" that you created in the tutorial *Create a Simple "Tiny-World" Application*.
2. Create a stored procedure to enable you to add content to a database table.

   The procedure you create in this step will be used to insert a new country into the database table `tinyworld.tinydb::tinyf.country`, but only if the country does not already exist in the database, and the continent is valid. Otherwise, an error message is displayed.

   a. Create a new `.hdbprocedure` file for your stored procedure.

Right-click the database-module folder `tinydb/src`, choose *New* > *Procedure* from the context menu, and name the new stored procedure **`createCountry`**.

An SQL editor opens for the new stored procedure.

b. Add the following SQLScript code to the procedure file `createCountry.hdbprocedure` and save the file.

Sample Code

```
PROCEDURE "tinyworld.tinydb::createCountry" (
    IN im_country NVARCHAR(100), im_continent NVARCHAR(100),
    OUT ex_error NVARCHAR(100))
    LANGUAGE SQLSCRIPT
    SQL SECURITY INVOKER AS
    --READS SQL DATA AS
BEGIN
declare noc integer;
select count(*) into noc
    from "tinyworld.tinydb::tinyf.world"
    where "continent" = im_continent;
if :noc = 0 then
    ex_error := 'ERROR: Continent ' || :im_continent || ' does not
exist!';
else
    select count(*) into noc
        from "tinyworld.tinydb::tinyf.country"
        where "name" = im_country;
    if :noc > 0 then
        ex_error := 'ERROR: Country ' || :im_country || ' already
exists!';
    else
        insert into "tinyworld.tinydb::tinyf.country"
            values (im_country, im_continent);
    end if;
end if;
END;
```

c. Build the application database module `tinydb` again.

Right-click the application module *tinyworld/tinydb*, choose *Build* > *Build* in the context menu, and follow the build progress in the output displayed in the console pane.

Output Code

```
2:10:51 PM (Builder) Build of /tinyworld/tinydb completed successfully.
```

3. Switch to the database explorer to test the stored procedure `createCountry` by attempting to add some data to your database table `tinyworld.tinydb::tinyf.country`..

a. Choose *Tools* > *Database Explorer*.

By default, the HDI container that was created when you built the database module `tinydb` appears in the database browser. The HDI container is named `<UserName>-<ID>-tinyworld-hdi-container`. If the HDI container is not present, choose *Add Database to the Database Explorer* from the database browser toolbar to open it.

b. Use the database browser to find the procedure `tinyworld.tinydb::createCountry` in the HDI container.

c. Select the `tinyworld.tinydb::createCountry` procedure in the *catalog* list, then right-click the procedure and choose *Generate Call Statement with UI* in the context-sensitive menu.

d. Add a country that already exists in the database.

   Specify **`Spain`** in the *IM_COUNTRY* field and **`Europe`** in the *IM_CONTINENT* field, and then choose *Run* from the global tool bar (or press F8).

   As expected, an error is returned: `ERROR: Country Spain already exists!`

e. Repeat the step to call the procedure and prompt for values, but this time add a country that does **not** exist in the database.

   Specify **`Ireland`** in the *IM_COUNTRY* field, and **`Europe`** in the *IM_CONTINENT* box, and then choose *Run* from the global tool bar (or press F8).

f. Verify that the new country, Ireland, was added to the table *tinyworld.tinydb::tinyf.country*.

   In the database browser, find the table, right-click it, and then choose *Open Data* to view the data in the table.

   There is a now a row in the table for Ireland.

4. Return to the development perspective in SAP Web IDE and create a new JavaScript file called "`country.xsjs`". Place it in the JavaScript application folder `tinyworld/tinyjs/lib/country.`.

   a. Switch to the development perspective by choosing *Tools* > *Development*.
   b. Open the application module `tinyworld/tinyjs` and display the `lib` folder.
   c. In the folder `tinyworld/tinyjs/lib`, create a new subfolder named `country`.

      Right-click the folder `tinyworld/tinyjs/lib`, choose *New* > *Folder* in the context menu, and name the new folder "`country`".

   d. In the folder `tinyworld/tinyjs/lib/country`, create a new JavaScript file called `country.xsjs`.

      Right-click the folder `tinyworld/tinyjs/lib/country/`, choose *New* > *File* in the context menu, and name the new JavaScript `country.xsjs`.

5. In the development perspective, add the business logic to your application.

   Open the new JavaScript file `country.xsjs` and add the following code to it.

   The following code snippet reads the values of two "GET" parameters, "`<name>`" and "`<continent>`" respectively, passes them to a function that calls the SQLScript procedure `createCountry`, which is used to write these values to the "`country`" table, and displays a confirmation message

> **i Note**
>
> To keep things simple, this tutorial uses the HTTP GET method to access the service used to create database entries; productive applications should always use update methods (for example, POST, PUT, or DELETE) to make changes or modifications to the database.

> **Sample Code**
>
> `country.xsjs`

```
function saveCountry(country) {
   var conn = $.hdb.getConnection();
   var output = JSON.stringify(country);
   var fnCreateCountry =
conn.loadProcedure("tinyworld.tinydb::createCountry");
   var result = fnCreateCountry({IM_COUNTRY: country.name, IM_CONTINENT:
country.partof});

   conn.commit();
```

```
    conn.close();
    if (result && result.EX_ERROR != null) { return result.EX_ERROR;}
    else { return output; }
}
var country = {
    name: $.request.parameters.get("name"),
    partof: $.request.parameters.get("continent")
};
// validate the inputs here!
var output = saveCountry(country);
$.response.contentType = "application/json";
$.response.setBody(output);
```

6. In the development perspective, save all files and run the Node.js module again.

   Right-click the module *tinyjs/* and choose *Run* > *Run as Node.js Application* in the context menu.

   The *Run as Node.js Application* command automatically builds the application, too. You can view the build-run progress in the run console at the bottom of the SAP Web IDE screen; it should look like the following system output:

   ```
   Application: https://host.acme.com:51012 Status: Running [] Stop
   ```

   If a Web browser does not open automatically and display the page `It's a tiny JS world!`, paste the URL indicated in the system output into your Web browser, for example: **`https://host.acme.com:51012`**.

7. Use your browser to test that the new XS JavaScript service is available.

   Since we did not change the code in the file application start-up file `index.xsjs`, the Web browser still displays the text "It's a tiny JS World!". However, we can us a modified URL to test if the new XS JavaScript service is up, running, and ready to use.

   In your Web browser, replace the URL element "`/index.xsjs`" with the string "`/country/country.xsjs?name=Zambia&continent=Africa`"; the resulting URL should look like the following example;

   `https://host.acme.com:51012/country/country.xsjs?name=Zambia&continent=Africa`

   The output should look like the following example:

   Output Code

   ```
   {
        "name": "Zambia"
        "partof": "Africa"
   }
   ```

   → Tip

   You can add additional countries to other continents by replacing the appropriate values, for example: `/country/country.xsjs?name=Vietnam&continent=Asia` or `/country/country.xsjs?name=Finland&continent=Europe`

8. Check that the tables have been successfully updated in the catalog and contain the countries you added.

   To view the contents of the HDI container that is bound to your development project, right-click the database module where the design-time tables are stored, and in the context menu choose *Open HDI Container*.

> → Tip
>
> The new tables are listed in the catalog's *Tables* node. Right-click the table `tinyf.country` and choose *Open Data* in the context menu. The countries you added (Zambia, Vietnam, and Finland) should appear in the *name* column alongside their respective continent (Africa, Asia, and Europe) in the adjacent *partof.continent* column.

Related Information

Deploy the "Tiny World" Application [page 52]

3.1.1.3.1 Debug the JavaScript Code in the "Tiny World" Application

Use the built-in debugging tools to find and fix bugs in the JavaScript code.

Prerequisites

- You have access to a running SAP HANA system where the XS advanced run time is installed and configured
- You have installed the XS command-line client on your development machine.
  The XS CLI client tools are installed by default on the SAP HANA server. However, you must download and install the client tools locally, if you want to connect to SAP HANA from your local machine. The XS CLI client tools (`xs.onpremise.runtime.client_<platform>-<version>.zip`) can be downloaded from the SAP HANA server, from the installation DVD, or from the SAP support portal. For more information about the SAP support portal, see the link to SAP note 2242468 in the related links below.
- SAP Node.js packages
  The archive (`xs_javascript-<version>-bundle.tar.gz`) is available on the SAP HANA media and the SAP Marketplace (SMP); it includes a selection of Node.js packages developed by SAP for use with the Node.js applications running in the XS advanced run time.

  > → Tip
  >
  > Alternatively, you can configure NPM to resolve SAP-related NPM dependencies using the public NPM Registry (at `https://registry.npmjs.org`).

- You have access to the developer tools included in the SAP Web IDE for SAP HANA
  If the SAP Web IDE for SAP HANA is not available at the default URL `https://host.acme.com:53075`, in a command shell, log in to the XS advanced run time and use the command `xs app webide` to display the URL required to access the tool.
- You have successfully completed the instructions described in the following tutorials:

- *Create a Simple "Tiny World" Application in XS Advanced*
- *Extend the "Tiny World" Application to use an OData Service and Client UI*
- *Add Business Logic to the "Tiny World" Application*

> ⚠ Caution
>
> The code examples included in this document for XS advanced are sometimes syntactically incomplete; as a general rule, code examples are intended for illustration purposes only.

Context

The built-in debugging tools enable you to perform standard debugging tasks, such as adding breakpoints to the code and starting, stepping through, and resuming code execution. You can also inspect variables and check the validity of expressions.

Procedure

1. Choose *Run* > *Run as* > *Node.js Application* in the tool bar for the Node.js module `tinyjs` in the "Tiny World" application.
2. Choose *View* > *Debugger* in the tool bar, to display the built-in JavaScript debugging tools.
3. Choose *(attach)* in the *(debugger)* pane and select your running Node.js application in the dialog.
4. Set a break point to test the debug mode.

   Open the JavaScript file `tinyjs/lib/country/country.xsjs` and set a break point at line 21 by clicking to the left of the line number displayed in the built-in JavaScript editor.
5. Test the JavaScript code.

   Open a new Web browser (or tab) and append the string "`/country/country.xsjs?name=Andorra&continent=Europe`" to the "`tinyui`" application entry point URL, as illustrated in the following example:

   ```
   https://host.acme.com:51013/country/country.xsjs?name=Andorra&continent=Europe
   ```

   > ℹ Note
   >
   > The debugger displays a notification indicating that the "`tinyui`" application is suspended at the break point you just set.
6. Debug the JavaScript code.

   In the SAP Web IDE for SAP HANA's debug tool you can examine the call stack and see the value assigned to any variable in the function "`saveCountry`". If you want to modify variables in the context of the current breakpoint, you can use the *Debug Console* to type in any valid JavaScript statement, including variable assignments.
7. Resume the debugging process.

Choose *Resume* to complete execution of the JavaScript file.

> **i** Note
>
> If the debugging process takes too long, the Web browser might time out; the *Debugger* tool displays a message indicating that the Websocket for the attached debug session has been closed. However, the operation should have completed.

Related Information

The SAP HANA XS Advanced JavaScript Run Time [page 627]
Run Unit Tests on the "Tiny World" Application's JavaScript Code [page 50]
Add Business Logic to the "Tiny World" Application [page 44]

3.1.1.3.2 Run Unit Tests on the "Tiny World" Application's JavaScript Code

Use the built-in Jasmine-based unit-test tools to check the JavaScript code.

Prerequisites

- You have access to a running SAP HANA system where the XS advanced run time is installed and configured
- You have installed the XS command-line client on your development machine.
  The XS CLI client tools are installed by default on the SAP HANA server. However, you must download and install the client tools locally, if you want to connect to SAP HANA from your local machine. The XS CLI client tools (`xs.onpremise.runtime.client_<platform>-<version>.zip`) can be downloaded from the SAP HANA server, from the installation DVD, or from the SAP support portal. For more information about the SAP support portal, see the link to SAP note 2242468 in the related links below.
- SAP Node.js packages
  Configure NPM to resolve dependencies to Node.js packages developed by SAP by using the public NPM Registry (at `https://registry.npmjs.org`). In the NPM registry configuration file `.npmrc`, add the following line:

  ```
  @sap:registry=https://registry.npmjs.org
  ```

- You have access to the developer tools included in the SAP Web IDE for SAP HANA
  If the SAP Web IDE for SAP HANA is not available at the default URL `https://host.acme.com:53075`, in a command shell, log in to the XS advanced run time and use the command `xs app webide` to display the URL required to access the tool.
- You have successfully completed the instructions described in the following tutorials:
  - *Create a Simple "Tiny World" Application in XS Advanced*

- *Extend the "Tiny World" Application to use an OData Service and Client UI*
- *Add Business Logic to the "Tiny World" Application*

> ⚠ Caution
>
> The code examples included in this document for XS advanced are sometimes syntactically incomplete; as a general rule, code examples are intended for illustration purposes only.

Context

SAP HANA XS classic supports a Jasmine-like unit-test pattern called "XSUnit", which can also be used in XS advanced to develop unit tests for a Node.js application. When you create a Node.js module for your multi-target application (MTA), the SAP Web IDE for SAP HANA creates a new folder named "`test/`", which includes a sample test library named "`sampleTest.xsjslib`". You can use this sample test library to run some simple tests on the Node.js files in the "Tiny World" application.

Procedure

1. Create a new test run-configuration for your Node.js application module `tinyjs`.
   a. Create a new test run-configuration.

      Choose *Run* > *Run Configurations* in the tool bar.
   b. In the *Run Configurations* dialog, choose *[+]*, select *Node.js Test* in the drop-down list.
   c. Type the name **`test`** and choose *Save and Run*.

   XSUnit runs your Node.js module `tinyjs` in XS JavaScript test mode. The test finds the `sampleTest.xsjslib` file because it matches the test file pattern "`*test`" defined in the run configuration you created. The *Node.js Test Execution* pane displays the results of the test.
2. Add a more meaningful function to the unit test.

   Since the original test is not doing anything particularly interesting (`expect(0).toBe(1);`), replace the function `it("not ok"...)` in the test file `sampleText.xsjslib` with something more meaningful, for example, a calculation of a simple sum, as illustrated in **bold text** in the following example:

   > Sample Code

```
describe("sample test suite", function() {
    beforeEach(function() {
    });
    it("add simple", function() {
     $.import("calc", "calculator");
     var calc = $.calc.calculator;
     var sum = calc.add(1, 1);
     expect(sum).toBe(2);
     });
});
```

At this point the test cannot run since neither the test library `calculator.xsjslib` nor the function `add` exists.

3. Add the calculator test code.

   Create a new JavaScript library named `calculator.xsjslib` in the folder `tinyjs/lib/calc`; the completed test library should look like the following example:

   Sample Code

   `tinyjs/lib/calc`

   ```
   function add(a, b) {
       return a+b;
   }
   ```

4. Add more tests as required.

   You can add new tests (for example, as additional `it` functions) or test **suites** (for example,using the `describe` function) either in the same test file (`sampleTest.xsjslib`) or other test libraries in the predefined `test/` folder.

   > **Note**
   >
   > The name of any new test file must contain the string "Test", for example, `myTest.xsjslilb` or `myotherTest.xsjslib`. This ensures that the file name matches the required `*Test` defined in the *Test File Pattern* box in the run configuration.

3.1.1.4 Deploy the "Tiny World" Application

Deploy the "Tiny World" application to the SAP HANA XS advanced run time.

Prerequisites

- You have access to a running SAP HANA system where the XS advanced run time is installed and configured
- You have installed the XS command-line client on your development machine.
  The XS CLI client tools are installed by default on the SAP HANA server. However, you must download and install the client tools locally, if you want to connect to SAP HANA from your local machine. The XS CLI client tools (`xs.onpremise.runtime.client_<platform>-<version>.zip`) can be downloaded from the SAP HANA server, from the installation DVD, or from the SAP support portal. For more information about the SAP support portal, see the link to SAP note 2242468 in the related links below.
- SAP Node.js packages
  The archive (`xs_javascript-<version>-bundle.tar.gz`) is available on the SAP HANA media; it includes a selection of Node.js packages developed by SAP for use with the Node.js applications running in the XS advanced run time.

> → Tip
>
> Alternatively, you can configure NPM to resolve SAP-related NPM dependencies using the latest packages available on the public NPM Registry (at `https://registry.npmjs.org`). For more information, see *Related Information* below.

- You have access to the developer tools included in SAP Web IDE for SAP HANA
  If SAP Web IDE for SAP HANA is not available at the default URL `https://host.acme.com:53075`, in a command shell, log in to the XS advanced run time and use the command `xs app webide` to display the URL required to access the tool.
- You have successfully completed the instructions described in the following tutorials:
  - *Create a Simple "Tiny World" Application in XS Advanced*
  - *Extend the "Tiny World" Application to use an OData Service and Client UI*
  - *Add Business Logic to the "Tiny World" Application*

> ⚠ Caution
>
> The code examples included in this document for XS advanced are sometimes syntactically incomplete; as a general rule, code examples are intended for illustration purposes only.

Context

Application life cycle management is the processes followed by an administrator to transport, install, deploy, maintain and upgrade applications for productive systems. A multi-target applications (MTA) is transported in the form of a (compressed and digitally signed) archive called an MTA archive (MTAR). Deploying an MTA to the XS advanced run time involves deploying each module to its corresponding target run time container, as described in this tutorial..

Procedure

1. Create the MTA archive.
   a. Build the individual application modules in the following order:
      - *tinydb*
        In the project navigation pane, right-click the module *tinydb* and choose *Build* in the context menu.
        ```
        12:44:25 PM (Builder) Build of /tinyworld/tinydb completed successfully.
        ```
      - *tinyjs*
        In the project navigation pane, right-click the module *tinyjs* and choose *Build* in the context menu.
        ```
        12:44:37 PM (Builder) Build of /tinyworld/tinyjs started.
        12:44:39 PM (DIBuild) Build of /tinyworld/tinyjs in progress
        12:45:25 PM (Builder) Build of /tinyworld/tinyjs completed successfully
        ```
      - *tinyui*

In the project navigation pane, right-click the module *tinyui* and choose *Build* in the context menu.

```
12:45:57 PM (Builder) Build of /tinyworld/tinyui completed successfully.
```

b. Specify the application **version**.

The `ID` and `version` properties in the application descriptor file `mta.yaml` define the application version in the production environment.

For the first version of our application, update the value of the `version` property in the `mta.yaml` file to "`1.0.0`", as illustrated below:

Sample Code

```
ID: tinyworld
description: Tiny world tutorial
version: 1.0.0
[...]
```

c. Build the complete "tinyworld" application project.

Right-click the project folder *tinyworld/* and choose *Build* in the context menu. You can follow the progress of the build in the console at the bottom of the details pane in the SAP Web IDE for SAP HANA.

Output Code

```
12:46:03 PM (Builder) Build of /tinyworld started.
12:46:20 PM (DIBuild) ********** Printing /tinyworld Build Log
**********
[INFO] Reading mta.yaml
[INFO] Processing mta.yaml
[INFO] Creating MTA archive
[INFO] Saving MTA archive tinyworld_1.0.0.mtar
********** End of /tinyworld Build Log **********
12:46:20 PM (DIBuild) Build results link: https://
host.acme.com:53075/che/builder/...
12:46:20 PM (Builder) Build of /tinyworld completed successfully.
```

If the build process completes successfully, an MTA archive named `tinyworld_1.0.0.mtar` is created in the root folder of the application project (`mta_archives/tinyworld/`).

d. Download the MTA archive to your local file system.

In the project navigation pane, right-click the MTA archive *tinyworld_1.0.0.mtar* and choose *Export* in the context menu.

2. Deploy the `tinyworld` MTA to your SAP HANA XS advanced run time.

In this tutorial, we use command-line tools to deploy the application to SAP HANA.

As an administrator, use the `xs` command-line tool to log in into XS advanced run time on the target system, organization and space. In this tutorial we will use "Rita" as our administrator, "myorg" as our organization and "PROD" as our space.

a. Log on as an administrator to the XS advanced run time on the target SAP HANA system.

```
xs login -a https://host.acme.com:30030 -u RITA --skip-ssl-validation
```

> **i Note**
>
> You will need to provide a password for RITA and,when prompted, choose the organization and space from the lists, for example, by typing the corresponding number: “0” for “PROD”.

In this example, the administrator user “RITA” has logged on to the XS advanced run-time space “PROD” in the default organization “myorg”.

> **Output Code**

```
xs login -u RITA
Existing spaces:
  0. PROD
  1. SAP
  SPACE> 0
  API endpoint:  https://host.acme.com:30030
  User:          RITA
  Org:           myorg
  Space:         PROD
```

b. Check that the user RITA has the required user permissions in the space “PROD”.

```
xs space-users myorg PROD
```

```
Listing users in space "PROD" of org "myorg" by role ...
role            users
------------------------
SpaceManager    RITA
SpaceDeveloper  RITA
SpaceAuditor
```

> **→ Tip**
>
> If necessary, grant RITA the required permissions:
>
> ```
> xs set-space-role RITA myorg PROD SpaceDeveloper
> ```

c. Check that you are in the correct space for the application deployment.

```
xs target
```

```
API endpoint:   https://host.acme.com:30030 (API version: 1)
User:           RITA
Org:            myorg
Space:          PROD
```

> **→ Tip**
>
> If necessary, switch to the target space for deployment (“PROD”):
>
> ```
> xs target -o myorg –s PROD
> ```

d. Deploy the MTA archive to the target space.

In the command shell, change directory to the location where you saved the MTA archive you want to deploy, then run the following command:

```
xs deploy tinyworld_1.0.0.mtar --use-namespaces
```

Follow the deployment progress in the output displayed in the command shell. On successful completion, the following message is displayed in the console:

Output Code

```
...
 1 of 1 instances running (1 running)
 Application "tinyworld.tinyui" started and available at "https://
host.acme.com:51016"
 Publishing public provided dependencies...
 Registering service URLs...
 Creating service brokers...
Process finished
```

→ Tip

As indicated in the system output, the services required by the deployed application are started automatically, for example: `hdi-container` and `tiny_uaa`.

3. Test the deployed application in a Web browser.
   a. Paste the URL assigned the deployed application into a Web browser; the URL can be copied from the system output in the console.

      `https://host.acme.com:51016`

      The application starts in the Web browser; you can log on using any valid database user, for example, “RITA”.
   b. Display details of the deployed “tinyui” application running in the current space.

      In the command shell, run the following command:

      `xs app tinyworld.tinyui`

```
Showing status and information about "tinyworld.tinyui"
  name:              tinyworld.tinyui
  requested state:   STARTED
  instances:         1
  memory:            <not specified>
  disk:              <not specified>
  buildpack:         <default>
  urls:              https://host.acme.com:51016
Instances of droplet 1 created at Mar 11, 2016 1:41:09 PM
index   created                  state     host           internal port
-------------------------------------------------------------------------
---------
0       Mar 11, 2016 1:41:18 PM   RUNNING   host.acme.com   50012
```

   c. Paste the URL displayed into your Web browser.

      `https://host.acme.com:51016`

      The Web browser displays the database table “European Countries” table.

i Note

The table is empty because we have just deployed this application to a new HDI container, which might not have all the content we added manually in other tutorials.

Related Information

Download and Consume SAP Node.js Packages [page 633]
Add Business Logic to the "Tiny World" Application [page 44]
Upgrade and Redeploy the "Tiny World" Application [page 57]
The XS Command-Line Interface Reference

3.1.1.5 Upgrade and Redeploy the "Tiny World" Application

Prerequisites

- You have access to a running SAP HANA system where the XS advanced run time is installed and configured
- You have installed the XS command-line client on your development machine.
  The XS CLI client tools are installed by default on the SAP HANA server. However, you must download and install the client tools locally, if you want to connect to SAP HANA from your local machine. The XS CLI client tools (`xs.onpremise.runtime.client_<platform>-<version>.zip`) can be downloaded from the SAP HANA server, from the installation DVD, or from the SAP support portal. For more information about the SAP support portal, see the link to SAP note 2242468 in the related links below.
- SAP Node.js packages
  The archive (`xs_javascript-<version>-bundle.tar.gz`) is available on the SAP HANA media; it includes a selection of Node.js packages developed by SAP for use with the Node.js applications running in the XS advanced run time.

  > → Tip
  >
  > Alternatively, you can configure NPM to resolve SAP-related NPM dependencies using the latest packages available on the public NPM Registry (at `https://registry.npmjs.org`). For more information, see *Related Information* below.

- You have access to the developer tools included in SAP Web IDE for SAP HANA
  If SAP Web IDE for SAP HANA is not available at the default URL `https://host.acme.com:53075`, in a command shell, log in to the XS advanced run time and use the command `xs app webide` to display the URL required to access the tool.
- You have successfully completed the instructions described in the following tutorials:
  - *Create a Simple "Tiny World" Application in XS Advanced*
  - *Extend the "Tiny World" Application to use an OData Service and Client UI*
  - *Add Business Logic to the "Tiny World" Application*
  - *Deploy the "Tiny World" Application*

> ⚠ Caution
>
> The code examples included in this document for XS advanced are sometimes syntactically incomplete; as a general rule, code examples are intended for illustration purposes only.

Context

After you have successfully deployed the initial version of the `tinyworld` application, you can add a new feature to the application and then upgrade the version of the application running in the production environment by re-deploying the modified application. In this example, the upgrade involves the following modifications:

- A new button in the user interface; the button enables you to add a country to the database table "European Countries" (`tinyworld/tinydb/tinyf.country`).
- A new form for collecting country information in the UI module (`tinyui`); this information will be passed to the `tinyjs` module by means of a "POST" HTTP request.
- A change to Node.js module (`tinyjs`) to process this new API

Procedure

1. Update the content of the file `resources/index.html` in the *tinyui* application module.

   Add the content highlighted in **bold text** in the following example to the file `tinyworld/tinyui/resources/index.html`:

   Sample Code

   ```
   <!DOCTYPE HTML><head><meta http-equiv="X-UA-Compatible"
   content="IE=Edge" />
   <title>DevXdemo</title>
   <script src="https://sapui5.hana.ondemand.com/resources/sap-ui-core.js"
     id="sap-ui-bootstrap" data-sap-ui-libs="sap.ui.commons, sap.ui.table"
   data-sap-ui-theme="sap_bluecrystal">
   </script>
   <script src="./Util.js" > </script>
   <script>
       var oModel = new sap.ui.model.odata.ODataModel("/euro.xsodata", true);
       var oTable = new sap.ui.table.Table({
           title: "European Countries", visibleRowCount: 5, id: "tinytab",
           toolbar: new sap.ui.commons.Toolbar({items: [
           new sap.ui.commons.Button({text: "Add Country",
   press:openFirstDialog})
       ]})});

   oTable.addColumn(new sap.ui.table.Column({ label: "Country Name",
   template: "name" }));
   oTable.setModel(oModel); oTable.bindRows("/euro");
   oTable.placeAt("content");

   </script>
   </head>
   <body class='sapUiBody'> <div id='content'></div> </body> </html>
   ```

   Note

   You create the JavaScript file `Util.js` in the next step.

2. Create a new JavaScript resource named `Util.js` in the folder `tinyui/resources/` folder of the "tinyworld" application project.

a. Right-click the folder `tinyui/resources/` and choose *New* > *File* from the context menu.
b. Type the name **`Util.js`**.
c. Paste the following code into the new JavaScript `Util.js`.

> → Tip
>
> This code creates a small form with two fields and 'OK' button where users can enter a new country name and continent. When the user clicks the 'OK' button we construct and send the POST request to the server.

> Sample Code

```
var oFirstDialog;
function openFirstDialog() {
   if (oFirstDialog) {
      oFirstDialog.open();
   } else {
      oFirstDialog = new sap.ui.commons.Dialog({
         width: "400px", // sap.ui.core.CSSSize
         height: "550px", // sap.ui.core.CSSSize
         title: "Country Details", // string
         applyContentPadding: true, // boolean
         modal: true, // boolean
         content: [new sap.ui.commons.form.SimpleForm({
            content: [
               new sap.ui.core.Title({ text: "Country Name" }),
               new sap.ui.commons.Label({ text: "name"}),
               new sap.ui.commons.TextField({ value: "", id: "name" }),
               new sap.ui.commons.Label({ text: "partof" }),
               new sap.ui.commons.TextField({ value: "", id: "partof" })
            ]
         })] // sap.ui.core.Control
      });
      oFirstDialog.addButton(new sap.ui.commons.Button({
         text: "OK",
         press: function() {
            var name = sap.ui.getCore().byId("name").getValue();
            var partof = sap.ui.getCore().byId("partof").getValue();
            var payload = {};
            payload.name = name;
            payload.partof = partof;
            var insertdata = JSON.stringify(payload);

            $.ajax({
               type: "POST",
               url: "country/country.xsjs",
               contentType: "application/json",
               data: insertdata,
               dataType: "json",
               crossDomain: true,
               success: function(data) {
                  oFirstDialog.close();

sap.ui.getCore().byId("tinytab").getModel().refresh(true);
                  alert("Data inserted successfully");
               },
               error: function(data) {
                  var message = JSON.stringify(data);
                  alert(message);
               }
            });
         }
      }));
      oFirstDialog.open();
```

```
    }
}
```

3. Add a new route to the application descriptor: the configuration file for the application router.

   Open the file `tinyui/xs-app.json` and add the code highlighted in **bold text** in the following example.

   Sample Code

```
{
    welcomeFile": "index.html",
    "authenticationMethod": "none",
    "routes": [{
         "source": "^/euro.xsodata/.*$",
         "destination": "tinyjs_be"
     }, {
         "source": ".*\\.xsjs",
         "destination": "tinyjs_be"
     }]
}
```

4. Make a small modification to the JavaScript `tinyjs/lib/country/country.xsjs`.

   The modification (highlighted in **bold text** in the following example) fetches country data from the request body instead of URL parameters (as expected by the HTTP POST method) and return the status “success” or “failure”:

   Open the file `country.xsjs` and add the code highlighted in **bold text** in the following example.

   Sample Code

```
function saveCountry(country) {
   var conn = $.hdb.getConnection();
   var output = JSON.stringify(country);
   var fnCreateCountry =
conn.loadProcedure("tinyworld.tinydb::createCountry");
   var result = fnCreateCountry({IM_COUNTRY: country.name, IM_CONTINENT:
country.partof});
   conn.commit();
   conn.close();
   if (result && result.EX_ERROR !== null) {
      return {body : result,
      status: $.net.http.BAD_REQUEST};

   } else {
      return {body : output,
      status: $.net.http.CREATED};
   }
}
var body = $.request.body.asString();
var country = JSON.parse(body);
// validate the inputs here!
var output = saveCountry(country);
$.response.contentType = "application/json";
$.response.setBody(output.body);
$.response.status = output.status;
```

5. Rebuild the application modules.

   a. Build the database application module `tinydb/`.

      Right-click the module *tinydb/* and choose *Build* in the context menu.

b. Run the Node.js application module `tinyjs/`.

   Right-click the module *tinyjs/* and choose *Run* > *Run as Node.js Application* in the context menu.

c. Run the UI5 application module `tinyui/`.

   Right-click the module `tinyui` and choose *Run* > *Run as* > *Web Application* > .

6. Start the application in your Web browser.

   Paste the application URL specified at the end of the `tinyui` build run in into the Web browser, for example:

```
Run tinyworld/tinyui
Application: https://host.acme.com:51013
```

   A new button *Add Country* is displayed in the application user interface: it is positioned beneath the table title *European Countries*.

7. Test the new *Add Country* feature.

   Choose *Add Country* and, in the *Country Details* dialog, type the name of a country (**`Ireland`**) and continent (**`Europe`**), and choose *OK*.

   The country "Ireland" appears in the database table *European Countries*.

8. Update the application version to reflect the changes you have made.

   The `ID` and `version` properties in the application descriptor file `mta.yaml` define the application version in the production environment.

   For the updated version of the "Tiny World" application, update the value of the `version` property in the `mta.yaml` file to "`2.0.0`", as illustrated below:

   Sample Code

```
ID: tinyworld
description: Tiny world tutorial
version: 2.0.0
[...]
```

9. Create the MTA archive.

   a. Build the individual application modules in the following order:

      - *tinydb*

        In the project navigation pane, right-click the module *tinydb* and choose *Build* in the context menu.

      - *tinyjs*

        In the project navigation pane, right-click the module *tinyjs* and choose *Build* in the context menu.

      - *tinyui*

        In the project navigation pane, right-click the module *tinyui* and choose *Build* in the context menu.

   b. Build the complete "tinyworld" application project.

Right-click the project folder *tinyworld/* and choose *Build* in the context menu. You can follow the progress of the build in the console at the bottom of the details pane in the SAP Web IDE for SAP HANA.

> Output Code

```
12:46:03 PM (Builder) Build of /tinyworld started.
12:46:20 PM (DIBuild) ********** Printing /tinyworld Build Log
**********
[INFO] Reading mta.yaml
[INFO] Processing mta.yaml
[INFO] Creating MTA archive
[INFO] Saving MTA archive tinyworld_2.0.0.mtar
********** End of /tinyworld Build Log **********
12:46:20 PM (DIBuild) Build results link: https://
host.acme.com:53075/che/builder/...
12:46:20 PM (Builder) Build of /tinyworld completed successfully.
```

If the build process completes successfully, an MTA archive named `tinyworld_2.0.0.mtar` is created in the root folder of the application project (`mta_archives/tinyworld/`).

c. Download the MTA archive to your local file system.

In the project navigation pane, right-click the MTA archive *tinyworld_2.0.0.mtar* and choose *Export* in the context menu.

10. Deploy the `tinyworld` MTA to your SAP HANA XS advanced run time.

In this tutorial, we use command-line tools to deploy the application to SAP HANA.

As an administrator, use the `xs` command-line tool to log in into XS advanced run time on the target system, organization and space. In this tutorial we will use “RITA” as our administrator, “myorg” as our organization and “PROD” as our space.

a. Log on as an administrator to the XS advanced run time on the target SAP HANA system.

```
xs login -a https://host.acme.com:30030 -u RITA -o myorg -s PROD --skip-ssl-validation
```

In this example, the administrator user “RITA” has logged on to the XS advanced run-time space “PROD” in the default organization “myorg”.

> → Tip
>
> To display details of the space you are logged into, use the `xs target` command.

b. Deploy the MTA archive to the target space.

In the command shell, change directory to the location where you saved the MTA archive you want to deploy, then run the following command:

```
xs deploy tinyworld_2.0.0.mtar --use-namespaces
```

Follow the deployment progress in the output displayed in the command shell. On successful completion, the following message is displayed in the console:

> Output Code

```
...
 1 of 1 instances running (1 running)
```

```
 Application "tinyworld.tinyui" started and available at "https://
host.acme.com:51016"
 Publishing public provided dependencies...
 Registering service URLs...
 Creating service brokers...
Process finished
```

→ Tip

As indicated in the system output, the services required by the deployed application are started automatically, for example: `hdi-container` and `tiny_uaa`.

11. Test the deployed application in a Web browser.
    a. Display a list of the applications running in the current space.

       In the command shell, run the following command:

       ```
       xs apps
       ```

       ```
       Getting apps in org "orgname" / space "PROD" as RITA...
       Found apps:
       name                                requested state  instances   memory
       ----------------------------------------------------------------------------
       --
       deploy-service                      STARTED          1/1         288 MB
       product-installer                   STARTED          1/1         512 MB
       hrtt-service                        STARTED          1/1         512 MB
       hrtt-core                           STARTED          1/1         512 MB
       di-core-db                          STOPPED          0/1         256 MB
       di-local-npm-registry               STARTED          1/1         <not
       spe..
       di-core                             STARTED          1/1         512 MB
       di-builder                          STARTED          1/1         640 MB
       di-runner                           STARTED          1/1         512 MB
       webide                              STARTED          1/1         1.00 GB
       RITA-j5d0r0x3h-tinyworld-tinyjs     STARTED          1/1         1.00 MB
       RITA-j5d0r0x3h-tinyworld-tinyui     STARTED          1/1         1.00 GB
       ```

    b. Display details of the “tinyui” application running in the current space.

       In the command shell, run the following command:

       ```
       xs app RITA-j5d0r0x3ukcz1y6h-tinyworld-tinyui --urls
       ```

       ```
       https://host.acme.com:51013
       ```

    c. Paste the URL displayed into a Web browser.

       ```
       https://host.acme.com:51013
       ```

       The Web browser displays the database table “European Countries” table and the new UI elements provided with the upgrade.

Related Information

Download and Consume SAP Node.js Packages [page 633]
Deploy the "Tiny World" Application [page 52]
Add User Authentication to the "Tiny World" Application [page 64]

The XS Command-Line Interface Reference

3.1.1.6 Add User Authentication to the "Tiny World" Application

Configure an application to require user authentication before responding to user requests.

Prerequisites

- You have access to a running SAP HANA system where the XS advanced run time is installed and configured
- You have access to the developer tools included in SAP Web IDE for SAP HANA
  If SAP Web IDE for SAP HANA is not available at the default URL `https://host.acme.com:53075`, in a command shell, log in to the XS advanced run time and use the command `xs app webide` to display the URL required to access the tool.
- You have successfully completed the instructions described in the following tutorials:
  - *Create a Simple "Tiny World" Application in XS Advanced*
  - *Extend the "Tiny World" Application to use an OData Service and Client UI*
  - *Add Business Logic to the "Tiny World" Application*
  - *Deploy and Upgrade the "Tiny World" Application*

> ⚠ Caution
>
> The code examples included in this document for XS advanced are sometimes syntactically incomplete; as a general rule, code examples are intended for illustration purposes only.

Context

Authentication forces the users of an application to identify themselves, and only those users with the required authorization will be allowed access. Authentication and authorization are managed using the XS advanced User Account and Authentication (UAA) service. XS UAA is an OAuth authorization service that provides support for SAP HANA and SAML2 authentication. The configuration steps performed in this tutorial require changes to the following design-time artifacts:

- `tinyjs/server.js`
  The startup file for the "Tiny World" JavaScript application.
- `tinyui/xs-app.json`
  The application descriptor for the "Tiny World" JavaScript application; the "application descriptor" is required to configure an MTA's Node.js application router package.
- `mta.yaml`
  The "Tiny World" JavaScript application's deployment descriptor, which specifies the dependencies required to ensure successful deployment to the XS advanced JavaScript run time.

Procedure

1. Start the SAP Web IDE for SAP HANA.
2. Open the MTA project “tinyworld” that you created in the tutorial *Create a Simple “Tiny-World” Application*.

   > **i** Note
   >
   > It is essential that you have also completed the steps required to extend the “Tiny World” application to use an OData service and JavaScript business logic.

3. Enable authentication for the Node.js module (`tinyjs/`).
   a. Configure the `tinyworld` MTA application's JavaScript module (`tinyjs/`) to use authentication.
   b. Open the JavaScript file `tinyjs/server.js` and uncomment the following line to configure authentication with the XS advanced UAA service.:

   ```
   // configure UAA
   try {
       options = Object.assign(options, xsenv.getServices({ uaa: {tag:
   "xsuaa"} }));
   } catch (err) {
       console.log("[WARN]", err.message);
   }
   ```

   c. Disable anonymous access to the “Tiny World” application.

   Comment the following line in the file `server.js` (or change the configured value from “`true`” to “`false`”).

   ```
   var options = xsjs.extend({
   // anonymous: true, // remove to authenticate calls
      redirectUrl: "/index.xsjs"
   });
   ```

4. Enable authentication for the HTML5 module (`tinyui/`).

   Configure the `tinyworld` MTA application's HTML5 module (`tinyui/`) to use authentication. This configuration is made in the application-router descriptor `xs-app.json`.

   a. Open the application-router configuration file `tinyui/xs-app.json` and define the authentication method; set the attribute `authenticationMethod` to “`route`”, as illustrated in the following example:

   ```
   "authenticationMethod": "route",
   ```

   In this way, the authentication method is defined in the application route itself; by default, this is “ xsuaa”, the XS advanced user account and authentication service. When logging on to the application, the user is redirected to the UAA's default logon form.

5. Add protection against cross-site request forgery (CSRF).

   CSRF exploits the trust between a Website and a user's Web browser and allows the execution of unauthorized and unwanted commands. When we switch on the authentication for the HTML5 module, the XS Advanced runtime will require any HTTP update request (for example, PUT, POST, or DELETE) to present a valid CSRF token. This is a standard security technique to prevent cross-site request forgery.

   Add the following code to the end of the `resources/Util.js` file in the client UI5 module `tinyui`.

Sample Code

```
$(function() {
  // one time fetch of CSRF token
  $.ajax({
    type: "GET",
    url: "/",
    headers: {"X-Csrf-Token": "Fetch"},
    success: function(res, status, xhr) {
      var sHeaderCsrfToken = "X-Csrf-Token";
      var sCsrfToken = xhr.getResponseHeader(sHeaderCsrfToken);
      // for POST, PUT, and DELETE requests, add the CSRF token to the
header
      $(document).ajaxSend(function(event, jqxhr, settings) {
        if (settings.type==="POST" || settings.type==="PUT" ||
settings.type==="DELETE") {
          jqxhr.setRequestHeader(sHeaderCsrfToken, sCsrfToken);
        }
      });
    }
  });
});
```

6. Create an instance of the XS advanced UAA service, which can be used by the HTML5 and Node.js modules.

   In order to bind the "Tiny World" application modules to the UAA service at run time, we first need to define a new UAA service. The configuration is performed in the application's central deployment descriptor `mta.yaml`, as illustrated in the following steps:

   a. Add a new service instance to the "`resources`" section of the `mta.yaml` application descriptor file, as illustrated in **bold text** in the following example.

      Sample Code

      ```
      resources:
      - name: hdi_tinydb
        properties:
          hdi-container-name: '${service-name}'
        type: com.sap.xs.hdi-container
      - name: tiny_uaa
        type: com.sap.xs.uaa
      ```

   b. Add a new service instance to the "`requires`" section of the `mta.yaml` application descriptor file, as illustrated in **bold text** in the following example.

      Note

      The name of the new UAA service (`tiny_uaa`) must be added to the `requires` section of both the `tinyjs` and `tinyui` modules in the `mta.yaml` file, as illustrated in **bold text** in the following example:

      Sample Code

      ```
      - name: tinyjs
        type: nodejs
        path: tinyjs/
        requires:
        - name: tiny_uaa
        - name: tinydb
      ```

```
    - name: hdi_tinydb
    provides:
    - name: tinyjs_api
      properties:
        url: ${default-url}
  - name: tinyui
    type: html5
    path: tinyui/
    requires:
    - name: tiny_uaa
    - name: tinyjs_api
      group: destinations
      properties:
        name: tinyjs_be
        url: ~{url}
```

c. Ensure that the authentication token acquired at run time by the `tinyui/` module is automatically forwarded to the `tinyjs/` module.

   Add the attribute `forwardAuthToken: true` to the `destinations` property of the `tinyui/` module in the `mta.yaml` application descriptor file, as illustrated in **bold text** in the following example.

   Sample Code

```
- name: tinyui
  type: html5
  path: tinyui/
  requires:
  - name: tiny_uaa
  - name: tinyjs_api
    group: destinations
    properties:
      name: tinyjs_url
      url: ~{service_url}
      forwardAuthToken: true
```

7. Rebuild the application modules `tinyjs/` and `tinyui/`.
   a. Rebuild the application module `tinyjs/`.

      Right-click the module *tinyjs/* and choose *Run* > *Run as Node.js Application* in the context menu.
   b. Rebuild the application module `tinyui/`.

      Right-click the module `tinyui` and choose *Run* > *Run as* > *Web Application*.

Related Information

Add Authorization Checks to the "Tiny World" Application [page 68]

3.1.1.7 Add Authorization Checks to the "Tiny World" Application

Configure an application to check authorizations before responding to user requests.

Prerequisites

- You have access to a running SAP HANA system where the XS advanced run time is installed and configured
- You have access to the developer tools included in SAP Web IDE for SAP HANA
  If SAP Web IDE for SAP HANA is not available at the default URL `https://host.acme.com:53075`, in a command shell, log in to the XS advanced run time and use the command `xs app webide` to display the URL required to access the tool.
- You have successfully completed the instructions described in the following tutorials:
  - *Create a Simple "Tiny World" Application in XS Advanced*
  - *Extend the "Tiny World" Application to use an OData Service and Client UI*
  - *Add Business Logic to the "Tiny World" Application*
  - *Deploy and Upgrade the "Tiny World" Application*
  - *Add User Authentication to the "Tiny World" Application*

> ⚠ Caution
>
> The code examples included in this document for XS advanced are sometimes syntactically incomplete; as a general rule, code examples are intended for illustration purposes only.

Context

To enable authorization in an applications (such as `tinyui`), it is necessary to specify the scope a user must be assigned to be allowed to access the resources available at a specific route. The assignment of scopes to routes is configured in the application descriptor (`xs-app.json`) file, which is located under the `tinyui/` MTA application folder. To impose an authorization check on any request to the Tiny World application, add an additional general route, and assign the appropriate scope to it, as follows:

Procedure

1. Start the SAP Web IDE for SAP HANA.
2. Open the MTA project "tinyworld" that you created in the tutorial *Create a Simple "Tiny-World" Application*.
3. Define authorization scopes and role-templates.

In XS advanced, the security descriptor `xs-security.json` is a configuration file that defines the authorization scopes used by the application and the role-templates that are used to build functional user roles.

For the purposes of our “Tiny World” application, the following security descriptor defines one scope to only view data (`$XSAPPNAME.view`, and another scope to create data (`$XSAPPNAME.create`) as well as the corresponding role-templates: `tinyworldView` and `tinyworldCreate`.

a. Create a security descriptor for your “Tiny World” application.

   The security descriptor must be named `xs-security.json` and must be placed in the root folder of your XS advanced application project.

   Right-click the root folder of your application project, choose *New* > *File* in the context menu, and name the new file `xs-security.json`.

b. Define details of the authorization scopes and role templates to use for the “Tiny World” application.

   Paste the following code into your new `xs-security.json` file.

   Sample Code

```
{
  "xsappname": "tinyworld",
  "scopes": [
    { "name": "$XSAPPNAME.view", "description": "View data" },
    { "name": "$XSAPPNAME.create", "description": "Create data"}
  ],
  "role-templates": [ { "name": "tinyworldView",
                        "description": "Role for viewing data",
                        "scope-references": [ "$XSAPPNAME.view" ] },
                      { "name": "tinyworldCreate",
                        "description": "Role for creating data",
                        "scope-references": [
                                    "$XSAPPNAME.create",
"$XSAPPNAME.view"
                        ]
                      }
                    ]
}
```

   Note

   You use the widely available JSON validation tools to ensure that the contents of the file meet the JSON guidelines and rules.

c. Configure the User Account and Authentication (UAA) service to use the new security configuration defined in the `xs-security.json` file.

   Open the application deployment descriptor (`mta.yaml`) and a parameter to the UAA service (`tiny_uaa`) in the `resources` section; the parameter defines the path to the security descriptor `xs-security.json`, as illustrated in the following example:

   Sample Code

   `mta.yaml` Resources Section

```
resources:
- name: hdi_tinydb
  properties:
    hdi-container-name: '${service-name}'
```

```
    type: com.sap.xs.hdi-container
  - name: tiny_uaa
    type: com.sap.xs.uaa
    parameters:
      path: ./xs-security.json
```

4. Provision the UAA service with the new security configuration.

   Next we need to update our `tiny_uaa` service with the new authorization configuration defined in the `xs-security.json` file.

   > **i Note**
   >
   > This manual step is only required if the SAP Web IDE for SAP HANA does not automatically recognize the change to the security configuration.

   a. Open a command shell and log on to the XS Controller for your XS advanced run time.
   b. Delete the exisiting UAA service used by the "Tiny World" application.

      ```
      xs delete-service tiny_uaa
      ```

   c. Create a new UAA service for the "Tiny World" application.

      ```
      xs create-service xsuaa devuser tiny_uaa -c ./xs-security.json
      ```

      > **i Note**
      >
      > The security configuration uses the authorization scopes and role templates defined in the previous steps. The service plan "`devuser`" is designed for use at development time; it enables multiple developers to create UAA service instances with the same application name in their `xs-security.json` file. In XS advanced, application names must be unique; in the event of a name clash, the UAA service broker adds a suffix to the application name to make every application unique per user, for example, "`tinyworld!u1`" .

5. Add authorization checks to the application.

   To enable user authorization in applications using an application router (for example, "`tinyui`"), it is necessary to specify the authorization scope a user must be assigned to access the resources provided by a specific route. The assignment of scopes to routes is configured in the so-called "application descriptor" (`xs-app.json`), which is located in the `tinyui/` folder.

   a. Open the application descriptor for the "Tiny World" application.
   b. Add (or modify) the application routes.

      We extend the routes for the OData service (`/euro.xsodata`), the business logic change request feature (`.*\\.xsjs`), and the general application route (`"^/(.*)$"`), as illustrated in **bold text** in the following example:

      > **Sample Code**
      >
      > `xs-app.json` Application Routes

      ```
      {
         "welcomeFile": "index.html",
         "authenticationMethod": "route",
         "routes": [{
          "source": "^/euro.xsodata/.*$",
          "destination": "tinyjs_be",
          "authenticationType": "xsuaa",
          "scope": "$XSAPPNAME.view"
      ```

```
    }, {
     "source": ".*\\.xsjs",
     "destination": "tinyjs_be",
     "scope": "$XSAPPNAME.create"
    }, {
     "source": "^/(.*)$",
     "localDir" : "resources",
     "scope" : "$XSAPPNAME.view"
    }]
}
```

As result of this change, a user must be granted the scope "`tinyworld.view`" to **view** the content of the "Tiny World" application.

> **i** Note
>
> To effect these changes, you must re-run the application modules `tinyjs` (JavaScript) and `tinyui` (HTML5).

c. Rebuild the application module `tinyjs/`.

   Right-click the module *tinyjs/* and choose *Run* > *Run as Node.js Application* in the context menu.

d. Rebuild the application module `tinyui/`.

   Right-click the module `tinyui` and choose *Run* > *Run as* > *Web Application*.

Authorization checks are now in place for the application "Tiny World"; you now need to create user roles (from the role templates) and assign them to users.

> **i** Note
>
> You use the application *SAP HANA XS Advanced Cockpit* (adminstration tools) to create user roles and assign them to SAP HANA XS advanced users.

Related Information

Create and Assign User Roles for the "Tiny World" Application [page 72]
Add User Authentication to the "Tiny World" Application [page 64]
Create the Security Descriptor for an XS Advanced Application [page 995]
Configure the XS Advanced Application Router [page 913]

3.1.1.8 Create and Assign User Roles for the "Tiny World" Application

Create user roles from role-templates using the *Application Role Builder*.

Prerequisites

- You have access to a running SAP HANA system where the XS advanced run time is installed and configured
- You have access to the SAP HANA *XS Advanced Administration and Monitoring Tools*
  XS advanced administration tools are available at the following default URL `https://host.acme.com:51017`

  > **! Restriction**
  >
  > Access to the SAP HANA run-time tools requires the administrator permissions defined in the user parameters `XS_RC_XS_AUTHORIZATION_ADMIN` (XS_AUTHORIZATION_ADMIN) and `XS_RC_XS_CONTROLLER_ADMIN` (XS_CONTROLLER_ADMIN).

- You have access to the *User Administration Tools* in the Web-based Development Workbench for SAP HANA XS classic model
  XS classic user-administration tools are available at the following default URL:
  `https://host.acme.com:43<HANA_instance>/sap/hana/ide/security`

  > **! Restriction**
  >
  > Access to the user-administration tools for SAP HANA XS classic model requires the administrator permissions defined in the user role *sap.hana.ide.roles::SecurityAdmin*.

- The application has already been configured to use authorization scopes
- You have successfully completed the instructions described in the following tutorials:
  - *Create a Simple "Tiny World" Application in XS Advanced*
  - *Extend the "Tiny World" Application to use an OData Service and Client UI*
  - *Add Business Logic to the "Tiny World" Application*
  - *Deploy and Upgrade the "Tiny World" Application*
  - *Add User Authentication to the "Tiny World" Application*
  - *Add Authorization Checks to the "Tiny World" Application*

> **⚠ Caution**
>
> The code examples included in this document for XS advanced are sometimes syntactically incomplete; as a general rule, code examples are intended for illustration purposes only.

Context

To enable authorization checks in app-router based applications (such as the Tiny World "`tinyui`" application), you need to specify the authorization "scope" that a user must be assigned to in order to access resources available under a specific route. You also need to create roles based on the role-templates defined in your security configuration. Both the authorization scopes and the role-templates are defined in the security descriptor `xs-security.json`.

Procedure

1. Create roles and assign them to a role collection.

   In XS advanced, you create a user role from the role-template defined in the application's security descriptor (`xs-security.json`) using the *Application Role Builder* included with the *XS Advanced Administration and Monitoring Tools*.

   a. Start the *XS Advanced Administration and Monitoring Tools*.

   > → Tip
   >
   > To display the URL required to start the tool, log on to the XS advanced run time in a command shell and run the following command:
   >
   > ```
   > xs app xsa-admin --urls
   > ```

   b. Start the *Application Role Builder* tool.

   c. Display details of the application for which you want to create roles.

   In the *Applications* list locate and choose *tinyworld!u1*; the default roles *tinyworldView* and *tinyworldCreate* are already displayed in the details pane. We can use these default roles in this tutorial.

2. Create role collections for the "Tiny World" application users.

   In XS advanced, you cannot assign a role to a user; you must assign one or more role collections. Role collections are aggregations of the roles needed by a user to access an application and perform a specific task, for example: **view** content or **create** content. In order to separate the *View* authorizations from the *Create* authorizations, you must create two different role collections: `TinyViewRC` and `TinyCreateRC`, as follows:

   a. Start the *Role Collection* tool.

   b. Create a new role collection for viewing content.

   In the list of role collections, choose *[+]*, type the name **`TinyViewRC`**, and choose *[+] Add Application Role*.

   In the dialog that pops up, use the drop-down menus to enter the required details of the application for which you want to create the role collection:

   - *Application Name*: **`tinyworld`**
   - *Template Name*: **`tinyworldView`**
   - *Application Role*: **`tinyworldView`**

   c. Create a new role collection for creating content.

In the list of role collections, choose *[+]*, type the name **`TinyCreateRC`**, and choose *[+] Add Application Role*.

In the dialog that pops up, use the drop-down menus to enter the required details of the application for which you want to create the role collection:

- *Application Name*: **`tinyworld`**
- *Template Name*: **`tinyworldCreate`**
- *Application Role*: **`tinyworldCreate`**

3. Assign the role collections to the application users.

In this step, you assign the role collections to the application users who want to use the *tinyworld* application. You assign role collections with the *Security* tool included with the *SAP HANA Web-based Development Workbench*.

a. Start the *Security* tool.

XS classic user-administration tools are available at the following default URL:

`https://host.acme.com:43<HANA_instance>/sap/hana/ide/security`

> **!** Restriction
>
> Access to the user-administration tools for SAP HANA XS classic model requires the administrator permissions defined in the user role *sap.hana.ide.roles::SecurityAdmin*.

b. Locate the user to whom you want to assign the role collection.

Expand the *Users* node, and double-click the appropriate user.

c. Assign the role collection to the selected user.

Display the *Application Role Collections* tab, choose *[+]*, select the newly added role collection, and in the drop-down list, choose *TinyViewRC.*. Repeat the operation to add the role collection *TinyCreateRC*. Save the changes and close the tool.

The authorization checks are now in place. To test your recent changes you must log on again to your “Tiny World” application; this action creates a new security token containing the new authorization scopes assigned to your user. Since there is no log-out option in our simple application, restart the Web browser, or open a new incognito session.

Related Information

Add Authorization Checks to the "Tiny World" Application [page 68]
Create the Security Descriptor for an XS Advanced Application [page 995]

3.1.1.9 Working with the SAP HANA Database Explorer

The SAP HANA database explorer provides a set of run-time tools that enable access to objects in HDI containers.

Use the database explorer to view and interact with HDI created content. After you've built your HDI containers, open the database explorer to verify that your objects were built correctly, debug your procedures, and query the database.

The database explorer includes the following tools:

A catalog browser	Browse, view, run, and visualize the content of all types of catalog objects, for example: tables, views, stored procedures, functions, and synonyms.
An SQL console	Create SQLScript, run SQL statements and visualize objects in text form (such as, procedures and functions).
An SQL analyzer	View query plans and analyze the performance of SQL queries.
An SQL debugger	View the call stack, set break points, view and evaluate expressions and variables.

From the SAP Web IDE for SAP HANA, start the SAP HANA database explorer by clicking *Tools* > *Database Explorer*.

Related Information

Add HDI Containers and Databases to the SAP HANA Database Explorer (SAP HANA Cockpit)
Managing SAP HDI Containers and Artifacts [page 234]
SAP Web IDE for SAP HANA Reference [page 1060]
SAP Note 2373065

3.1.2 Working with the XS Advanced Command-Line Client

SAP HANA XS advanced provides a set of command-line tools that enable access to (and control of) the SAP HANA XS advanced run-time environment.

The XS CLI client tools are installed by default on the SAP HANA server; they can be used if you have also installed the XS advanced run-time in your SAP HANA instance. However, you must download the client tools and install them locally, if you want to connect to SAP HANA from your local development machine. The package containing the XS CLI client tools (`xs.onpremise.runtime.client_<platform>-<version>.zip`) can be downloaded from the SAP HANA server, from the installation DVD, or from the SAP support portal. For more information about the SAP support portal, see the link to SAP Note 2242468 in *Related Infomration* below.

The XS command-line tools enable you to perform all the steps in the development process which are required to deploy multi-target applications to SAP HANA XS advanced run time; the process includes the following high-level steps:

1. Set up an application project environment (for example: users, organizations and spaces).
2. Develop your application modules.
   For example, for the underlying data model, the application business logic, and even the client user interface.
3. Build and run the individual application modules.
4. Deploy the complete multi-target application to SAP HANA XS advanced.

Related Information

SAP Note 2242468
The XS Command-Line Interface Reference
Get Started with the XS CLI Client [page 76]
Deploy a Node.js Hello-World Application using the XS CLI [page 82]
Deploy a Node.js Application that Uses Database Artifacts and OData Services [page 87]

3.1.2.1 Get Started with the XS CLI Client

Get started with the XS command-line tools that enable access to the SAP HANA XS advanced run-time environment.

Prerequisites

- You have installed the XS command-line client on your development machine.
  The XS CLI client tools are installed by default on the SAP HANA server. However, you must download and install the client tools locally, if you want to connect to SAP HANA from your local machine. The XS CLI client tools (`xs.onpremise.runtime.client_<platform>-<version>.zip`) can be downloaded from the SAP HANA server, from the installation DVD, or from the SAP support portal. For more information about the SAP support portal, see SAP Note 2242468.
- SAP Node.js packages
  You configure NPM to resolve SAP-related NPM dependencies, for example, to Node.js packages developed by SAP for use with the Node.js applications running in the XS advanced run time.
- You have a Git client installed on your development machine.
- You have access to the Internet from your development machine.
- Logon access to the SAP HANA database with the privileges to create SAP HANA users (for example, SYSTEM)

Context

You can use the XS command-line client to perform a wide-variety of developer- and administrator-related tasks. For example, in the role of a developer, you can use the XS CLI to connect to the XS advanced run time installed on the SAP HANA server, log on as a specific user, deploy and manage your applications. If you have the correct role assigned, you can also maintain selected aspects of the run-time space you are assigned to. As an administrator, and with the apropriate roles assigned, you can use the XS CLI to create and manage new users, maintain organizations and spaces, and assign authorizations to the new users in organizations and spaces.

In the following tutorial, you learn how to use the XS CLI to connect to SAP HANA, set up organizations and spaces for use by your development teams, assign roles to users in the organizations and spaces, create services, and bind services to the corresponding applications:

Procedure

1. Check that the XS advanced run time is installed and available on the SAP HANA server.

   Enter the following URL in your Web browser:

   ```
   https://<hana_hostname>:3<instance>30
   ```

   For example, for the SAP HANA instance “00” on the host “xsa.acme.com”:

   ```
   https://xsa.acme.com:30030
   ```

   This displays a Web page indicating that the XS advanced Web server is available.

   > → Tip
   >
   > Adding `/v2/info` to the URL above displays a JSON string in the Web browser, which includes details of a successful connection to the XS advanced controller.

2. Check that the XS client for XS advanced is installed and available.

   The XS client tools are required to connect to the XS advanced run time on SAP HANA and deploy your XS advanced applications. If the XS advanced XS client tools are not installed on your developer machine, you can download the XS client package from the SAP HANA server, the installation DVD, or the SAP Service Portal.

   On your client machine, run the following command in a command shell:

   ```
   $ xs help
   $ xs -v
   ```

3. Connect to the XS advanced controller on the SAP HANA server.

   a. Specify the URL of the API end point on the SAP HANA server you want to connect to:

      ```
      $ xs api https://hostname:3<instanceNr>30
      ```

If this command fails due to a missing SSL certificate, you can download the missing certificate from the SAP HANA server; the certificate `default.root.crt.pem` is typically installed in the following default location:

`<installation_path>/<SID>/xs/controller_data/controller/ssl-pub/router/default.root.crt.pem`

For example, where `<installation_path>=/hana/shared/` and `<SID>`=HDB:

`/hana/shared/HDB/xs/controller_data/controller/ssl-pub/router/default.root.crt.pem`

> → Tip
>
> For temporary testing purposes you can also use the option `--skip-ssl-validation`, but this is not recommended in a production environment.

b. Install the security certificate on the machine where you are running the XS CLI client.

Place the copy of the root certificate in a safe location in your client file system, for example, `/<path>/default.root.crt.pem`

c. Set the API end point for client connections to the SAP HANA server using the newly installed certificate for authentication.

```
$ xs api https://<hostname>:3<instanceNr>30 -cacert /<path>/
default.root.crt.pem
```

4. Log on to the XS advanced run time.

You log on to the SAP HANA instance specified in the API end point set in a previous step. SAP HANA provides a default user *XSA_ADMIN* with administrator permissions; you can use this user ID to test the logon. However, it is recommended to create a new user with more limited permissions, which you can use to log on for developer tasks.

```
$ xs login -u XSA_ADMIN -p <PassWord>
```

> ℹ Note
>
> The password is assigned to the `XSA_ADMIN` user during installation.

In the following example, the default administrator user `XSA_ADMIN` has logged on to the XS advanced run-time space “PROD” in the default organization “initial”.

> ‹≡› Output Code

```
$ xs login -u XSA_ADMIN
  Existing spaces:
  0. PROD
  1. SAP
  SPACE> 0
  API endpoint:  https://host.acme.com:30030
  User:          XSA_ADMIN
  Org:           initial
  Space:         PROD
```

5. Create a new SAP HANA user called “DEVUSER” with the appropriate developer permissions.

Make a note of the initial password.

You can use standard tools such as the SAP HANA studio or the Web-based Workbench to create the new SAP HANA user. To enable access to the XS advanced run time, the new user must be assigned at least the following role collections (listed in the *Role Collections* tab):

- XS_CONTROLLER_USER
  Modify access within the assigned organization or space
- XS_AUTHORIZATION_DISPLAY
  Read-only access to tools

6. Create a new space (for example, called "DEV") for the development team.

> → Tip
>
> In the XS advanced run time, a space is an area that assigned developers can use to develop applications. You must be logged on as administrator (for example, XSA_ADMIN) to create the new run-time space.

```
$ xs create-space DEV
```

7. Create a new organization (for example, called "devteam1") for the development team. The space "DEV" will be assigned to the organization "devteam1".

> → Tip
>
> In the XS advanced run time, an organization is like a team. You must be logged on as administrator (for example, XSA_ADMIN) to create a new run-time organization.

```
$ xs create-org devteam1
```

8. Assign roles to the `DEVUSER` user for the run-time organization ("devteam1") and space ("DEV").

   In this step, you assign the following user roles:

   - Organization role: "OrgManager"
     Create and manage users, and enable features for a specified organization
   - Space role: "SpaceDeveloper"
     Create and manage applications and services, and view logs and reports in the specified space

```
$ xs set-org-role DEVUSER devteam1 OrgManager
$ xs set-space-role DEVUSER devteam1 DEV SpaceDeveloper
```

9. Log on to the SAP HANA XS advanced run time as user "DEVUSER".

```
$ xs login -u DEVUSER
```

> → Tip
>
> If it is necessary to change the password on first logon, start the SAP HANA Web-based Workbench at `hostname.acme.com:80<instanceNr>/sap/hana/xs/formLogin/login.html`, enter the initial and new password and choose *Change Password*.

10. Change the default target space for automatic logon (for example, from "PROD" to "DEV").

All subsequent logons will automatically be routed to the specified organization and space.

```
$ xs target -o devteam1 -s DEV
```

```
$ xs login -u DEVUSER
  API endpoint:  https://host.acme.com:30030
  User:          DEVUSER
  Org:           devteam1
  Space:         DEV
```

11. Log on to SAP HANA XS advanced as the new user “DEVUSER” and test access to the XS CLI.

```
$ xs login -u DEVUSER
$ xs -v
```

12. List all running default services in the organization “devteam1” and space “DEV”.

```
$ xs marketplace
```

The following information should be displayed:

Output Code

```
$ xs marketplace
Getting services from marketplace in org “devteam1” / space “DEV” as
DEVUSER...
OK
service       plans                       description
------------------------------------------------------------------------
fs-storage    free                        Environment variable denotes the
                                          root of client app's file system
hana          hdi-shared, sbss, schema    HDI container on shared HANA db
              securestore
xsuaa         default, devuser, space     XS UAA Service Broker for user
                                          authentication & authorization...
jobscheduler  default                     Job Scheduler on the XS advanced
Platform
```

13. List all manually defined service instances.

```
$ xs services
```

On a freshly installed system, no services have yet been defined, so the following information should be displayed:

Output Code

```
$ xs marketplace
Getting services from marketplace in org “devteam1” / space “DEV” as
DEVUSER...
Found services:
  No services found
```

14. Create an instance of the XS advanced User Account and Authentication (UAA) service.

The following information should be displayed:

Output Code

```
$ xs create-service xsuaa default my-uaa
Creating service “my-uaa”...
OK
```

15. Create an instance of the SAP HANA HDI (Deployment Infrastructure) service.

    The following information should be displayed:

    Output Code

    ```
    $ xs create-service hana hdi-shared my-hdi-container
    Creating service “my-hdi-container”...
    OK
    ```

16. List the newly created services in organization “devteam1” and space “DEV”.

    The following information should be displayed:

    Output Code

    ```
    $ xs services
      Getting services in org "devteam1" / space  "DEV" as DEVUSER
      Found services:
      name              service     plan          Bound apps
      ------------------------------------------------------
      my-uaa            xsuaa       default
      my-hdi-container  hana        hdi-shared
    ```

Related Information

The XS Command-Line Interface Reference

SAP Note SAP Note 2242468: Setting up the XS Advanced CLI

Standard Node.js Packages for XS Advanced [page 635]

Deploy a Node.js Hello-World Application using the XS CLI [page 82]

Deploy a Node.js Application that Uses Database Artifacts and OData Services [page 87]

3.1.2.2 Deploy a Node.js Hello-World Application using the XS CLI

Use command-line tools to deploy a simple Node.js application to the SAP HANA XS advanced run-time environment.

Prerequisites

- The XS advanced run time is installed and available on the SAP HANA server
- You have installed the XS command-line client on your development machine.
  The XS CLI client tools are installed by default on the SAP HANA server. However, you must download and install the client tools locally, if you want to connect to SAP HANA from your local machine. The XS CLI client tools (`xs.onpremise.runtime.client_<platform>-<version>.zip`) can be downloaded from the SAP HANA server, from the installation DVD, or from the SAP support portal. For more information about the SAP support portal, see the link to SAP note 2242468 in the related links below.
- SAP Node.js packages
  You configure NPM to resolve SAP-related NPM dependencies, for example, to Node.js packages developed by SAP for use with the Node.js applications running in the XS advanced run time.
- You have a GIT client installed on your development machine.
- Your development machine has access to the Internet
- The XS advanced services specified in the application manifest files are available, for example:
  - `my-uaa` (user account and authentication)
  - `my-hdi-container` (access to database artifacts in an HDI container).

Context

In the following tutorial, you learn how to use the XS CLI to deploy a simple Node.js application (“`myxsaapp1`”) to the XS advanced-model JavaScript run time; the deployed application displays a “Hello World” page and allows you to run a simple JavaScript service:

Procedure

1. Clone the XS advanced GIT repository "XSA" containing some getting-started Node.js/JavaScript code examples.

   The XS advanced GIT repository "XSA" contains code samples and artifacts for two simple XS advanced JavaScript applications, which you can use to learn how to set up the module structure of the XS advanced application and deploy the application to the XS advanced run time:

   - “myxsaapp1”
     Simple “Hello-World” JavaScript application

- “myxsaapp2”
  JavaScript “Hello-World” application that uses database artifacts and an XS OData service

> ### Output Code
>
> ```
> $ git clone https://github.com/saphanaacademy/XSA.git
> cloning into 'XSA' ...
>     remote: Counting objects: 33, done.
>     remote: Compressing objects: 100% (27/27), done.
>     remote: Total 33 (delta 2), reused 33 (delta 2), pack-reused 0
>     unpacking objects: 100% (33/33), done
>     Checking connectivity... done.
> ```

In the cloned repository on your local machine, you should see the following high-level structure; in this tutorial, we focus on the first of the cloned applications, “`myxsaapp1`”:

> ### Note
>
> The contents of the XSA Git repository might include additional objects that are not required to complete this tutorial.

```
<gitrepo>
|- myxsaapp1/
|   |- web/
|   |- xsjs/
|   \- manifest.yml
|- myxsaapp2/
|   |- db/
|   |- web/
|   |- xsjs/
|   \- manifest.yml
.gitattributes
README.md
```

2. Check the contents of the cloned `myxsaapp1` folder.

   The cloned `myxsaapp1` folder should look like the following example:

```
|- myxsaapp1/
   |- web/
   |   \- resources/
   |      |- .npmrc
   |      |- package.json
   |      \- xs-app.json
   |- xsjs/
   |   |- .npmrc
   |   |- package.json
   |   |- server.js
   |   \- lib/
   |      \-index.xsjs
   \- manifest.yml
```

3. Adapt the application's deployment dependencies defined in the `manifest.yml` file to suit your SAP HANA XS advanced environment.
   a. Check the `myxsaapp1` module names, for example: “`web`” and “`xsjs`”.
   b. Choose a port for the `web/` and `xsjs/` modules.

      You can allow SAP HANA XS advanced to allocate ports automatically, or choose a free port manually, for example, 51510, for `xsjs/`.

> **Note**
>
> Not assigning a port for the `web` application forces XS advanced to assign a port automatically during deployment. After successful deployment, use the command `xs apps web --urls` to display a list of assigned ports.

c. Check the module paths, for example: "`web`" and "`xsjs`". The path must point to the location where module-specific files are stored.

d. Adapt the application's destination in the `web/` module to point to the application's entry point `server.js` with the appropriate host and port.

> **Sample Code**
>
```
[
  {"name":"xsjs-backend", "url":"https://host.acme.com:51510"}
]
```

The `manifest.yml` file for the application "`myxsaapp1`" should look like the following example:

> **Sample Code**
>
```
ID: myxsaapp1
_schema-version: '2.0'
version: 0.0.1
applications:
 - name: xsjs
   type: nodejs
   path: xsjs
   memory: 128M
   port: 51510

 - name: web
   type: html5
   path: web
   memory: 128M
   env:
    destinations: >
     [
      {"name":"xsjs-backend", "url":"https://host.acme.com:51510"}
     ]
```

4. Configure command-line tools to use the NPM Registry to resolve package dependencies.

   The applications `myxsaapp1` and `myxsaapp2` have dependencies to the following packages included in the NPM Registry.

   - `@sap/hdi-deploy`
   - `@sap/approuter`
   - `@sap/xsenv`
   - `@sap/xsjs`

   If you are using command-line tools to build your Node.js applications for XS advanced, you can configure the tools to use a particular NPM registry to resolve the dependencies between resources specified

in the deployment descriptors. To specify the NPM Registry, add the `"@sap:registry="` property to the `.npmrc` resource-configuration file, as illustrated in the following example:

> ### Sample Code
>
> NPM Registry Configuration in the `.npmrc` File
>
> ```
> @sap:registry=https://registry.npmjs.org
> strict-ssl=false
> ```

> ### Note
>
> The `strict-ssl=false` option is only required if XS advanced is behind a firewall.

5. Check the package dependencies for the application's `web` module; the dependencies are defined in the file `myxsaapp1/web/package.json`.

> ### Sample Code
>
> Application Router's Package Dependencies (`myxsaapp1/web/package.json`)
>
> ```
> {
>   "name": "web-approuter",
>   "engines": {
>      "node": ">=4.0.0 <7.0.0"
>   },
>   "dependencies": {
>      "@sap/approuter": "latest"
>   },
>   "scripts": {
>       "start": "node node_modules/@sap/approuter/approuter.js"
>   }
> }
> ```

6. Check the specifications of the application's router configuration; the details are specified in the file `myxsaapp1/web/xs-app.json`.

   Application routes are defined in one or more “destinations”, which specify where files are expected to be found for the relevant application modules, for example, `xsjs`.

> ### Sample Code
>
> Application Router Configuration (`myxsaapp1/web/xs-app.json`)
>
> ```
> {
>     "welcomeFile": "index.html",
>     "authenticationMethod": "none",
>     "routes": [
>         {
>          "source": "^(.*)(.xs(js|odata))(.*)$",
>          "destination": "xsjs-backend"
>         }
>     ]
> }
> ```

7. Deploy the JavaScript application “myxsaapp1” to the XS advanced run time.
   a. Change directory to the application's root folder, which contains the deployment manifest file `manifest.yml`.

b. Deploy the application.

Output Code

```
$ xs push
Using manifest file “manifest.yml”
Creating app “myxsaapp1-web” in org “devteam1” / space “DEV” as DEVUSER
Creating route “myxsaapp1-web.acme.com:51511” in org “devteam1” / space
“DEV” as DEVUSER
Binding route “myxsaapp1-web.acme.com:51511” to app “myxsaapp1-web”
...
  staging app "myxsaapp1-xsjs"
  starting app "myxsaapp1-xsjs"
Showing status and information about app "myxsaapp1-xsjs"
  name:              myxsaapp1-xsjs
  requested state:   STARTED
  instances:         1
  memory:            128 MB
  disk:              <unlimited>
  buildpack:         <default>
  urls:              https://acme.com:51510

Instances:
index    created                     state
-----------------------------------------
0        Dec 25, 2015 10:22:37 AM    RUNNING
```

8. Check which applications are currently running in the XS advanced run time.

Output Code

```
$ xs apps
Getting apps in org "devteam1" / space "DEV" as DEVUSER...
Found apps:

name            requested state   instances   memory   urls
-------------------------------------------------------------------------
-----------------
myxsaapp1-web   STARTED           1/1         128 MB   https://
acme.com:51511
myxsaapp1-xsjs  STARTED           1/1         128 MB   https://
acme.com:51510
```

9. Check the application in a Web browser.

   Specify the entry point for the application router: **`https://acme.com:51511`**

   A logon screen is displayed; log on as DEVUSER.

   You should see the `web/resources/index.html` page displaying the text `Static HTML - Hello World!`.

   Click the “Welcome” link to test the JavaScript service “`index.xsjs`”.

Related Information

Working with the XS Advanced Command-Line Client [page 75]

3.1.2.3 Deploy a Node.js Application that Uses Database Artifacts and OData Services

Use command-line tools to deploy a more advanced Node.js application to the SAP HANA XS advanced run-time environment.

Prerequisites

- The XS advanced run time is installed and available on the SAP HANA server
- You have installed the XS command-line client on your development machine.
  The XS CLI client tools are installed by default on the SAP HANA server. However, you must download and install the client tools locally, if you want to connect to SAP HANA from your local machine. The XS CLI client tools (`xs.onpremise.runtime.client_<platform>-<version>.zip`) can be downloaded from the SAP HANA server, from the installation DVD, or from the SAP support portal. For more information about the SAP support portal, see the link to SAP note 2242468 in the related links below.
- SAP Node.js packages
  You configure NPM to resolve SAP-related NPM dependencies, for example, to Node.js packages developed by SAP for use with the Node.js applications running in the XS advanced run time.
- You have a GIT client installed on your development machine.
- Your development machine has access to the Internet
- The XS advanced services specified in the application manifest files are available, for example:
  - `my-uaa` (access to the XS advanced User Account and Authentication service)
  - `my-hdi-container` (access to database artifacts in the HDI container).

Context

In the following tutorial, you learn how to use the XS CLI to deploy a more advanced Node.js application ("`myxsaapp2`") to the XS advanced-model JavaScript run time; the deployed application uses an HDI container with simple database artifacts, implements some simple business logic to insert and extract data from a table, and calls a simple OData service, too:

Procedure

1. Clone the XSA GIT repository containing the Node.js/JavaScript code examples used in this tutorial.

   The GIT repository contains code samples and artifacts for the following simple JavaScript applications, which you can use to learn how to set up the module structure of the XS advanced application:

   - "myxsaapp1"
     Simple JavaScript application
   - "myxsaapp2"

JavaScript application with database artifacts and an XS OData service

Output Code

```
$ git clone https://github.com/saphanaacademy/XSA.git
cloning into 'XSA' ...
    remote: Counting objects: 33, done.
    remote: Compressing objects: 100% (27/27), done.
    remote: Total 33 (delta 2), reused 33 (delta 2), pack-reused 0
    unpacking objects: 100% (33/33), done
    Checking connectivity... done.
```

In the cloned repository on your local machine, you should see the following high-level structure; in this tutorial, we focus on the cloned application "`myxsaapp2`":

i Note

The contents of the XSA Git repository might include additional objects that are not required to complete this tutorial.

Output Code

```
<gitrepo>
|- myxsaapp1/
|  |- web/
|  |- xsjs/
|  \- manifest.yml
|- myxsaapp2/
|  |- db/
|  |- web/
|  |- xsjs/
|  \- manifest.yml
.gitattributes
README.md
```

2. Check the contents of the cloned `myxsaapp2` folder.

   The cloned `myxsaapp2` folder should look like the following example:

Output Code

```
|- myxsaapp2/
   |- db/                      # Database artifact
   |  |- .npmrc                # NPM Registry configuration
   |  |- package.json          # Node.js package dependencies
   |  \- src/
   |     |- .hdiconfig
   |     |- .hdinamespace
   |     \- MyContext.hdbcds
   |- web/                     # Static Web resources
   |  |- .npmrc                # NPM Registry configuration
   |  |- package.json          # Node.js package dependencies
   |  |- xs-app.json           # Route configuration
   |  \- resources/
   |     \- index.html
   |- xsjs/                    # Application code
   |  |- .npmrc                # NPM Registry configuration
   |  |- package.json          # Node.js package dependencies
   |  |- server.js             # Application entry point
   |  \- lib/
   |     |- insert.xsjs
```

```
|     |- index.xsjs
|     \- service.xsodata
|
\- manifest.yml            # Application overview
```

3. Adapt the application's deployment dependencies defined in the `manifest.yml` file to suit your SAP HANA XS advanced environment.
   a. Check the module names, for example: "`myxsaapp2-web`", "`myxsaapp2-hdb-db`", and "`myxsaapp2-xsjs`".
   b. Choose a port for the `web/` and `xsjs/` modules.

      You can allow SAP HANA XS advanced to allocate ports automatically, or choose a free port manually, for example, "64021" and "64022", for `web/` and `xsjs/` respectively.

      > → Tip
      >
      > If no port is specified manually, a port is assigned automatically during application deployment. A port allocation is not required for the database module `db`.

   c. Check the module paths, for example: "`db`", "`web`" and "`xsjs`". The path must point to the location where module-specific files are stored.
   d. Check that the database module `myxsaapp2-db` includes the entry "`no-route: true`". This entry is mandatory and signifies that no direct access to the HDI container is necessary from the UI; access is ensured by other means, for example, JavaScript code or an OData service.
   e. Check that the service names match the service instances you created, for example, "`my-hdi-container`" for the HDI container for the deployed database artifacts" and `my-uaa`" for user authentication and authorization in XS advanced.
   f. Adapt the application's destination in the `web/` module to point to the application's entry point `server.js` with the appropriate host and port.

      > Sample Code
      >
      ```
      [
        {"name":"xsjs-backend", "url":"https://acme.com:64022",
      "forwardAuthToken": true}
      ]
      ```

   When complete, the `manifest.yml` file for the application "`myxsaapp2`" should look like the following example:

> Sample Code
>
```
ID: myxsaapp2
_schema-version: '2.0'
version: 0.0.1
applications:
 - name: hdi-db
   path: db
   memory: 256M
   no-route: true
   services:
    - my-hdi-container
 - name: xsjs
   type: nodejs
   path: xsjs
   memory: 128M
   port: 51510
```

```
    services:
     - my-uaa
     - my-hdi-container
  - name: web
    type: html5
    path: web
    memory: 128M
    services:
     - my-uaa
    env:
     destinations: >
      [
       {"name":"xsjs-backend",
        "url":"https://acme.com:64021",
        "forwardAuthToken":true}
      ]
```

4. Configure the use of the NPM Registry to resolve package dependencies.

   The applications `myxsaapp1` and `myxsaapp2` have dependencies to the following packages included in the NPM Registry.

   - `@sap/hdi-deploy`
   - `@sap/approuter`
   - `@sap/xsenv`
   - `@sap/xsjs`

   If you are using command-line tools to build your Node.js applications for XS advanced, you can configure the tools to use a particular NPM registry to resolve the dependencies between resources specified in the deployment descriptors. To specify the NPM Registry, add the `"@sap:registry="` property to the `.npmrc` resource-configuration file, as illustrated in the following example:

   ### Sample Code

   ```
   @sap:registry=https://registry.npmjs.org
   strict-ssl=false
   ```

   ### Note

   The `strict-ssl=false` option is only required if XS advanced is behind a firewall.

5. Check the package dependencies for the new database module “`myxsaapp2-db`”; the dependencies are defined in the file `myxsaapp2/db/package.json`.

   ### Sample Code

   Database Container Dependencies (`myxsaapp2/db/package.json`)

   ```
   {
     "name": "deploy",
     "dependencies": {
       "@sap/hdi-deploy": "latest"
     },
     "scripts": {
       "start": "node node_modules/@sap/hdi-deploy/deploy.js"
     }
   }
   ```

6. Check the contents of the database module's `db/src` folder; it contains some configuration files and any database artifacts.

Output Code

```
|- myxsaapp2/
   |- db/
      |- .npmrc
      |- package.json
      \- src/
         |- .hdiconfig
         |- .hdinamespace
         \- MyContext.hdbcds
```

The database artifact `.hdiconfig` specifies the plug-ins (and the version) to use when deploying a particular **type** of database artifact to an HDI container; the artifact type is defined in the file suffix, for example, `.hdbcds` or `.hdbsynonym`. The plug-in definition ensures that the appropriate run-time object is created from the specified design-time artifact, as specified in the following example:

Sample Code

HDI Configuration (`.hdiconfig`)

```
{
  "file_suffixes": {
    "hdbcds": {
      "plugin_name": "com.sap.hana.di.cds",
      "plugin_version": "2.0.0.0"
    },
  "file_suffixes": {
    "hdbsynonym": {
      "plugin_name": "com.sap.hana.di.synonym",
      "plugin_version": "2.0.0.0"
    },
  "file_suffixes": {
    "hdbsynonymconfig": {
      "plugin_name": "com.sap.hana.di.synonym.config",
      "plugin_version": "2.0.0.0"
    },
}
```

The database artifact `.hdinamespace` specifies a design-time namespace to use in run-time object name, for example, `"<name>.<space>::RunTimeObject"`.

Sample Code

HDI Namespace Configuration (`.hdinamespace`)

```
{
  "name"     : "myxsaapp2"
  "subfolder": "append"
}
```

Note

`"append"` signifies that design-time folder paths (for example under “`db/`”) should be added as a prefix to the run-time object names, for example, `"<name>.<space>.<subfolder(s)>::RunTimeObject"`.

The database artifact `MyContext.hdbcds` is a simple document that defines database objects (in this example, a table specifying an ID and a corresponding value) using the Core Data Services (CDS) syntax.

Sample Code

Database Artifact (`MyContext.hdbcds`)

```
namespace myxsaapp2;
context MyContext {
  Entity MyTable {
    key id      : Integer generated always as identity;
        created : UTCTimestamp;
  };
};
```

7. Check the contents of the `myxsaapp2` application's JavaScript `xsjs` module; the `lib/` subfolder contains the code for the business logic used by the application.

Sample Code

```
|- myxsaapp2/
   |- xsjs/
   |  |- .npmrc
   |  |- package.json
   |  |- server.js
   |  \- lib/
   |     |- insert.xsjs       # Insert data into a table
   |     |- index.xsjs        # Query data in a table
   |     \- service.xsodata   # XS OData service to extract data
   \- manifest.yml
```

The `service.xsodata` defines an OData service used by the application `myxsaapp2` to extract data from the table "`MyTable`" (in the context "`MyContext`" and the name space "`myxsaapp2`):"

Sample Code

```
service: {
   "myxsaapp2"::MyContext.MyTable" as "MyEntity";
       create forbidden
       update forbidden
       delete forbidden
}
```

8. Check the package dependencies for the application's `web` module "`myxsaapp2-web`"; the dependencies are defined in the file `myxsaapp2/web/package.json`.

Sample Code

Application Router's Package Dependencies (`myxsaapp2/web/package.json`)

```
{
  "name": "web-approuter",
  "engines": {
     "node": ">=4.0.0 <7.0.0"
  },
  "dependencies": {
     "@sap/approuter": "latest"
  },
  "scripts": {
     "start": "node node_modules/@sap/approuter/approuter.js"
```

```
  }
}
```

9. Check the specifications of the application's router configuration; the details are specified in the file `myxsaapp2/web/xs-app.json`.

   Application routes are defined in one or more “destinations”, which specify where files are expected to be found for the relevant application modules, for example, `xsjs`.

   Sample Code

   Application Router Configuration (`myxsaapp2/web/xs-app.json`)

```
{
  "welcomeFile": "index.html",
  "authenticationMethod": "route",
  "routes": [{
      "source": "^(.*)(.xs(js|odata))(.*)$",
      "destination": "xsjs-backend"
  }]
}
```

10. Check the routing configuration for the application's Web module.
11. Deploy the JavaScript application “myxsaapp2” to the XS advanced run time.
    a. Change directory to the folder that contains the `myxsaapp2` application's deployment manifest file “`manifest.yml`”.
    b. Deploy the application

    Output Code

```
$ xs push
Using manifest file “manifest.yml”
Creating app “myxsaapp2-web” in org “devteam1” / space “DEV” as DEVUSER
Creating route “myxsaapp2-web.acme.com:64021” in org “devteam1” / space
“DEV” as DEVUSER
Binding route “myxsaapp2-web.acme.com:64021” to app “myxsaapp1-web”
Uploading “myxsaapp2-web""...
...
  staging app "myxsaapp2-xsjs" ...
  starting app "myxsaapp2-xsjs" ...
Showing status and information about app "myxsaapp2-xsjs"
  name:              myxsaapp2-xsjs
  requested state:   STARTED
  instances:         1
  memory:            128 MB
  disk:              <unlimited>
  buildpack:         <default>
  urls:              https://acme.com:64022

Instances:
index    created                    state
-----------------------------------------
0        Dec 25, 2015 10:22:37 AM    RUNNING
```

12. Check which applications are currently running in the XS advanced run time.

    Output Code

```
$ xs apps
Getting apps in org "devteam1" / space "DEV" as DEVUSER...
```

```
  Found apps:

  name          requested state   instances   memory   urls

------------------------------------------------------------------------
-----
  myxsaapp2-web    STARTED           1/1         100 MB   https://
acme.com:64021
  myxsaapp2-db     STARTED           1/1         100 MB   <none>
  myxsaapp2-xsjs   STARTED           1/1         128 MB   https://
acme.com:64022
```

> **i** Note
>
> The database application has no URL assigned because it not accessed directly from the UI (`no-route:true`); it is accessed from the HDI container by means of the JavaScript and OData business logic.

13. Check the running services.

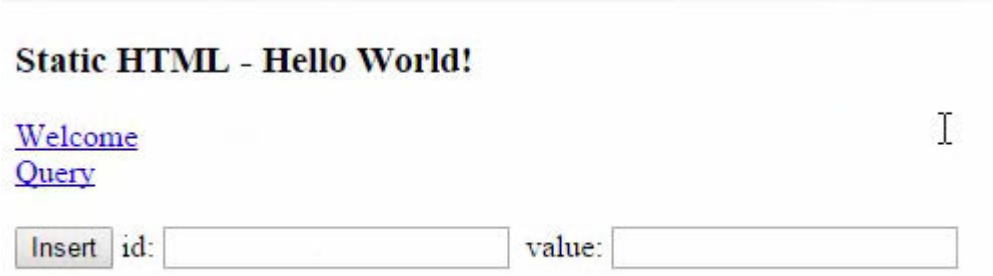
    The following information should be displayed:

> Output Code
>
> ```
> $ xs services
> Getting services in org "devteam1" / space  "DEV" as DEVUSER
> Found services:
> name              service  plan        Bound apps
> ------------------------------------------------------------------
> my-uaa            xsuaa    default     myxsaapp2-web, myxsaapp2-xsjs
> my-hdi-container  hana     hdi-shared  myxsaapp2-db
> ```

14. Check the application `myxsaapp2` in a Web browser.

    Specify the entry point for the application router: **`https://acme.com:64021`**

    The following Web page is displayed



    1. Test the “Welcome” link.
       Click *Welcome*; a Web page is displayed with the message: `XSJS - Hello DEVUSER`
    2. Insert some data into your table by using the “Insert” button.
       Enter an *id* (for example, “1”) and a *value* (for example, “Kofi”) and choose *Insert*. If the operation is successful, a Web page is displayed with the message `XSJS Insert`.
    3. Run a query on the table data by using the “Query” link.
       Choose *Query*. If the operation is successful, a Web page is displayed with the content: `[{"id":1,"value":"Kofi"}]`. At this point, this is in the only data in the table.

4. Test the OData service that you defined.
   In the Web browser, enter the following URL: **`https://acme.com:64021/service.xsodata`**

   Output Code

```
<service xmlns:atom="http:/www.w3.org/2005/Atom"
         xmlns:app="http://www.w3.org/2007/app"
         xmlns:app="http://www.w3.org/2007/app"
         xml:base="https://host:64021/service.xsodata/">
  <workspace>
    <atom:title>Default</atom:title>
    <collection href="MyEntity">
      <atom:title>MyEntity</atom:title>
    </collection>
  </workspace>
</service>
```

Related Information

Getting Started with Application Development in XS Advanced [page 27]
Set up an XS Advanced Application Project [page 101]
Get Started with the XS CLI Client [page 76]
Standard Node.js Packages for XS Advanced [page 635]
The NPM Registry [page 687]

3.1.3 Working with Gerrit Version Control in XS Advanced

An overview of the version-control tools available for the management of design-time artifacts in SAP HANA XS advanced model.

Gerrit is an open-source Git server that provides access control for the hosted Git repositories as well as a Web-based client for doing code review. Code review is a core (but optional) of Gerrit; individual development teams can decide whether to work with or without code review.

Gerrit is an optional component of the XS advanced platform which can be used to store and manage versions of the source code for XS advanced applications, for example, JavaScript or Java applications and SAPUI5 in Git repositories.

> **i Note**
>
> SAP Gerrit for XS advanced is deprecated as indicated in SAP Note 1589311. However, SAP Note 2889508 describes how you can migrate an existing installation of Gerrit for XS advanced to the open-source version of Gerrit while retaining all repositories and review data, and continuing to use XS advanced user-account-and-authentication service (XSUAA) to manage SAP HANA XS advanced user accounts. For information about how to perform a fresh installation of the open-source version 3.1 Gerrit server for use with XS advanced, see SAP Note 2890964.

Git is a widely used open-source system for revision management of source code that facilitates distributed and concurrent large-scale development work flows. It supports a large variety of commands for working with repositories.

> **i Note**
>
> To work with Gerrit you must understand the basic concepts of Git and be familiar with the basic Git commands and work flows used when developing an application. In addition, to perform Git operations, you must also have access to a suitable Git client, for example, the SAP Web IDE for SAP HANA. You can find the documentation for Gerrit for XS advanced at the following URL on the system where you installed Gerrit for XS advanced: `https://<GerritServer>.acme.com:8080/Documentation/index.html`

Related Information

Set up Gerrit for XS Advanced Application Development [page 96]
Using Source Control (Git) [page 1254]

3.1.3.1 Set up Gerrit for XS Advanced Application Development

A description of the basic steps developers need to perform to get started with Gerrit and Git version-management tools.

Prerequisites

To start working with Gerrit version control, the following prerequisites apply:

- A Gerrit server is available in your development environment
- The Gerrit server is configured to work with XS advanced

  > **i Note**
  >
  > SAP Gerrit for XS advanced is deprecated as indicated in SAP Note 1589311. However, SAP Note 2889508 describes how you can migrate an existing installation of Gerrit for XS advanced to the open-source version of Gerrit while retaining all repositories and review data, and continuing to use XS advanced user-account-and-authentication service (XSUAA) to manage SAP HANA XS advanced user accounts. For information about how to perform a fresh installation of the open-source version 3.1 Gerrit server for use with XS advanced, see SAP Note 2890964.

- You have access to a Git client, for example, the client included with SAP Web IDE for SAP HANA

Context

To get started with Gerrit in XS advanced, you need to perform the following steps:

> → Tip
>
> The documentation for Gerrit for XS advanced is available on the system where the Gerrit for XS advanced server is running at the following URL:
>
> `https://<GerritServer>.acme.com:8080/Documentation/index.html`

Procedure

1. Sign in to Gerrit.

   Before you can start uploading files and changes to Gerrit, you must create and activate your Gerrit account.

   a. Open the Gerrit Web UI's start page in a Web browser.

   > i Note
   >
   > The URL of the Gerrit server is defined during the installation process.

   Check the information displayed for `git-service` in the system output under the section `Registered Service ULRs`, as illustrated in the following example:

   > i Note
   >
   > You will need to change the port number to the one published on your system, for example:
   >
   > `https://host.acme.com:29<HANA_Instance_Nr>9`

   > ‹≡› Output Code

```
xs version
Client version: xs v0.1614-cons
Server version information:
   name                    = XS Controller
   support                 = http://service.acme.com/message
   build                   = v0.1614-cons
   version                 = 1
   user                    = <not set>
   description             = SAP XS Runtime on premise
   controllerEndpoint      = https://host.acme.com:30030
   authorizationEndpoint   = https://host.acme.com:30032/uaa-security
   loggingEndpoint         = <not set>
   allowDebug              = true
   ...
Registered service URLs:
   deploy-service       = https://host.acme.com:51002
   component-registry   = https://host.acme.com:51003
   product-installer    = https://host.acme.com:51005
   git-service          = https://host.acme.com:29419
```

b. Choose *Sign In* in the top right-hand corner of the screen.

   Gerrit is attached to the user management system (XSUAA) of the XS Advanced platform. As a result, the sign-in process redirects you to the standard log-in screen of the SAP HANA database.

   > **i Note**
   >
   > If you do not have an SAP HANA identity in the XS UAA server, contact your SAP HANA system administration team and ask them to create a user account for you.

c. Enter your SAP HANA user credentials to sign in to Gerrit.

   Gerrit displays your personal *My Reviews* dashboard.

2. Have a look at your *Reviews Dashboard*.

   You can manage the projects you are assigned to in *Watched Projects*, which you can find in *Settings* > *Watched Projects*.

   > **i Note**
   >
   > When you sign in to Gerrit for the first time, your personal dashboard will typically be empty, because you have not yet been assigned to any Gerrit project.

3. Select a Gerrit development project to join.

   To display a list of all available projects, choose *Projects* > *List*.

   This list shows all projects to which you have at least "read" access rights together with a short description and a link to the repository browser. The repository browser allows you to examine the content and history of the Git repository associated with the Gerrit project.

4. Display details of a Gerrit development project.

   To display details of a Gerrit project, choose the name of the project in the project list. The Gerrit project contains the reference to the Git repository used to manage the project's application code.

   > **→ Tip**
   >
   > To clone the project to your workplace, you need the project's URL, which is displayed immediately below the project title. To clone the Gerrit project, you can use any standard Git client, for example, a Git command-line client or the Git tools provided with SAP Web IDE for SAP HANA.

5. Create a new project.

   If you are not assigned to a Gerrit project, and cannot find a project in the list of existing projects, you might have to create a new project. Gerrit provides the following methods for creating a new project:

   > **i Note**
   >
   > In a production environment, the permissions required to create a new Gerrit project are granted to a selected group of "admin" users. As a new Gerrit user, you will need to ask the Gerrit administration group to create the new project for you.

   - In the Gerrit Web UI under *Projects* > *Create Project*

> → Tip
>
> If you intend to bind the new repository to an XS advanced application, you must create it with the XS advanced service broker. Git repositories not created with the XS advanced service broker are not visible for the XS Advanced run time.

6. Access the Git repository.

   The Git repository stores the authoritative copy of the application source code being developed in the project. Project team members fetch code from (and push code to) this repository. Build servers and other packaging tools use the Git repository to obtain the source-code, too.

   By default, Gerrit requires authentication and authorization for all Git operations. For example, to upload changes to a Git repository, you must either be a project owner or have been granted the appropriate access rights for the target repository. If you need write access to a Gerrit project, contact the project owner or the Gerrit administrator.

   > i Note
   >
   > Default access rules allow all authenticated users to `push` changes for review to any Gerrit project. However, only project owners are allowed to submit changes to the underlying repository. This facilitates the implementation of the simple contributor-committer work flow followed by many open-source projects.

   It is possible to enable anonymous read access to a repository by adding a "Read" permission for the reference refs/* and assigning that permission to the predefined group "Anonymous Users". To grant anonymous read access for **all** repositories, you must adapt the configuration of the project *All-Projects*, from which all other projects inherit their permissions.

7. Set up your Gerrit profile.

   Before you start working with Gerrit you should check and, if necessary, amend your contact information, and customize Gerrit to your personal needs and preferences. You might also want to enable SSH communication for your account.

   - Contact information
     Git identifies the author and committer of a change by their e-mail addresses, and you cannot upload any changes to Gerrit if the configured e-mail address is wrong or invalid. Gerrit stores these attributes in your contact information, which you can access under *Settings* > *Contact Information*.

     > i Note
     >
     > When you sign in to Gerrit for the first time, an account is created using the full name and e-mail address received from the XS advanced user-management system (XSUAA). In some cases, this information might not be accurate, especially if you sign in using the credentials of an SAP HANA database user. Furthermore, in a productive environment it might be necessary to use your corporate e-mail address and real name, regardless of your identity in the SAP HANA system.

   - Preferences
     You can customize Gerrit by configuring your preferences in your profile, for example, under *Settings* > *Preferences*.
   - Secure Shell (SSH) settings

Gerrit supports the SSH protocol for Git operations. If you want to use SSH for communication with Gerrit, you first must generate a suitable SSH key pair and upload the public key to your profile, for example, under *Settings* > *SSH Public Key*.

8. Clone an existing repository. (optional)

   If you already have a Git repository that you want to clone, you can use the tools provided by SAP Web IDE for SAP HANA to perform the clone operation.

   a. Obtain the URL of the Gerrit server where the Git repository that you want to clone is located.

   > **Note**
   >
   > The URL of the Gerrit server is defined during the installation process.

   Check the information displayed for `git-service` in the system output under the section `Registered Service ULRs`, as illustrated in the following example:

   > **Output Code**

```
xs version
Client version: xs v0.1614-cons
Server version information:
   name                    = XS Controller
   support                 = http://service.acme.com/message
   build                   = v0.1614-cons
   version                 = 1
   user                    = <not set>
   description             = SAP XS Runtime on premise
   controllerEndpoint      = https://host.acme.com:30030
   authorizationEndpoint   = https://host.acme.com:30032/uaa-security
   ...
Registered service URLs:
   deploy-service       = https://host.acme.com:51002
   component-registry   = https://host.acme.com:51003
   product-installer    = https://host.acme.com:51005
   git-service          = https://host.acme.com:29419
```

   b. Start SAP Web IDE for SAP HANA.

   If SAP Web IDE for SAP HANA is not available at the default URL `https://host.acme.com:53075`, you can use the `xs` CLI to display the URL allocated to the application. In a command shell, log in to the XS advanced run time and use the command `xs app webide --urls` to display the URL required to access SAP Web IDE for SAP HANA in a Web browser.

   c. Clone the Git repository.

   In SAP Web IDE for SAP HANA, choose *File* > *Git* > *Clone Repository*.

   You will need to provide the following details:

   - *URL*
     The URL to the Gerrit server where the project's Git repository is located, for example, **`https://host.acme.com:29419`**
   - *Repository Path*
     The path to (and name of) the Git repository you want to clone, for example, **`/<RepositoryName>.git`**
   - *Connection Protocol*
     The protocol to use for the connection to the Gerrit server, for example, **`HTTPS`**.

- *Authentication*
  The log-on credentials for a SAP HANA user with access permissions for the cloned Git repository.

Related Information

Working with Gerrit Version Control in XS Advanced [page 95]
Using Source Control (Git) [page 1254]

3.1.4 Set up an XS Advanced Application Project

An overview of the steps required to set up and deploy an application project for SAP HANA XS advanced.

Prerequisites

- Development tools
  You can choose between graphical and command-line tools:
  - SAP Web IDE for SAP HANA
    If the SAP Web IDE for SAP HANA is not available at the default URL `https://host.acme.com:53075`, log in to the XS advanced run time with the XS command-line interface (CLI) and use the command `xs app webide --urls` to display the URL required to access the tool.
  - You have installed the XS command-line client on your development machine.
    The XS CLI client tools are installed by default on the SAP HANA server. However, you must download and install the client tools locally, if you want to connect to SAP HANA from your local machine.
    The XS CLI client tools (`xs.onpremise.runtime.client_<platform>-<version>.zip`) can be downloaded from the SAP HANA server, from the installation DVD, or from the SAP support portal. For more information about the SAP support portal, see the link to SAP note 2242468 in the related links below.
  - SAP Node.js packages
    The archive (`xs_javascript-<version>-bundle.tar.gz`) is available on the SAP HANA media and the SAP Marketplace (SMP); it includes a selection of Node.js packages developed by SAP for use with the Node.js applications running in the XS advanced run time.

    > → Tip
    >
    > Alternatively, you can configure NPM to resolve SAP-related NPM dependencies using the public NPM Registry (at `https://registry.npmjs.org`).
- Version control, for example, GIT/Gerrit
  Gerrit for SAP HANA XS advanced is an optional component of the XS advanced platform which can be used to store and manage versions of the source code for XS advanced applications (for example, JavaScript or Java applications and SAPUI5) in Git repositories.

> → Tip
>
> The SAP Web IDE for SAP HANA includes an integrated Git client.

- Access to a running SAP HANA system where the XS advanced run time is installed and configured

> ⚠ Caution
>
> The code examples included in this document for XS advanced are often syntactically incomplete; they are intended for illustration purposes only.

Context

The end-to-end process of developing an application that you want to deploy on SAP HANA is straight forward, as illustrated in the steps described in this high-level overview.

XS Advanced Application Development Overview

Step	Description
1	Create the design-time folder infrastructure for your application source files
2	Create the deployment descriptor files for your application
3	Add database artifacts and content
4	Add application business logic to work with the database artifacts
5	Create any required OData service definitions
6	Create the user interface in which to display the database content
7	Add security to protect what you have built
8	Define application routes (and destinations)
9	Create any service instances needed by your application
10	Add scheduled jobs if required (optional)
11	Deploy your project
12	Test your project in the user interface

> ℹ Note
>
> Some configuration artifacts are subject to restrictions regarding their location; SAP HANA XS advanced deployment tools **expect** to find particular artifacts in a particular location.

Procedure

1. Create the design-time folder infrastructure for your application source files.

   To make the maintenance of source files easier and facilitate the task of deploying the application, it is also essential to create a clear and consistent structure for the design-time artifacts that you develop for your application. The folders typically include modules for the following areas: database, Web, application code, OData (optional). The security description (OAuth2 client configuration) can be placed either in its own root-level module or in the application code structure, for example, `js/security/` or `java/security/`.

   → Tip

   SAP Web IDE for SAP HANA provides Wizards and templates to assist with the creation of application modules and the related resources for your MTA. If you are using the XS command-line tools to develop an MTA for SAP HANA XS advanced, you must create all modules and resources yourself.

   Sample Code

```
<myAppName>
|- db/                              # Database module
|  |- package.json                  # Database details/dependencies
|  \- src/                          # Database artifacts: tables, views,
|     |- .hdiconfig                 # HDI plug-in configuration
|     |- .hdinamespace              # HDI plug-in namespace configuration
(optional)
|     \- myModel.hdbcds             # Database design-time definition
|- js/                              # JavaScript app module
|  |- start.js                      # JavaScript app entry point
|  |- package.json                  # JavaScript app details/dependencies
|  \- src/                          # JavaScript source code
|     \- odata/                     # OData module
|        |- resources/              # OData service resources
|        \- services/               # OData service definitions
|           | - srv1.xsodata       # OData service definition
|           \ - srvN.xsodata       # OData service definition
|- ui/                              # Application UI module
|  |- xs-app.json                   # Application routes configuration
|  |- package.json                  # Application details/dependencies
|  \- resources/                    # Static UI resources
|     \- index.html                 # Client UI start page
|- xs-security.json                 # Application security description
|- manifest.yml                     # Manifest description incl.
destinations
|- mta.yaml                         # Central development manifest (SAP Web
IDE only)
\- mtad.yaml                        # Central deployment manifest
```

2. Create the development and deployment descriptor files for your application.

   The development and deployment descriptors specify what application components to build (and how), the dependencies between the application components, and where (and how) to deploy the components. The development and deployment manifest files for the multi-target application must be located in the application's root folder.

   - `mta.yaml`
     Development manifest for a multi-target application (MTA)

> → Tip
>
> The `mta.yaml` is only required if you are working with SAP Web IDE for SAP HANA, which creates (and maintains) the file automatically when you create a new application project and maintain the project's components.

- `mtad.yaml`
  Deployment manifest for a multi-target application (MTA)

  > → Tip
  >
  > If you are working with the XS advanced command-line client, you must create (and maintain) the `mtad.yaml` file manually. If you are using SAP Web IDE for SAP HANA, no action is required; SAP Web IDE for SAP HANA creates the deployment manifest `mtad.yaml` transparently at deployment time using the information specified in the corresponding development descriptor (`mta.yaml`).

3. Add database artifacts and content.

   Create database artifacts such as tables and views, for example using Core Data Services (CDS). You can also create procedures and sequences, for example, using SQLScript.

   > i Note
   >
   > The database content must include a `package.json` file containing details of dependencies and, in addition, the container configuration (`.hdiconfig`) and run-time name space configuration (`.hdinamespace`) files.

4. Add application business logic to work with the database artifacts.

   The business logic is used to help retrieve data from the database, for example, using OData services. The data is requested from and displayed in the client user interface.

   For applications written in JavaScript (Node.js or XS JavaScript), you need to perform the following high-level steps:

   a. Add a package description file (`package.json`) to your JavaScript resource-files folder with details of your application's contents and dependencies.
   b. Add a startup file for your JavaScript application, for example, `main.js`.
   c. Create any additional JavaScript (`.js` or `.xsjs`) files and libraries.

   For applications written in Java, you need to perform the following high-level steps:

   a. Write Java application's code.
   b. Look up the required data source.
   c. Build the WAR file based on the project object model file (`pom.xml`), for example, with Maven.
   d. Use the Java Persistence API (JPA) to work with CDS entities (if required).

5. Create any required OData service definitions.

   An OData service definition is described in an `xsodata` artifact, for example, `myOdataService.xsodata`; the service definition should be placed in the `resources/` folder of your OData project structure.

6. Create the user interface (GUI) to display the database content (HTML pages).
7. Add security to protect what you have built (`xs-security.json`).

   The tasks required to set up authorization artifacts in SAP HANA XS advanced are performed by two distinct user roles:

- Application developer
  Creates the security descriptor (`xs-security.json`) for the XS advanced application; maintains role templates and authorization scopes (in the `xs-security.json`) to define who has access to the application; binds the application to an XS advanced service instance; deploys the application to the XS advanced run-time environment.
- SAP HANA XS administrator.
  After deployment of the authorization artifacts as role templates, the XS advanced run-time administrator uses the role templates provided by the developers to build roles, group the roles into role collections, and assign the role collections to business users, for example, using the *SAP HANA XS Administration Tools*.

8. Define application routes (and destinations).

   The application router is used to serve static content, propagate user information, and act as a proxy to forward requests to other micro services. Application routes are defined in the application-router descriptor `xs-app.json`, a JSON-formatted file that should be located in the application's Web resources module.

   Destinations are defined in an environment variable for the `@sap/approuter` application. The destinations are typically specified in the application manifest (`manifest.yml`) file. The router information is added to the `env: destinations` section of the `manifest.yml`, as illustrated in the following example.

9. Create any service instances needed by your application.

   > **i Note**
   >
   > SAP Web IDE for SAP HANA creates any required services based on the information in the "`resources`" section of the application's development/deployment descriptor (`mta.yaml`) and binds the application to the service instance created.
   >
   > If you are using the XS advanced CLI, you must create any services and manually bind the application to the service instance, for example, using the `xs create-service` and `xs bind-service` commands.

   ```
   xs create-service <SERVICE> <PLAN> <SERVICE_INSTANCE>
   ```

   - HDI container service instance
     ```
     xs create-service hana hdi-shared my-hdi-container
     ```
   - User Account and Authentication (UAA) service
     Using information specified in the application's security descriptor (`xs-security.json`)
     ```
     xs create-service xsuaa default -c </path/to/>xs-security.json
     ```

10. Add scheduled jobs if required (optional).

    Jobs can be scheduled using an instance of the Job Scheduler service.

    > **i Note**
    >
    > For more information, see *Scheduling Jobs in XS advanced* in the *SAP HANA Administration Guide* (*Related Links* below).

11. Deploy your application project.
    a. Deploy an application to the XS advanced run-time environment.
       - XS CLI client

- Push an individual application component/module:
  `xs push <myAppName>`
- Deploy a complete MTA archive:
  `xs deploy <myMTAName>.[mtar | zip]`
- Install a Software Component Version (SCV):
  `xs install <myAppArchive>.zip`

12. Test your application project in the user interface.

    On build or deployment, an XS advanced application is assigned a URL with a specific port. You can paste the assigned URL into a Web Browser to test it. This information is displayed differently depending on the tools you are using:

    - SAP Web IDE for SAP HANA

      In the UI, select the application's UI module and, in the context menu, choose *Run* > *Run as* > *Web Application*. When the application is up and running, SAP Web IDE displays the URL required to access the application as well as the application's current status at the bottom of the run console in the details pane:

      → Tip

      If the application is already running, you do not need to restart it; selecting an application module in the SAP Web IDE project pane displays the URL and status of the application in the run console:

      Output Code

      ```
      Run tinyworld/tinyui
      Updating application
      Application is starting
      Application is running

      Application: https://acme.com:51031          Status: Running
      ```

    - XS advanced CLI

      Displays the ports allocated to applications running in a given run-time space, for example, using the `xs apps` command, as illustrated in the following example:

      Output Code

      ```
      $>xs apps
      Getting apps in org "myorg" / space "PROD" as XSA_ADMIN...
      Found apps:
      name                    state     instances  urls
      ------------------------------------------------------------------------
      ---------
      hrtt-service            STARTED   1/1        https://acme.com:51021
      hrtt-core               STARTED   1/1        https://acme.com:51020
      hrtt-db                 STOPPED   0/1        <none>
      di-core-db              STOPPED   0/1        <none>
      di-local-npm-registry   STARTED   1/1        https://acme.com:51025
      di-core                 STARTED   1/1        https://acme.com:53030
      di-runner               STARTED   1/1        https://acme.com:51027
      webide                  STARTED   1/1        https://acme.com:53075
      di-builder              STARTED   1/1        https://acme.com:51029
      [...]-tinyjs            STARTED   1/1        https://acme.com:51030
      [...]-tinyui            STARTED   1/1        https://acme.com:51031
      ```

Related Information

XS Advanced Application Resource Files [page 107]
Maintaining Application Services in XS Advanced [page 826]
Managing SAP HDI Containers and Artifacts [page 234]
Maintaining Application Security in XS Advanced [page 957]
The XS Command-Line Interface Reference
SAP Web IDE for SAP HANA Reference [page 1060]

3.1.4.1 XS Advanced Application Resource Files

SAP HANA XS advanced applications require some mandatory configuration artifacts.

In SAP HANA XS advanced, applications require a number of mandatory configuration and deployment files. Which files are required depends on the language you are using to develop the application, for example, JavaScript (including node.js) or Java, and the deployment options you need. The following table provides an overview of the artifacts required by each application. There are rules and requirements concerning where to store these artifacts in the application folder hierarchy.

XS Advanced Application Artifacts Overview

File Name	Description	Database	App Router	Node/JS	Java	Python
`mta.yaml`	MTA development descriptor	Yes	Yes	Yes	Yes	Yes
`mtad.yaml`	MTA deployment descriptor	Yes	Yes	Yes	Yes	Yes
`package.json`	Description and dependencies of node modules	Yes	Yes	Yes		
`xs-app.json`	Application router description		Yes			
`*.hdi*`	Configuration artifacts for the SAP HANA deployment infrastructure (HDI), for example, `.hdiconfig` and `.hdinamespace`	Yes				
`*.java`	Java application source files				Yes	
`*.xsjs`	SAP HANA XS JavaScript source files			Yes		
`*.js`	JavaScript (node.js) source files			Yes		
`*.py`	Python source files			Yes		Yes

File Name	Description	Database	App Router	Node/JS	Java	Python
`xs-security.json`	Shared description of OAuth2 client					

> **i** Note
>
> If you use SAP Web IDE for SAP HANA to develop an MTA, both the `mta.yaml` (**development** descriptor) and the `mtad.yaml` (**deployment** descriptor) are created automatically. The `mta.yaml` is generated when you create the application project in SAP Web IDE for SAP HANA, and the `mtad.yaml` file is created when you build the project.
>
> The `mta.yaml` is not required if you are using command-line tools to build an MTA for XS advanced; you must specify all build and deployment dependencies in the deployment descriptor (`mtad.yaml`).

Related Information

Maintaining the XS Advanced Application Development & Deployment Descriptors [page 136]
The XS Advanced Application Descriptor [page 917]
The XS Advanced Application Package Descriptor [page 819]
The SAP HDI Container Configuration File [page 241]
The HDI Name-Space Configuration File [page 250]
The Application Security Descriptor [page 996]

3.2 Working with the SAP Cloud Application Programming Model

An overview of the steps required to create a simple Node.js application using SAP's Cloud application programming model.

This tutorial aims to show you how to use the Node.js and the Odata V4 framework as well as some exit handlers to construct an application using the SAP Cloud Application Programming Model (CAP) with the tools provided in the SAP Web IDE for SAP HANA. The use of so-called "exit handlers" enables real-time interaction with data in the database, for example, by means of a Fiori or SAPUI5 Web User Interface (UI). The tutorial comprises the following high-level steps:

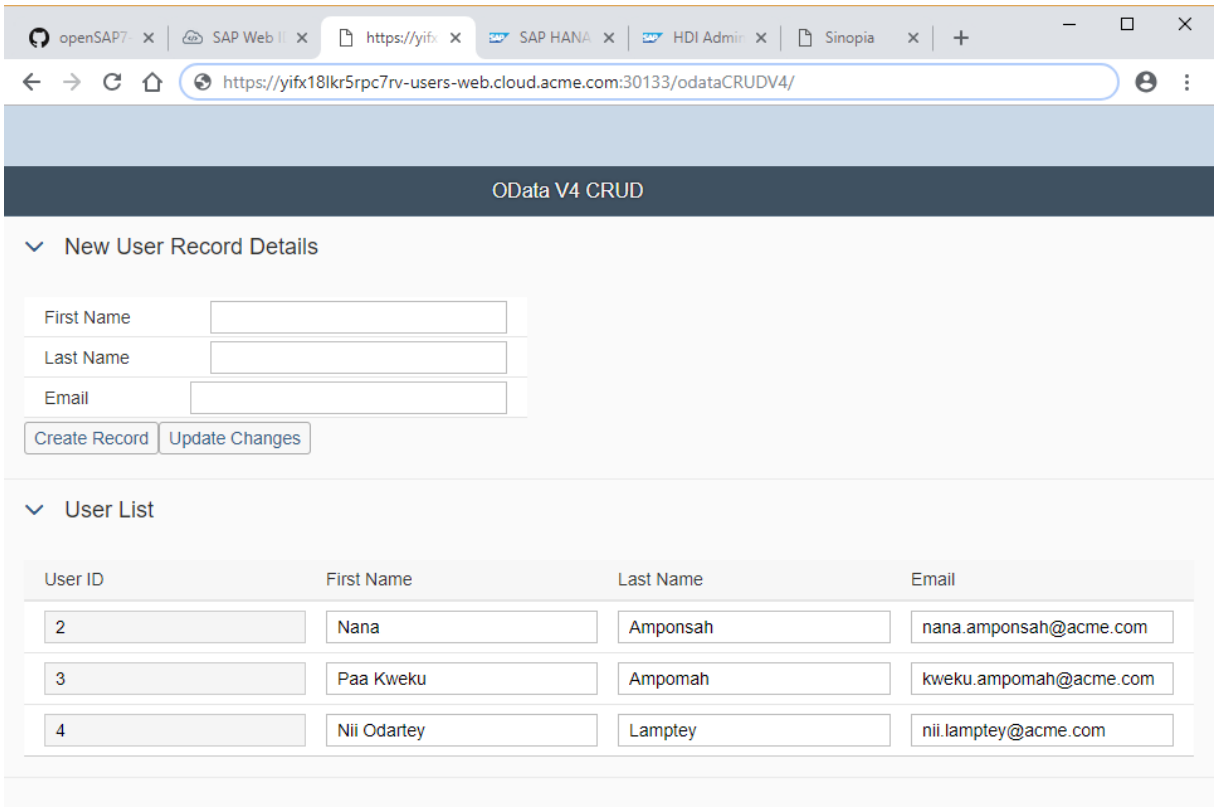
- Create a basic HTML5 application module
- Define the core data model and core services in SAP CAP's Core Data Services (CAP CDS)

> **i** Note
>
> SAP CAP Core Data Services (CAP CDS) is intended for use in SAP CAP scenarios and is not the same as the SAP HANA Core Data Services (CDS) component used in XS classic or XS advanced

applications, which focuses solely on the design-time representation of a data-persistence model. For more information about SAP CAP CDS, see *Developing SAP Cloud Platform Business Applications* in *Related Information* below.

- Define the back-end, business-logic services in Node.js
- Create the Web user interface using SAPUI5

In this tutorial, the simple Web-based user interface you build as the Node.js application's front-end includes features that enable you to create users in the SAP HANA database (*Create Record*), update a user's details (*Update Changes*), and view a list of all existing database users (*User List*), as illustrated in the following example:



→ Tip

Code snippets and examples used in these tutorials are taken from the source versions stored in the GitHub repository "openSAP7-xsa-node-2". For more details, see *Related Information* below.

Related Information

Create a Basic HTML5 Module for your SAP Cloud Platform Business Application [page 110]

OpenSAP XS advanced tutorial content on GitHub

Welcome to SAP Cloud Application Programming Model (CAP)

3.2.1 Create a Basic HTML5 Module for your SAP Cloud Platform Business Application

Set up the basic components of your SAP Cloud Platform business application.

Prerequisites

To complete this tutorial successfully, note the following prerequisites:

- You must have access to an SAP HANA system where the XS advanced model run-time platform is installed and running and the SAP HANA Deployment Infrastructure (HDI) is enabled.
- For the purposes of this tutorial, you must have access to the SAP Web IDE for SAP HANA. Generally speaking though, you can use any text editor to create a design-time artifact.

  > → Tip
  >
  > The SAP Web IDE provides access to the SAP HANA Database Explorer tools that enable you to view the contents of the catalog. You can also open the XS advanced Administration Cockpit to add and maintain platform services.

  - The SAP Web IDE must be able to connect to the Internet
  - The SAP Web IDE must be configured to use the following registries for the Node.js Package Manager (NPM):
    - Public NPM Registry (`http://registry.npmjs.org`)

    For more information, see *Related Information* below.
- Code snippets and examples used in this tutorial are taken from the source versions stored in the GitHub repository "openSAP7-xsa-node-2". For more details, see *Related Information* below.

Context

In the first part of this tutorial to develop a simple SAP Cloud Platform business application for deployment to XS advanced, you create the basic HTML5 module that is used to host and serve the application's front-end content, as described in the following steps:

Procedure

1. Start SAP HANA Web IDE for SAP HANA.

   The SAP Web IDE for SAP HANA is typically available at the following URL:

   `https://<HANA_HOST>:53075/`

> → Tip
>
> To display the URL for the SAP Web IDE for SAP HANA, open a command shell, log on to the XS advanced run time using the `xs login` command, and run the following command:
>
> ```
> xs app webide --urls
> ```

2. Create a new project for your SAP Cloud business-application.

   The project provides a convenient way to store and organize all the design-time artifacts that belong to your new application.

   a. Create the new project from an existing template.

      File > *New* > *Project from Template*

   b. Select the template you want to use as a model for your SAP Cloud business-application project.

      SAP Cloud Platform Business Application > *Next*

   c. Specify the new application project's name.

      In the setup Wizard's *Basic Information* tab, type **`users`** as the *Project Name* and choose *Next*.

   d. Specify the details of the new project.

      In the setup Wizard's *Project Details* tab, use the drop-down lists to provide the following information:

      - *Service Module (srv)*: Node.js
      - *Database Module (db)*: SAP HANA Database
      - *Space*: `<mySpace>`

        > → Tip
        >
        > Adjust to suit the name of the development space in your XS advanced organization.

      Choose *Next* and *Finish* to generate the new project and close the project-creation Wizard.

3. Adjust the names of the project's modules.

   The project-creation Wizard adds two modules to a new project: a database module and a service module. By default, the project name ("*users*" here) is appended to the start of the names of the *hdb* and *nodejs* modules, for example, *users-db* and *users-srv*. For the purposes of this tutorial, we need to change the names.

   a. Open the project's development descriptor (`mta.yaml`) file.

   b. Switch to the *Code Editor* view and rename the modules *users-db* and *users-srv* to *db* and *srv* respectively, as illustrated in the following example:

      > Sample Code
      >
      > ```
      > ID: users
      > _schema-version: '2.1'
      > version: 0.0.1
      > modules:
      >   - name: db
      >     type: hdb
      >     path: db
      >     parameters:
      >       memory: 256M
      >       disk-quota: 256M
      >     requires:
      > ```

```
      - name: users-hdi-container
  - name: srv
    type: nodejs
    path: srv
    parameters:
      memory: 512M
      disk-quota: 256M
```

c. Save your changes and switch back to the *MTA Editor* view.

4. Add a resource to handle user authentication when logging on to the new application later in the tutorial.

   At this point it is necessary to add the user-account-and-authentication service resource *users-uaa* to the `mta.yaml` file. This is because we plan to require user authentication to access our application.

   a. Name the new resource.

      In the *MTA Editor*, open the *Resources* tab and choose *[+]* to add a new resource to the application's development descriptor. This displays a new input field into which you can type the name of the resource: *users-uaa*.

   b. Define the resource *Type* as *com.sap.xs.uaa-space*.

   c. Save the changes and navigate back to the *srv (nodejs)* module in the MTA Editor.

      Ensure that the properties and dependencies match the following names and values:

      - *Properties*
        - Key: *UAA_SERVICE_NAME*
        - Value: *users-uaa*
      - *Requires*
        - Name: *users-hdi-container (resource)*
        - Name: *users-uaa (resource)*
        - Name: *db (module)*

5. Add and configure an HTML5 module to host and serve the application's front-end content.

   a. Right-click the root module (*users*) of your application project and choose *New* > *Basic HTML5 Module* .

   b. Name the new module *web* and choose *Next*.

   c. Define the module's requirements.

      In the *Modules* tab, select your new *web* module, click *[+]* in the *Requires* section, and choose the newly created UAA resource *users-uaa (resource)* from the drop-down list. This creates the connection between the *web* module and the selected UAA resource (*users-uaa*).

   d. Map the application's routes and destinations in the application descriptor (`xs-app.json`).

      The `web` folder in our project contains the resources that will be served by the HTML5 module. The *Add Module* Wizard adds a simple `index.html` file to the resources folder, which displays the message “Hello World”. The HTML5 module is configured in the application descriptor file `xs-app.json`. This is the file we use to map application routes to the destinations defined in the development descriptor `mta.yaml`. We can also set up the authentication type and other options, too. For the moment, just change the `authenticationMethod` from `none` to `route`, as illustrated in the following example:

      Sample Code

      `xs-app.json`

```
{
```

```
    "welcomeFile": "index.html",
    "authenticationMethod": "route",
    "routes": [ ] }
```

6. Create an instance of the User Account and Authentication (UAA) service.

   The SAP Web IDE can provision some resource types, but it cannot currently provision the UAA resource type. To create an instance of the UAA service, use the *SAP HANA XS Advanced Cockpit* as described in the following steps:

   a. Launch the *SAP HANA XS Advanced Cockpit* by choosing *Tools* > *SAP HANA XS Advanced Cockpit*.
   b. Navigate to your organization and space, for example, *myOrg* > *mySpace*.
   c. Expand *Services*, choose *Service Marketplace*, and *Authorization & Trust Management*.
   d. In the *Instances* tab, choose *New Instance*.
   e. Use the *Plan* drop-down list to specify the service plan *space* and choose *Next* and *Next*.
   f. Provide a name for the new service instance; type **`users-uaa`** and choose *Finish*.

7. Build the Web application and deploy it to the XS advanced server.

   Select the `web/` folder in your application project and, in the tool bar, choose *[|>]* (Run) or press `Alt` + `F5`, which performs a build and deploys the Web application to the server. If the build is successful, the deployed application opens a new Web browser tab and displays the application's default Web page.

   > → Tip
   >
   > It can take a minute or two for the first build, deploy, or run operation to complete. Upon completion, you should see that the application's status has changed to *Running* and the *Run Console* in the SAP Web IDE displays a link to the application and a link to the run logs.

8. Check the application is running correctly.

   If authentication is successful, you should see a button *[Hello World!]* displayed in a new tab in the Web browser.

   > ℹ Note
   >
   > If you do not see the application running, it might be necessary to configure the Web Browser to allow pop-up windows for the new application. You will need to disable the pop-up blocker and choose the option *Always allow pop-ups from* `<myAppURL>`. You can either open the application's URL again or run the *web* module again.

Related Information

Working with the SAP Cloud Application Programming Model [page 108]
The NPM Registry (XS Advanced Developer Guide)
OpenSAP XS advanced tutorial content on GitHub

3.2.2 Create the Core Data Services for your SAP Cloud Platform Business Application

Use CD and S to create the basic data model for your SAP Cloud Platform Business Application.

Prerequisites

To complete this tutorial successfully, note the following prerequisites:

- You must have access to an SAP HANA system where the XS advanced model run-time platform is installed and running and the SAP HANA Deployment Infrastructure (HDI) is enabled.
- For the purposes of this tutorial, you must have access to the SAP Web IDE for SAP HANA. Generally speaking though, you can use any text editor to create a design-time artifact.

  > → Tip
  >
  > The SAP Web IDE provides access to the SAP HANA Database Explorer tools that enable you to view the contents of the catalog. You can also open the XS advanced Administration Cockpit to add and maintain platform services.

  - The SAP Web IDE must be able to connect to the Internet
  - The SAP Web IDE must be configured to use the following registries for the Node.js Package Manager (NPM):
    - Public NPM Registry (`http://registry.npmjs.org`)

    For more information, see *Related Information* below.
- You must have completed the following tutorials already: they contain artifacts that this tutorial depends on:
  - *Create a Basic HTML5 Module for your SAP Cloud Platform Business Application*
- Code snippets and examples used in this tutorial are taken from the source versions stored in the GitHub repository "openSAP7-xsa-node-2". For more details, see *Related Information* below.

Context

SAP HDI enables you to manage database catalog objects using design-time artifacts but in a way that allows multiple copies (and versions) of the same core objects to be used on the same SAP HANA database at the same time. HDI also introduces the concept of the container as an abstraction of the database schema; the container dynamically generates the schema along with a container-specific database user who owns all objects in the container, and a password for the database user. With this infrastructure in place, XS-advanced-based services then only need access to the container; they do not need to know the actual schema, the technical user, or user password since this information is all stored in the container definition.

In the second part of this tutorial to develop a simple SAP Cloud Platform business application for deployment to XS advanced, you create the basic code data services used by the application's business logic, as described in the following steps:

Procedure

1. Check the version of the plug-in library used by SAP HDI to build and deploy database artifacts.

   The project-creation Wizard adds an entry to the SAP HDI container's configuration file (`.hdiconfig`) which defines the version of the build plug-in to use when generating database objects; this version must be at least `2.0.30.0` (HANA 2.0 SPS 03). The `.hdiconfig` file is not visible by default.

   a. Show all hidden files in your application project.

      In the SAP Web IDE's tool bar, choose *View* > *Show Hidden Files* and open the `.hdiconfig` file, which is located in the folder `users/db/src`. The entry for the SAP HDI plug-in version should look like the following example:

      Sample Code

      ```
      {
          "plugin_version": "2.0.30.0",
          "file_suffixes": {
              "hdbsynonym": {
                  "plugin_name": "com.sap.hana.di.synonym"
              },
      [...]
      }
      ```

2. Define the application's data-model.

   For the purposes of this tutorial, we need to change some of the sample content added to the default files by the project-creation Wizards. In the application's database module (*db*), open the file `db/data-model.cds` and replace the default template content with the following CDS code, which defines a simple table (entity).

   Sample Code

   ```
   Entity USER_DETAILS {
     EMAIL: String(255) @title: 'Email Address';
     FIRSTNAME: String(40) @title: 'Given Name';
     LASTNAME: String(40) @title: 'Family Name';
       key USERID: Integer @title: 'User Id';
   }
   ```

   → Tip

   This CDS data-model file is available on the openSAP GitHub Web site at `https://github.com/matteogp/openSAP7-xsa-node-2/blob/master/ex1/ex1_2`.

3. Define a service that uses your application's CDS data-model definition.

   For the purposes of this tutorial, we need to change some of the sample content added to the default files by the project-creation Wizards. In the application's service module (*srv*), open the file `srv/`

`myservice.cds` and replace the default template content with the following code, which defines a simple table (entity).

> ### Sample Code
>
> ```
> using USER_DETAILS as UserDetails from '../db/data-model';
> service CatalogService {
>     entity User @(
>             title: '{i18n>userService}',
>             Capabilities: {
>                 InsertRestrictions: {Insertable: true},
>                 UpdateRestrictions: {Updatable: true},
>                 DeleteRestrictions: {Deletable: true}
>             }
>         ) as projection on UserDetails;
> }
> ```

> ### Note
>
> This CDS service-definition file is available on the openSAP GitHub Web site at `https://github.com/matteogp/openSAP7-xsa-node-2/blob/master/ex1/ex1_3`.

4. Build the CDS database-specific files.

   Save all the `.cds` files in the *db* module, right-click the *users* project, and choose *Build* > *Build CDS*.

   > ### Tip
   >
   > The *Build CDS* operation does not create anything in the database; it simply "compiles" the CDS files into SAP-HANA-specific, HDI-compliant `.hdbcds` files and stores the generated files in the project folder `db/src/gen`: in this case a file called `USER_DETAILS.hdbcds`.

5. Build the database.

   Use the CDS artifacts generated in the previous step to build the database itself. Right-click the `db` folder in the `users` application project and choose *Build* > *Build*.

   > ### Note
   >
   > Check the *Run Console* at the bottom of the SAP Web IDE to view the build's progress:
   >
   > ```
   > 08:31:57 AM (DIBuild) ********** End of /users/db Build Log **********
   > 08:31:57 AM (DIBuild) Build results link: https://
   > webide.hana.acme.com:30573/[...]
   > 08:31:57 AM (Builder) Build of /users/db completed successfully.
   > ```

6. Set up the database-container view.

   You can use the *Database Explorer* to view the catalog objects generated in the application's SAP HDI container.

   a. Open the *Database Explorer*.

      Choose *Tools* > *Database Explorer* or click the Database Explorer icon in the tools bar on the left-hand side of SAP Web IDE for SAP HANA.

   b. Add a database connection if necessary.

If the Database Explorer is empty and has no connections to a database, a connection dialog is displayed: Choose *Yes* to add a new connection. Otherwise choose *[+]* (the Add button).

Choose *HDI Container* from the *Database Type* drop-down list, enter the following details and choose *OK*:

- In *HDI Containers*: Type **`users`**
- In *Name to Show in Display*: Type **`users_db`**

The Database Explorer should now display the database container "*users_db*" to which your application's database artifacts are deployed.

7. View the CDS entity (table) that you created in a previous step.

   In the Database Explorer, expand the HDI container *users_db* and choose *Tables*; the table *USER_DETAILS* you created in a previous step is displayed at the bottom of the pane. Double-click *USER_DETAILS* to display the table definition, which contains the following four columns: *EMAIL*, *FIRSTNAME*, *LASTNAME*, and *USERID*.

   > **Note**
   >
   > This table is the same table that we interact with later on in this tutorial, when we define our Node.js back end. The *Open Data* button enables you to view the data stored in the table, although at this point the table is still empty.

8. Create a database sequence to increment the User ID for each new user added to the table.

   Navigate to the *Development* perspective in SAP Web IDE, open up the *users* project, and perform the following steps:

   a. In the *srv* module, create a new folder called `sequences/`.
   b. Add the file `userSeqId.hdbsequence` to the new `sequences/` folder.
   c. Add the following code to the new `userSeqId.hdbsequence` file.

      The following sequence file enables you to uniquely increment each user ID as it is created in our database:

      > **Sample Code**
      >
      > Contents of `userSeqId.hdbsequence`
      >
      ```
      SEQUENCE "userSeqId" START WITH 1 MAXVALUE 1999999999
      RESET BY SELECT IFNULL(MAX(USERID), 1) + 1 FROM USER_DETAILS
      ```

      > **Note**
      >
      > This sequence definition file is available on the openSAP GitHub Web site at `https://github.com/matteogp/openSAP7-xsa-node-2/blob/master/ex1/ex1_6`.

9. Save all files and build the database (*db*) module.

Related Information

Create the Node.js Back-end Services for your SAP Cloud Platform Business Application [page 118]

Working with the SAP Cloud Application Programming Model [page 108]
The NPM Registry (XS Advanced Developer Guide)
OpenSAP XS advanced tutorial content on GitHub

3.2.3 Create the Node.js Back-end Services for your SAP Cloud Platform Business Application

Use Node.js to define the back-end services for your SAP Cloud Platform Business Application.

Prerequisites

To complete this tutorial successfully, note the following prerequisites:

- You must have access to an SAP HANA system where the XS advanced model run-time platform is installed and running and the SAP HANA Deployment Infrastructure (HDI) is enabled.
- For the purposes of this tutorial, you must have access to the SAP Web IDE for SAP HANA. Generally speaking though, you can use any text editor to create a design-time artifact.

  > → Tip
  >
  > The SAP Web IDE provides access to the SAP HANA Database Explorer tools that enable you to view the contents of the catalog. You can also open the XS advanced Administration Cockpit to add and maintain platform services.

  - The SAP Web IDE must be able to connect to the Internet
  - The SAP Web IDE must be configured to use the following registries for the Node.js Package Manager (NPM):
    - Public NPM Registry (`https://registry.npmjs.org`)

    For more information, see *Related Information* below.
- You must have completed the following tutorials: they contain artifacts that this tutorial depends on:
  - *Create a Basic HTML5 Module for your SAP Cloud Platform Business Application*
  - *Create the Core Data Services for your SAP Cloud Platform Business Application*
- Code snippets and examples used in this tutorial are taken from the source versions stored in the GitHub repository "openSAP7-xsa-node-2". For more details, see *Related Information* below.

Context

In the third part of this tutorial to develop a simple SAP Cloud Platform business application for deployment to XS advanced, you use Node.js to define the back-end, business-logic services for the SAP Cloud Platform business application, as described in the following steps:

Procedure

1. Add a new service provider to the application's development descriptor (`mta.yaml`).
   a. Open the application project's `mta.yaml` file, and in the *MTA Editor* select the *web* module.
   b. In the *Requires* tab, choose *[+]* and add *srv_api* as a new provider as part of the group *destinations*.
   c. Define the following properties (keys and values) for the new *srv_api* service.

Name	Group	Key	Value
srv_api (provider)	destinations	name	srv_api
		url	~{url}
		forwardAuthToken	true

2. Create a central Node.js file to hold the reusable functions required by other services and files.
   a. In the *srv* module of the *users* project, create a new folder called *utils*.

      Right click the *src* module and choose *New* > *Folder* in the context menu.
   b. In the new `utils` folder, create a new file called `util.js`.

      Right click the `utils` folder and choose *New* > *File* in the context menu.
   c. Add the following code to the new `util.js` file.

Sample Code

```
/*eslint no-console: 0, no-undef: 0, no-unused-vars: 0, no-shadow: 0,
  quotes: 0, no-use-before-define: 0, new-cap:0 */
'use strict';
var xsenv = require('@sap/xsenv');
module.exports = {
    getLocale: function (req) {
        let langparser = require("accept-language-parser");
        let lang = req.headers["accept-language"];
        if (!lang) {
            return null;
        }
        var arr = langparser.parse(lang);
        if (!arr || arr.length < 1) {
            return null;
        }
        var locale = arr[0].code;
        if (arr[0].region) {
            locale += "_" + arr[0].region;
        }
        return locale;
    },
    callback: function (error, res, message) {
```

```
        if (error) {
            res.writeHead(500, {
                'Content-Type': 'application/json'
            });
            res.end(JSON.stringify({
                message: message
            }));
        } else {
            res.writeHead(200, {
                'Content-Type': 'application/json'
            });
            res.end(JSON.stringify({
                message: message
            }));
        }
    },
    appconfig: function () {
        var services = xsenv.getServices({
            jobscheduler: {
                tag: 'jobscheduler'
            }
        }).jobscheduler;
        return {
            timeout: 15000,
            user: services.user,
            password: services.password,
            baseURL: services.url
        };
    },
    isAlphaNumeric: function (str) {
        var code, i, len;
        for (i = 0, len = str.length; i < len; i++) {
            code = str.charCodeAt(i);
            if (!(code > 47 && code < 58) && // numeric (0-9)
                !(code > 64 && code < 91) && // upper alpha (A-Z)
                !(code > 96 && code < 123)) { // lower alpha (a-z)
                return false;
            }
        }
        return true;
    },
    isAlphaNumericAndSpace: function (str) {
        var res = str.match(/^[a-z\d\-_\s]+$/i);
        if (res) {
            return true;
        } else {
            return false;
        }
    },
    isValidDate: function (date) {
        console.log("date" + date);
        var timestamp = Date.parse(date);
        console.log("timsestamp" + timestamp);
        if (isNaN(timestamp) === true) {
            return false;
        }
        return true;
    }
};
```

→ Tip

The Node.js file `util.js` is also available on the openSAP GitHub Web site at `https://github.com/matteogp/openSAP7-xsa-node-2/blob/master/ex1/ex1_7`.

d. Save the file.

3. Enable the Node.js server to connect to (and interact with) the database.

   In this step we create a Node.js file called `dbPromises.js` that will contain the functions required to connect to the SAP HANA database.

   a. In the database module's `utils` folder, create a new file called `dbPromises.js`.

      Right click the `utils` folder and choose *New* > *File* in the context menu.

   b. Add the following code to the new `dbPromises.js` file.

      Sample Code

```
/*eslint no-console: 0, no-unused-vars: 0, no-shadow: 0, new-cap: 0,
dot-notation:0,
  no-use-before-define:0 */
/*eslint-env node, es6 */
"use strict";
module.exports = class {
    static createConnection() {
        return new Promise((resolve, reject) => {
            const xsenv = require("@sap/xsenv");
            let options = xsenv.getServices({
                hana: {
                    tag: "hana"
                }
            });
            var hdbext = require("@sap/hdbext");
            hdbext.createConnection(options.hana, (error, client) => {
                if (error) {
                    reject(error);
                } else {
                    resolve(client);
                }
            });
        });
    }
    constructor(client) {
        this.client = client;
        this.util = require("util");
        this.client.promisePrepare =
this.util.promisify(this.client.prepare);
    }
    preparePromisified(query) {
        return this.client.promisePrepare(query);
    }
    statementExecPromisified(statement, parameters) {
        statement.promiseExec = this.util.promisify(statement.exec);
        return statement.promiseExec(parameters);
    }
    loadProcedurePromisified(hdbext, schema, procedure) {
        hdbext.promiseLoadProcedure =
this.util.promisify(hdbext.loadProcedure);
        return hdbext.promiseLoadProcedure(this.client, schema,
procedure);
    }
    callProcedurePromisified(storedProc, inputParams) {
        return new Promise((resolve, reject) => {
            storedProc(inputParams, (error, outputScalar, results) => {
                if (error) {
                    reject(error);
                } else {
                    resolve({
                        outputScalar: outputScalar,
                        results: results
                    });
```

```
                }
            });
        });
    }
};
```

→ Tip

The Node.js file `dbPromises.js` is also available on the openSAP GitHub Web site at `https://github.com/matteogp/openSAP7-xsa-node-2/blob/master/ex1/ex1_8`.

c. Save the file.

4. Define the handlers used to create users in the SAP HANA database and update the corresponding user records.

   The code used to add users to the database should be stored in the folder `users/srv/lib/handlers`, which you must first create, as follows:

   a. In the *srv* module of the *users* project, create a new folder called `lib`.

      Right click the *src* module and choose *New* > *Folder* in the context menu.

   b. In the new `lib` folder, create a new folder called `handlers`.

      Right click the `lib` folder and choose *New* > *Folder* in the context menu.

   c. In the new `handlers` folder, create a new file called `index.js`.

      Right click the `handlers` folder and choose *New* > *File* in the context menu.

   d. Add the following code to the new `index.js` file.

      Sample Code

```
/*eslint no-console: 0, no-unused-vars: 0, no-undef:0, no-process-
exit:0, new-cap:0*/
/*eslint-env node, es6 */
"use strict";
const uuid = require("uuid/v4");
const cds = require("@sap/cds");
/**
 * Register handlers to events.
 * @param {Object} entities.User
 */
module.exports = function (entities) {
    const {
        catalog
    } = entities;
    function validateEmail(email) {
        var re =
            /^(([^<>()[\]\\.,;:\s@\"]+(\.[^<>()[\]\\.,;:\s@\"]+)*)|
(\".+\"))@((\[[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\])|(([a-zA-
Z\-0-9]+\.)+[a-zA-Z]{2,}))$/;
        return re.test(email);
    }
    this.on("CREATE", entities.User, async(User) => {
        console.log("Before User Create");
        const {
            data
        } = User;
        if (data.length < 1) {
            return null;
        }
        if (!validateEmail(data.EMAIL)) {
```

```
            throw new Error(`Invalid email for ${data.FIRSTNAME}. No
Way! E-Mail must be valid and ${data.EMAIL} has problems`);
        }
        const dbClass = require(global.__base + "utils/dbPromises");
        var client = await dbClass.createConnection();
        let db = new dbClass(client);
        const insStatement = await db.preparePromisified(
            `INSERT INTO USER_DETAILS
                    (FIRSTNAME, LASTNAME, EMAIL, USERID)
                    VALUES (?,?,?, "userSeqId".NEXTVAL)`
        );
        const updResults = await
db.statementExecPromisified(insStatement, [data.FIRSTNAME,
data.LASTNAME, data.EMAIL]);
        console.log('After User Create');
        const statement = await db.preparePromisified(
            `SELECT MAX(USERID) FROM USER_DETAILS`);
        const userResults = await
db.statementExecPromisified(statement, []);
        data.USERID = userResults[0]["MAX(USERID)"];
        console.log(JSON.stringify(data));
        return data;
    });
    this.on("UPDATE", entities.User, async(User) => {
        console.log("Before User Update");
        const {
            data
        } = User;
        if (data.length < 1) {
            return null;
        }
        const dbClass = require(global.__base + "utils/dbPromises");
        var client = await dbClass.createConnection();
        let db = new dbClass(client);
        const statement = await db.preparePromisified(
            `SELECT * FROM USER_DETAILS
                        WHERE USERID = ?`);
        const userResults = await
db.statementExecPromisified(statement, [data.USERID]);
        Object.keys(data).forEach(function (key) {
            userResults[0][key] = data[key];
        });
        const updStatement = await db.preparePromisified(
            `UPDATE USER_DETAILS SET
                        "FIRSTNAME" = ?,
                        "LASTNAME" = ?,
                        "EMAIL" = ?
                        WHERE "USERID" = ?`
        );
        const updResults = await
db.statementExecPromisified(updStatement, [userResults[0].FIRSTNAME,
userResults[0].LASTNAME, userResults[0].EMAIL,
            userResults[0].USERID
        ]);
        return userResults[0];
    });
};
```

→ Tip

The Node.js file `index.js` is also available on the openSAP GitHub Web site at `https://github.com/matteogp/openSAP7-xsa-node-2/blob/master/ex1/ex1_9`.

e. Save the file.

5. Create a new file called `server.js` that defines the primary business logic for our Node.js server back end.

   The file `users/srv/server.js` contains the primary logic that the Node.js server uses to handle any calls to our oData service and assign the appropriate routes. This concerns, in particular, those services that are required to interact with the exit handlers.

   a. In the module `srv/`, create a new file called `server.js`.

      Right click the `srv/` folder and choose *New* > *File* in the context menu.

   b. Add the following code to the new `server.js` file.

      Sample Code

```
/*eslint no-console: 0, no-unused-vars: 0, no-undef:0, no-process-
exit:0*/
/*eslint-env node, es6 */
"use strict";
const https = require("https");
const port = process.env.PORT || 3000;
const server = require("http").createServer();
const cds = require("@sap/cds");
//Initialize Express App for XSA UAA and HDBEXT Middleware
const xsenv = require("@sap/xsenv");
const passport = require("passport");
const xssec = require("@sap/xssec");
const xsHDBConn = require("@sap/hdbext");
const express = require("express");
https.globalAgent.options.ca = xsenv.loadCertificates();
global.__base = __dirname + "/";
global.__uaa = process.env.UAA_SERVICE_NAME;
//logging
var logging = require("@sap/logging");
var appContext = logging.createAppContext();
//Initialize Express App for XS UAA and HDBEXT Middleware
var app = express();
passport.use("JWT", new xssec.JWTStrategy(xsenv.getServices({
    uaa: {
        tag: "xsuaa"
    }
}).uaa));
app.use(logging.middleware({
    appContext: appContext,
    logNetwork: true
}));
app.use(passport.initialize());
var hanaOptions = xsenv.getServices({
    hana: {
        tag: "hana"
    }
});
//hanaOptions.hana.rowsWithMetadata = true;
app.use(
    passport.authenticate("JWT", {
        session: false
    }),
    xsHDBConn.middleware(hanaOptions.hana)
);
//CDS OData V4 Handler
var options = {
    driver: "hana"
};
Object.assign(options, hanaOptions.hana, {
    driver: options.driver
});
cds.connect(options);
```

```
var odataURL = "/odata/v4/opensap.hana.CatalogService/";
// Main app
cds.serve("gen/csn.json", {
        crashOnError: false
    })
    .at(odataURL)
    .with(require("./lib/handlers"))
    .in(app)
    .catch((err) => {
        console.log(err);
        process.exit(1);
    });
// Redirect any to service root
app.get("/", (req, res) => {
    res.redirect(odataURL);
});
//Start the Server
server.on("request", app);
server.listen(port, function () {
    console.info(`HTTP Server: ${server.address().port}`);
});
```

→ Tip

The Node.js file `server.js` is also available on the openSAP GitHub Web site at `https://github.com/matteogp/openSAP7-xsa-node-2/blob/master/ex1/ex1_10`.

c. Save the file.

6. Update the `srv/package.json` file to reflect the latest changes and additions to the project's *srv* module.

   Update the file `srv/package.json` to look like the following example:

Sample Code

```
{
    "name": "serve",
    "dependencies": {
        "@sap/cds": ">=2.8",
        "express": "^4.16.2",
        "@sap/xssec": "^2.1.12",
        "@sap/xsenv": "^1.2.9",
        "hdb": "^0.16.0",
        "@sap/hdbext": "4.7.5",
        "@sap/hana-client": "~2.3",
        "@sap/logging": "~4.0",
        "@sap/audit-logging": "2.2.4",
        "passport": "~0.4.0",
        "async": "latest"
    },
    "scripts": {
        "postinstall": "cds build/all --project .. --clean",
        "start": "node server.js"
    },
    "engines": {
        "node": "8.x",
        "npm": "6.x"
    },
    "cds": {
        "data": {
            "driver": "hana"
        }
    }
}
```

→ Tip

The JSON file `srv/package.json` is also available on the openSAP GitHub Web site at `https://github.com/matteogp/openSAP7-xsa-node-2/blob/master/ex1/ex1_11`.

7. Run the applications *srv* module.

   Right-click the *srv* module and choose *Run* > *Run as Node.js Application*.

   → Tip

   The first time you start a build (run or deploy) operation, it can take a few minutes to complete. However, you can follow the build progress in the *Run Console*, and on completion, a message is displayed showing the operation status.

```
12/3/18 10:48:18.582 AM [APP/2-0] OUT [cds] - serving CatalogService at /
odata/v4/opensap.hana.CatalogService/ - impl: <inline>
Application is running
```

   Before testing the running service, we need to make some changes to the application (router) configuration, as described in the following steps.

8. Define the routes and authentication methods required to access the application and its oData services.

   Update the application-router's configuration file `web/xs-app.json` to include the new routes `"/odata/v4/(.*)"` and `"/node(.*)"` as illustrated in the following example:

   Sample Code

```
{
    "welcomeFile": "index.html",
    "authenticationMethod": "route",
    "routes": [{
        "source": "/odata/v4/(.*)",
        "destination": "srv_api",
        "authenticationType": "xsuaa"
    }, {
        "source": "/node(.*)",
        "destination": "srv_api",
        "authenticationType": "xsuaa"
    }]
}
```

   → Tip

   This `xs-app.json` file is also available on the openSAP GitHub Web site at `https://github.com/matteogp/openSAP7-xsa-node-2/blob/master/ex1/ex1_12`.

9. Save all files and re-run the *web* module.

   Right-click the *web* module and choose *Run* > *Run as Web Application*.

> → Tip
>
> You can follow the start-up progress in the *Run Console*. On completion, a message is displayed showing the application status (`running`), and the running application *Hello World!* is displayed in a new Web-browser tab.

```
Restarting Module
Application is starting
Application is running
```

10. Test that the OData V4 service is running and available.

    In the Web-browser tab where the *web* application is running, replace the `index.html` part of the URL with the path to the OData service (`odata/v4/opensap.hana.CatalogService/$metadata`).

    ```
    https://myapp-users-web.acme.com:30133/index.html
    ```

    The modified URL should look like the following:

    ```
    https://myapp-users-web.acme.com:30133/odata/v4/opensap.hana.CatalogService/
    $metadata
    ```

    The OData metadata contains details of the service and underlying table, as illustrated in the following incomplete ([...]) excerpt:

    ```
    <edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx"
    Version="4.0">
    [...]
      <edmx:DataServices>
        <Schema xmlns="http://docs.oasis-open.org/odata/ns/edm"
    Namespace="CatalogService">
          <EntityContainer Name="EntityContainer">
            <EntitySet Name="User" EntityType="CatalogService.User"/>
          </EntityContainer>
          <EntityType Name="User">
            <Key>
              <PropertyRef Name="USERID"/>
            </Key>
              <Property Name="EMAIL" Type="Edm.String" MaxLength="255"/>
              <Property Name="FIRSTNAME" Type="Edm.String" MaxLength="40"/>
              <Property Name="LASTNAME" Type="Edm.String" MaxLength="40"/>
              <Property Name="USERID" Type="Edm.Int32" Nullable="false"/>
          </EntityType>
          <Annotations Target="CatalogService.User">
            <Annotation Term="Common.Label" String="{i18n>userService}"/>
          </Annotations>
        [...]
        </Schema>
      </edmx:DataServices>
    </edmx:Edmx>
    ```

Related Information

Working with the SAP Cloud Application Programming Model [page 108]

The NPM Registry (XS Advanced Developer Guide)

OpenSAP XS advanced tutorial content on GitHub

3.2.4 Create the SAPUI5 Web UI for your SAP Cloud Platform Business Application

Use SAPUI5 to create the Web user interface for your SAP Cloud Platform Business Application.

Prerequisites

To complete this tutorial successfully, note the following prerequisites:

- You must have access to an SAP HANA system where the XS advanced model run-time platform is installed and running and the SAP HANA Deployment Infrastructure (HDI) is enabled.
- For the purposes of this tutorial, you must have access to the SAP Web IDE for SAP HANA. Generally speaking though, you can use any text editor to create a design-time artifact.

  > → Tip
  >
  > The SAP Web IDE provides access to the SAP HANA Database Explorer tools that enable you to view the contents of the catalog. You can also open the XS advanced Administration Cockpit to add and maintain platform services.

  - The SAP Web IDE must be able to connect to the Internet
  - The SAP Web IDE must be configured to use the following registries for the Node.js Package Manager (NPM):
    - Public NPM Registry (`https://registry.npmjs.org`)

    For more information, see *Related Information* below.
- You must have completed the following tutorials already: they contain artifacts that this tutorial depends on:
  - *Create a Basic HTML5 Module for your SAP Cloud Platform Business Application*
  - *Create the Core Data Services for your SAP Cloud Platform Business Application*
  - *Create the Node.js Back-end Services for your SAP Cloud Platform Business Application*
- Code snippets and examples used in this tutorial are taken from the source versions stored in the GitHub repository "openSAP7-xsa-node-2". For more details, see *Related Information* below. For this tutorial we also need the following archives, which contain data required by the application:
  - `common.zip`
  - `odataCRUDV4.zip`

  > ℹ Note
  >
  > The archives are available for download on openSAP's GitHub repository in the following location: `https://github.com/matteogp/openSAP7-xsa-node-2/blob/master/ex1/`

Context

In the fourth and final part of this tutorial to develop a simple SAP Cloud Platform business application for deployment to XS advanced, you use SAPUI5 to create a Web application that provides the user interface

required to add a new user to the SAP HANA database, update the user's details in the database, and check if the user's e-mail address is valid, as described in the following steps:

Procedure

1. Import two file archives into the *users* application project.

   The archives `common.zip` and `odataCRUDV4.zip` contain data that the steps in this tutorial rely on.

   a. In the SAP Web IDE tool bar, choose *File* > *Import* > *File or Project*
   b. Import the files to the following locations in the `users` project structure and choose *OK*:

   > **i Note**
   >
   > Make sure you check the option *Extract Archive* to ensure that the archive's contents are available in the expected folder locations.

   - `common.zip`
     /users/web/resources/**common**
   - `odataCRUDV4.zip`
     /users/web/resources/**odataCRUDV4**

2. Add a new resource called *users-ui5* to the application's MTA development descriptor (`mta.yaml`).
   a. Open the application project's `mta.yaml` file, and navigate to the *Resource* tab.
   b. In the *Resources* tab, choose *[+]* and add a new resource called *users_ui5*.
   c. Define the details of the new *users_ui5* resource.

   Choose the resource type *org.cloudfoundry.managed-service* from the drop-down list and define the following parameter keys and values:

Parameter Key	Parameter Value
service	sapui5_sb
service-plan	sapui5-1.52

   > **i Note**
   >
   > Switch to the MTA's *Code Editor* view and back to the *MTA Editor* to refresh the screen and ensure the newly defined resource is available for selection in the list of resources.

3. Add the new *users_ui5* resource as a dependency for the project's *web* module.
   a. In the *MTA Editor* display the *Modules* tab and choose the *web html5* module.
   b. In the *Requires* list, choose *users-ui5 (resource)*.
   c. Save the changes.

In the MTA *Code Editor*, the development descriptor (`mta.yaml`) should now look like the following example:

> **→ Tip**
>
> You can find a complete version of this MTA development descriptor on the openSAP GitHub Web site at `https://github.com/matteogp/openSAP7-xsa-node-2/blob/master/ex1/ex1_13`.

Sample Code

`mta.yaml`

```
ID: users
_schema-version: '2.1'
version: 0.0.1
modules:
  - name: db
    type: hdb
    path: db
    parameters:
      memory: 256M
      disk-quota: 256M
    requires:
      - name: users-hdi-container
  - name: srv
    type: nodejs
    path: srv
    parameters:
      memory: 512M
      disk-quota: 256M
    provides:
      - name: srv_api
        properties:
          url: '${default-url}'
    requires:
      - name: users-hdi-container
      - name: users-uaa
      - name: db
    properties:
      UAA_SERVICE_NAME: users-uaa
  - name: web
    type: html5
    path: web
    requires:
      - name: users-uaa
      - name: srv_api
        group: destinations
        properties:
          name: srv_api
          url: '~{url}'
          forwardAuthToken: true
      - name: users-ui5
resources:
  - name: users-hdi-container
    properties:
      hdi-container-name: '${service-name}'
    type: com.sap.xs.hdi-container
  - name: users-uaa
    type: com.sap.xs.uaa-space
  - name: users-ui5
    type: org.cloudfoundry.managed-service
    parameters:
      service: sapui5_sb
      service-plan: sapui5-1.52
```

4. Update the application router configuration to take into account the SAPUI5 Web user interface.

   In the SAP Web IDE's *Development* perspective, open the application-router configuration file `xs-app.json` in the *web* module and add the following new route for `localdir`, which is highlighted in bold type in the following example:

   > → Tip
   >
   > You can find a complete version of this application-router configuration file (`xs-app.json`) on the openSAP GitHub Web site at `https://github.com/matteogp/openSAP7-xsa-node-2/blob/master/ex1/ex1_14`.

   > Sample Code
   >
   > `xs-app.json`

```
{
    "welcomeFile": "index.html",
    "authenticationMethod": "route",
    "routes": [{
        "source": "/odata/v4/(.*)",
        "destination": "srv_api",
        "authenticationType": "xsuaa"
    }, {
        "source": "/node(.*)",
        "destination": "srv_api",
        "authenticationType": "xsuaa"
    },{
        "source": "/(.*)",
        "localDir": "resources",
        "authenticationType": "xsuaa",
        "replace": {
            "pathSuffixes": [
                "index.html"
            ],
            "services": {
                "sapui5_sb": {
                    "tag": "ui5"
                }
            }
        }
    }]
}
```

5. Create the SAPUI5 backing service.

   Since it is not possible to create this SAPUI5 service using the SAP Web IDE; use the *SAP HANA XS Advanced Cockpit*, as described in the following steps:

   a. Start the *SAP HANA XS Advanced Cockpit* under *Tools* > *SAP HANA XS Advanced Cockpit*.
   b. Navigate to the organization and space where you are developing this *users* application.

      In the navigation pane, expand *Services*, choose *Service Marketplace*, and in the services pane, choose *sapui5_sb*.
   c. In the navigation pane, choose *Instances* and in the *Instances* pane choose *New Instance*
   d. In the *Create Instance* tab, choose *sapui5-1.52* from the *Plan* drop-down list and press *Next*. Press *Next* again to skip the definition of any optional parameters.
   e. In the *Confirm* tab of the *Create Instance* Wizard, type the name **`users-ui5`** in the *Instance Name* box and press *Finish*.

6. Test the *Create Record* feature in the *odataCRUDV4* user interface by adding a new user to the database.

   In this step, you start the SAPUI5 user interface of our Web application and use the new user interface to make changes in the database, for example, by defining details of a new user.

   a. Open the *Development* perspective of the SAP Web IDE and (re)start the *users* Web application.

      Right-click the *web* module in the *users* project and choose *Run* > *Run as Web Application*.

      When the Web application is running, the *[Hello World!]* button is visible in a new tab in the Web browser.

   b. Start the OData V4 CRUD user interface.

      In the Web browser tab where the *users* Web application is now running, replace "`index.html`" in the URL with the text "`odataCRUDV4/`", which is the path to the OData V4 user interface and refresh the browser (`Shift` + `F5`). The new URL should look like the following example:

      ```
      https://myapp-users-web.acme.com:30133/odataCRUDV4/
      ```

      The new Web user interface using SAPUI5 should be visible, which enables you to define details of a user, whom you want to add to the database, or modify a user's existing details.

   c. Add details of a new user to your SAP HANA database.

      Type the following user details into the appropriate fields in the user interface and choose *Create Record*:

Parameter	Parameter Value
First Name	**`Kweku`**
Last Name	**`Ampomah`**
Email	**`k.ampomah@acme.com`**

      An *Alert* dialog is displayed indicating that a new user has been created; the new user has a unique ID, for example, 2. Choose *OK*.

      The new user is displayed immediately in the *User List* of the ODATA V4 CRUD Web application.

      In the SAP Web IDE's *Run Console*, you should see messages similar to the following extract from a run log:

      ```
      Application is running
      [...]
      12/4/18 9:44:37.665 AM [APP/4.0] OUT [cds] - service CatalogService at /
      odata/V4/...
      12/4/18 9:44:58.838 AM [APP/4.0] OUT Before User Create
      12/4/18 9:44:58.862 AM [APP/4.0] OUT After User Create
      12/4/18 9:44:58.873 AM [APP/4.0] OUT
      {"USERID":2,"FIRSTNAME":"Kweku","LASTNAME":"Ampomah","EMAIL":"k.ampomah@acm
      e.com"}
      ```

7. Confirm that the new user has been added to the records stored in the `USER_DETAILS` database table.

> **i Note**
>
> The `USER_DETAILS` database table is the same table that we created in a previous tutorial, where we used CD and S to define our data model.

a. In the *Database Explorer*, expand the HDI container *users_db* and choose *Tables*; the table *USER_DETAILS* you created in a previous tutorial is displayed at the bottom of the pane.
b. Double-click *USER_DETAILS* to display the table definition in the details pane.
c. Choose *Open Data* to view the data stored in the table, which should contain details of the user you just added, as illustrated in the following example:

EMAIL	FIRSTNAME	LASTNAME	USERID
k.ampomah@acme.com	Kweku	Ampomah	2

8. Test the *Update Changes* feature in the *odataCRUDV4* user interface by modifying the details of an existing database user.

   In this step, you use the SAPUI5 user interface of our Web application to make changes to the details of a existing database user, for example, by changing the user's first name, as described in the following steps:

   a. In the Web browser tab where the *odataCRUDV4* UI is running, press F5 to refresh the Web application or restart it by running the application module again.
   b. Change some of the details the user you created in the last step.

      In the *User List*, change the first name of an existing user (in this example, from **`Kweku`** to **`Paa Kweku`**) and choose *Update Changes*:

Parameter	Parameter Value
First Name	**`Paa Kweku`**

      In the SAP Web IDE's *Run Console*, you should see messages similar to the following extract from the application's run log:

```
Application is running
[...]
12/4/18 9:53:14.665 AM [APP/4.0] OUT [cds] - service CatalogService at /
odata/V4/...
12/4/18 9:53:37.653 AM [APP/4.0] OUT
{"USERID":2,"FIRSTNAME":"Kweku","LASTNAME":"Ampomah","EMAIL":"k.ampomah@acm
e.com"}
12/4/18 9:53:37.136 AM [APP/4.0] OUT Before User Update
12/4/18 9:53:37.653 AM [APP/4.0] OUT {"USERID":2,"FIRSTNAME":"Paa
Kweku","LASTNAME":"Ampomah","EMAIL":"k.ampomah@acme.com"}
12/4/18 9:53:37.662 AM [APP/4.0] OUT After User Update
```

9. Confirm that the changes you made to the user details are reflected in the records stored in the *USER_DETAILS* database table.

> **→ Tip**
>
> If the *USER_DETAILS* database table is still open in the *Database Explorer*, press the refresh button in the table tools header.

If not, open the *Database Explorer*, locate and double-click the table *USER_DETAILS*, and choose *Open Data*.

> → Tip
>
> You might need to press the refresh button in the table header.

EMAIL	FIRSTNAME	LASTNAME	USERID
k.ampomah@acme.com	Paa Kweku	Ampomah	2

10. Confirm that the Web application tests the validity of a new user's e-mail address.

    In this step, you test the functionality that checks the validity of a new user's e-mail address when adding the new user to the database, for example, by specifying an invalid e-mail address.

    a. In the Web browser tab where the *odataCRUDV4* UI is running, press F5 to refresh the Web application or restart it by running the application module again.
    b. Add details of a new user to your SAP HANA database.

       Type the following user details into the indicated fields in the user interface and choose *Create Record*:

Parameter	Parameter Value
First Name	**`Bad`**
Last Name	**`Data`**
Email	**`not@email`**

       This time, the Web application does not display any alert dialog confirming that a new user was successfully created, but the new user *Bad Data* does appear in the Web application's user list, albeit with the *User ID* "0".

       In the SAP Web IDE's *Run Console*, you should see messages indicating what happened; the OData exit handler noticed the invalid e-mail address and, as a result, did not create the new user:

       ```
       Application is running
       [...]
       12/4/18 9:49:17.665 AM [APP/4.0] OUT [cds] - service CatalogService at /
       odata/V4/...
       12/4/18 9:49:38.838 AM [APP/4.0] OUT Before User Create
       [...]
       12/4/18 9:49:38.862 AM [APP/4.0] ERR Stacktrace: Error: Invalid email for
       Bad...
       ```

       In the *Database Explorer*, refresh the data in the *USER_DETAILS* table to check that no user with the name "*Bad Data*" is listed, which confirms the user was not added to the database.

       > → Tip
       >
       > If you refresh the Web page of the OData V4 DRUD application you used to add the new user, you will notice that the new user *Bad Data* no longer appears in the *User List*. This is because the list of users is fetched from the database each time the Web page is loaded.

Related Information

Working with the SAP Cloud Application Programming Model [page 108]
OpenSAP XS advanced tutorial content on GitHub
The NPM Registry [page 687]

4 Maintaining the XS Advanced Application Development & Deployment Descriptors

Development descriptors are used to generate deployment descriptors, which define the details required at application-deployment time.

The deployment description is contained in the application deployment "descriptor", which specifies what you want to build (and how) as well as where (and how) to deploy it, as follows:

MTA Development and Deployment Descriptors

File Name	Description	SAP Web IDE	MTAR Builder	XS CLI
`mta.yaml`	**Development** descriptor for a multi-target application (MTA). The information in the `mta.yaml` file provides instructions for the MTA development and build process.	Yes	Yes	-
`mtad.yaml`	**Deployment** descriptor for a multi-target application (MTA). The information in the `mtad.yaml` file provides instructions for the deploy service.	*	*	Yes

The information defined in the development descriptor (`mta.yaml`) is used by the SAP Web IDE for SAP HANA and the MTA archive (MTAR) builder to generate the `mtad.yaml` file that is required to deploy the application to the XS advanced run-time environment. The information in the generated **deployment** descriptor `mtad.yaml` file provides the instructions that are carried out when the application is deployed.

> **i** Note
>
> If you use SAP Web IDE for SAP HANA to develop an MTA, both the `mta.yaml` and the `mtad.yaml` are created automatically. The `mta.yaml` is generated when you create the application project, and this is where you specify details of the development and build process. The `mtad.yaml` deployment descriptor is created during the application project's build process.

MTAs are transported in the form of a compressed and digitally signed archive called an MTA archive or "MTAR". Deploying an MTA involves the deployment of each application module to its corresponding target run time. The MTA archive (for example, `myMTAarchive.mtar`) that is deployed conforms to the standard Java JAR specification. For this reason, an additional file, the `MANIFEST.MF` file, is also required for the deployment operation; the `MANIFEST.MF` contains a list of the files in the MTAR archive to be deployed.

The development and deployment descriptors for an multi-target application must be located in the root folder of the application project, as illustrated (in **bold text**) in the following example.

Example

Sample Code

```
AppName
|- db/
|  |- package.json
|  \- src/
|- web/
|  |- package.json
|  \- resources/
|- js/
|  |- start.js
|  |- package.json
|  \- src/
|- xs-security.json
\- mta[d].yaml       # Development [deployment] build manifest *
```

Note

* You only need to concern yourself with the contents of the `mtad.yaml` deployment descriptor if you are using command-line or custom tools; in these cases you must also ensure that the `mtad.yaml` is checked into version control along with the other application-project artifacts. If you are using SAP Web IDE for SAP HANA or the MTA Archive Builder, the `mtad.yaml` file is generated for you automatically and does not need to be managed with version-control tools.

Related Information

Create the MTA Description Files [page 137]
The MTA Development Descriptor [page 141]
The MTA Deployment Descriptor [page 147]

4.1 Create the MTA Description Files

Multi-target applications are defined in multiple descriptor files.

Context

A multi-target application (MTA) comprises multiple pieces of software called “modules” which all share a common lifecycle for development and deployment. An MTA is described in two descriptor files: the design-

time descriptor and the deployment descriptor. To create a multi-target application, perform the following steps:

Procedure

1. Create the application resource-file structure.
2. Create the **development** descriptor for the multi-target application.

   The development descriptor for an XS advanced MTA is defined in the design-time artifact `mta.yaml`, which must be placed in the root folder of your MTA project. The MTA development descriptor is used by the SAP Web IDE for SAP HANA to generate an MTA deployment descriptor (`mtad.yaml`).

   > **i** Note
   >
   > This step is only required if you are using the SAP Web IDE for SAP HANA to develop and deploy an MTA, and the SAP Web IDE for SAP HANA creates the `mta.yaml` file automatically when you create a new project.
   >
   > If you use command-line tools to deploy an MTA, you do not need an MTA **development** descriptor. However, for this reason, you must create the MTA **deployment** descriptor (`mtad.yaml`) manually.

3. Create the **deployment** descriptor for the multi-target application.

   The deployment descriptor for an XS advanced MTA is defined in the design-time artifact `mtad.yaml`, which must be placed in the root folder of your MTA project.

   > **i** Note
   >
   > This step is only required if you are using command-line tools to deploy an MTA. If you use the SAP Web IDE for SAP HANA to deploy an MTA, the deployment descriptor is created for you automatically from the MTA **development** descriptor (`mta.yaml`).

   The deployment description of an MTA is defined in the `mtad.yaml` file, which must be located in the application's root folder.

   The following restrictions apply to the MTA **Deployment** Descriptor (`mtad.yaml`):

   MTA Descriptor File Size Restrictions

MTA Component	Maximum Size
MTA archive	4 GB
MTA **module** content	4 GB
MTA **resource** content	1 GB
MTA deployment descriptor (`mtad.yaml`)	1 MB
MTA manifest (`MANIFEST.MF`)	1 MB

4. If required, create a deployment extension description for the multi-target application. (optional)

The **extension** of an MTA deployment description is defined in a file with the suffix `.mtaext`, for example, `production-config.mtaext`.

Related Information

Multi-Target XS Advanced Applications [page 139]
The MTA Development Descriptor [page 141]
The MTA Deployment Descriptor [page 147]

4.1.1 Multi-Target XS Advanced Applications

Multi-target applications collect multiple modules and resource references in a single, deployable archive.

In SAP HANA Extended Application Services, Advanced Model (XS Advanced), the applications you develop are so-called multi-target applications (MTA). An MTA is basically the natural evolution of the so-called "multi-part" application; an MTA, however, comprises multiple software "**modules**" that share a common lifecycle for development and deployment. Although these modules can be developed using different technology and languages and then deployed to different targets, they all serve (different aspects of) a particular purpose for the application users.

An MTA is any unit of deployable functionality. This can be a single module or a group of modules packaged into an MTA archive along with an MTA deployment descriptor. MTAs can be distributed across multiple containers and target platforms. The MTA approach enables you to adopt a common life-cycle process (from development to deployment) for all application artifacts to be deployed to the target platform, for example, Cloud Foundry or SAP HANA XS Advanced.

> → Tip
>
> MTAs also enable you to model and configure run-time dependencies.

You define a multi-target application in the following files:

- MTA deployment descriptor (`mtad.yaml`)
- MTA deployment **extension** descriptor (`myMTAextension.mtaext`)

> i Note
>
> The extension descriptor is optional; it is used to provide system-, service-, or module-specific details that are not known until deployment time.

> ! Restriction
>
> For the MTA deployment operation, locally specific modifications to the deployment configuration must be defined in an MTA **deployment-extension** file.

The MTA Model

The MTA model is a platform-independent description of the different modules that define an application, the dependencies between the modules, the configuration data the modules expose, and the resources the application modules require to run. The MTA model is specified using YAML, in descriptor files that accompany the development and deployment processes. The MTA model serves the following purposes:

1. It defines an application composed of multiple (heterogeneous) subcomponents.
2. It declares the resources that the application depends on at run time and (or) at deployment time.
3. It defines configuration variables (and their relationship), whose values distinguish different deployment scenarios for the application.

Many people find it useful to see the MTA model as a sort of formal contract between developers (using development tools) and the MTA deployer. The deployer is a tool that consumes a description of the MTA model and translates it into target platform-specific “native” commands that can be used in the following scenarios:

- The provisioning of run-time containers
- The creation and binding of resources, for example, "service instances" on Cloud Foundry or in SAP XS advanced
- The installation, running, or update of the application modules

> **i** Note
>
> Although the MTA model is not target-specific, the MTA deployer is multi-target aware. Indeed, even when targeting a single platform (for example, SAP XS advanced or Cloud Foundry), the MTA deployer is still needed to resolve the dependencies between the different application parts. Using an MTA deployer rather than the underlying “native” installation tools provides a single orchestrated application-deployment step which can span multiple run-time tiers on more than one target platform.

Since deployment (for example, for testing) is an integral part of the development process, development environments contain such functionality as well.

MTA Architecture

An SAP HANA XS advanced application that consists of UI and database modules, and even application code, is an example of an MTA. A Java EE application composed of Beans, Web and application modules, resource adapters etc., which are all subject to the same development lifecycle and deployed across multiple computing tiers, is another example of a typical MTA. Both the application developer and the landscape administrator want to be able to manage the development, versioning, deployment, and operation of MTAs as one logical unit.

SAP Cloud Platform introduces new distribution requirements for orchestrated cross-platform deployments. When acting as a Software-as-a-Service (SaaS) extension platform and employing Fiori as a Service (FaaS) concepts, application developers need to be able to distribute their applications across heterogeneous targets (for example, Java Virtual Machines (VM), the front-end server, or the SaaS back end component), each of which has its own deployment application programming interface (API), while providing a carefully managed, single application life cycle.

By adopting a model that does not required an application to depend on any formal target or technology, the MTA addresses the challenges of deploying applications to multiple targets by isolating the developer

from target-specific native tools. Instead, developers describe the application's modules, any dependencies to other modules, MTAs and (micro) services, and any required or exposed interfaces. Application lifecycle-management frameworks that are MTA-aware can validate, orchestrate, and automate the MTA deployment process both on-premise and on Cloud platforms.

MTA Development Tools

SAP Web IDE provides "MTA-aware" development support both in the context of XS advanced applications and Fiori-as-a-Service (FaaS) solutions. MTA-aware deployers for SAP Cloud Platform, XS advanced, and Cloud Foundry provide deployment-related services for these platforms; the services integrate with SAP product assembly and production, are integrated into traditional SAP transport tools, and support delivery via SAP Service Market Place for on-premise deployment.

> **!** Restriction
>
> The MTA-aware tools used in each of these different use cases plan to implement the full MTA specification over a period of time. For this reason, at any point in time, each tool might reflect only a subset of the features described here; support for the features described will depend on priorities derived from the corresponding use case. For more information about the current feature scope, refer to the documentation for the respective tool.

Related Information

Create the MTA Description Files [page 137]
The MTA Development Descriptor [page 141]
The MTA Deployment Descriptor [page 147]

4.1.2 The MTA Development Descriptor

Multi-target applications are defined in a design-time **development** descriptor.

The **development** descriptor (`mta.yaml`) is used to define the elements and dependencies of an XS advanced-compliant multi-target application (MTA), as illustrated in the following example:

> Sample Code
>
> MTA Development Descriptor (`mta.yaml`)

```
ID: com.acme.mta.sample
version: 1.0.1
modules:
   - name: pricing-ui
     type: nodejs
     path: ./ui
     requires:
       - name: thedatabase
```

```
    - name: pricing-backend
      type: java
      path: ./backend
      provides:
        - name: price_opt
          properties:
            protocol: http
            uri: myhost.mydomain
resources:
    - name: thedatabase
      type: com.sap.xs.hdi-container
```

> **i** Note
>
> The MTA development descriptor (`mta.yaml`) is used by SAP Web IDE and the MTA Archive Builder tool to generate the **deployment** descriptor (`mtad.yaml`) that is required to deploy an MTA to the XS advanced run-time platform. If you use command-line tools to deploy an MTA, you do not need an MTA **development** description (`mta.yaml`). However, you must create the MTA deployment descriptor (`mtad.yaml`) manually.

The MTA development descriptor (`mta.yaml`) includes the following main components:

- Global elements
  An identifier and version that uniquely identify the MTA, plus additional **optional** information such as: a description, the providing organization, and a copyright notice for the author.
- Modules
  A set of code or content which needs to be deployed to a specified location.
- Resources
  Something required by the MTA at run time but not provided by the MTA, for example, a managed service or an instance of a user-provided service.
- Properties
  Key-value pairs that must be made available to the respective module at run time
- Parameters
  Parameters are reserved variables that affect the implementation of the MTA-aware tools, such as the deployer.

Modules

A **module** is a set of code or content which needs to be deployed to a specified location. The `modules` section of the MTA development descriptor enables you to define a set of source artifacts of a certain type which resides in a file system and which can be addressed by a file path. The following elements are mandatory:

- `name`
  Must be unique within the MTA it identifies

Optional module attributes include: `type`, `properties`, and `parameters`, plus `requires` and `provides`.

Order of deployment

Modules and therefore applications are deployed in an order that is based on the dependencies they have on each other. This is done in order to ensure that an application that depend on other applications is deployed only after any dependent applications are already up and running. To determine the order of application

deployment and start up, the modules defined in the application development (and deployment) descriptor are sorted in such a way that the first module has the least number of (and preferably no) dependencies on other modules in the MTA, while the last one has the most dependencies. For example, the modules in the following deployment descriptor will be deployed in the order m3, m2, and finally m1:

> **i Note**
>
> The number of dependencies between the modules and resources has no influence the on the deployment order; during deployment, the creation of services always takes place before any of the modules is deployed.

> **‹≡› Sample Code**
>
> MTA Development Descriptor (`mta.yaml`)

```
ID: com.sap.xs2.sample
version: 0.1.0
modules:
  - name: m1
    type: java.tomcat
    requires:
      - name: m2
      - name: m3
  - name: m2
    type: java.tomcat
    requires:
      - name: m3
  - name: m3
    type: java.tomcat
    requires:
      - name: r1
      - name: r2

resources:
  - name: r1
    type: com.sap.xs.uaa
  - name: r2
    type: com.sap.xs.hdi-container
```

Circular dependencies

If two modules have a circular dependency (for example `m1` depends on `m2`, but `m2` also depends on `m1`, the parameter "`dependency-type`" can be specified to control the order in which the modules are deployed. Modules with parameter `dependency-type: hard` (`m1`, in the following example) are always deployed first.

> **i Note**
>
> The default setting for the dependency type `dependency-type: soft` for all modules. The obvious consequence of this default dependency setting is that the deployment order of modules with circular dependencies is arbitrary and cannot relied upon.

> **‹≡› Sample Code**
>
> MTA Development Descriptor (`mta.yaml`)

```
ID: com.sap.xs2.sample
version: 0.1.0
modules:
  - name: m1
```

```
    type: java.tomcat
    requires:
      - name: m2
    parameters:
      dependency-type: hard
  - name: m2
    type: java.tomcat
    requires:
      - name: m1
```

Resources

A **resource** is something which is required by the MTA at run time but not provided by the MTA, for example, a managed service or an instance of a user-provided service. In the development description's `resources` section, the following elements are mandatory:

- `name`
  Must be unique within the MTA it identifies

Optional resource attributes include: `type`, `description`, `properties`, and `parameters`. The resource `type` is one of a reserved list of resource types supported by the MTA-aware deployment tools, for example: `com.sap.xs.uaa`, `com.sap.xs.hdi-container`, `com.sap.xs.job-scheduler`; the **type** indicates to the deployer how to discover, allocate, or provision the real resource, for example, a managed service such as a database, or a user-provided service.

> **!** Restriction
>
> System-specific parameters for the deployment descriptor must be included in a so-called MTA deployment **extension** descriptor.

Referencing Resource-Configuration Parameters Defined in a Separate File

When you deploy an application, some of the application's resources may require additional configuration,which you can provide in the form of parameters. For example, for a resource of type `com.sap.xs.uaa`, it might be necessary to provide details of the authentication methods and authorization types to use for access to your application. Typically, this information is provided using parameters declared in the development descriptor (`mta.yaml`) file, as illustrated in the following example:

> Sample Code
>
> Configuration Parameters in the Development Descriptor

```
...
resources:
  - name: node-uaa
    type: com.sap.xs.uaa
    parameters:
      config:
        xsappname: myApp
        scopes:
         - name: $XSAPPNAME.Display
           Description: Display
         - name: $XSAPPNAME.Edit
           Description: Edit
         - name: $XSAPPNAME.Delete
```

```
            Description: Delete
...
```

Alternatively, you can define the configuration parameters in a separate file stored within the same project. In this example, security-related configuration parameters are declared in the security descriptor `xs-security.json`, and the file's location is specified as a value of the application resource's `path` parameter, as illustrated in the following code snippet:

> **Note**
>
> The values specified for `path` should be the **relative** path from the root directory of the MTA to be deployed to the file or directory containing the specified content, as illustrated in the following example:

Sample Code

Configuration Path in the Development Descriptor

```
...
resources:
  - name: nodejs-uaa
    type: com.sap.xs.uaa
    parameters:
      path: security/xs-security.json
...
```

In the example above, it is assumed that the referenced resource-configuration file `xs-security.json` contains the information illustrated in the following (incomplete) code snippet:

Sample Code

Excerpt from the Security Descriptor (`xs-security.json`)

```
{
  "xsappname" : "myApp",
  "scopes"     : [ {
                    "name" : "$XSAPPNAME.Display",
                    "description" : "display" },
                  {
                    "name" : "$XSAPPNAME.Edit",
                    "description" : "edit" },
                  {
                    "name" : "$XSAPPNAME.Delete",
                    "description" : "delete"  }
                ],
...
}
```

During the application build process, the information included in the file referenced in the `path` parameter (for example, referencing `xs-security.json`) is converted from JSON to YAML and inserted into the **deployment** descriptor (`mtad.yaml`) generated by the build, as illustrated in the following (incomplete) example of the deployment descriptor:

Sample Code

Excerpt from the generated Deployment Descriptor (`mtad.yaml`)

```
...
resources:
```

```
  - name: node-uaa
    type: com.sap.xs.uaa
    parameters:
      config:
        xsappname: myApp
        scopes:
         - name: $XSAPPNAME.Display
           description: display
         - name: $XSAPPNAME.Edit
           description: edit
         - name: $XSAPPNAME.Delete
           description: delete
...
```

Properties

The MTA development descriptor (`mta.yaml`) contains two types of properties: **modules** and **requires**, which are basically very similar.

The values of properties can be specified at design time, in the deployment description (`mta.yaml`). More often, though a property value is determined during deployment, where the value is either explicitly set by the administrator (for example, in an extension descriptor file) or inferred by the MTA deployer from a target platform configuration. When set, the deployer injects the property values into the module's environment. The deployment operation reports an error, if it is not possible to determine a value for a property.

Both kinds of properties (“module” and “required”) are injected into the module’s environment. In the “requires” section, properties can reference other properties declared in the corresponding “provides” section, for example, using the `~{}` syntax.

Parameters

Parameters are reserved variables that affect the behavior of the MTA-aware tools, such as the deployer. Parameters can be “system” (read-only) values, “write-only”, or “read-write” (default value can be overwritten). Each tool publishes a list of reserved parameters and their (default) values for its supported target environments. All parameter values can be referenced as part of other property or parameter value strings. To reference a parameter value, use the placeholder notation `${<parameter>}`. The value of a parameter which is “read-only” cannot be changed in descriptors; only its value can be referenced using the placeholder notation.

Examples of common read-only parameters are `user`, `app-name`, `default-host`, `default-uri`. The value of a writable parameter can be specified within a descriptor. For example, a module might need to specify a non-default value for a target-specific parameter that configures the amount of memory for the module’s target run-time environment.

Sample Code

Parameters and Placeholders

```
modules:
  - name: node-hello-world
    type: javascript.nodejs
```

```
    path: web/
    requires:
      - name: nodejs-uaa
      - name: nodejs
        group: destinations
        properties:
          name: backend
          url: ~{url}
          forwardAuthToken: true
    parameters:
      host: ${user}-node-hello-world
```

Related Information

Create the MTA Description Files [page 137]
The MTA Deployment Descriptor [page 147]

4.1.3 The MTA Deployment Descriptor

Description of the deployment options for a multi-target XS Advanced application.

The MTA **deployment** descriptor defines the prerequisites and dependencies for the deployment of a multi-target application. The deployment description of an XS Advanced-compliant MTA is specified in an `mtad.yaml` file.

> **i** Note
>
> If you use command-line tools to deploy an MTA, you must create the MTA deployment descriptor manually.

> Sample Code
>
> Simple MTA Deployment Description (`mtad.yaml`)

```
ID: com.acme.mta.sample
version: 1.0.1
modules:
   - name: pricing-ui
     type: javascript.nodejs
     requires:
       - name: thedatabase
   - name: pricing-backend
     type: java.tomcat
     provides:
       - name: price_opt
         properties:
           protocol: http
           uri: myhost.mydomain
resources:
   - name: thedatabase
     type: com.sap.xs.hdi-container
```

The deployment descriptor (`mtad.yaml`) for a multi-target application comprises the following high-level components:

- Global elements
  An identifier and version that uniquely identify the MTA, plus additional **optional** information such as: a description, the providing organization, and a copyright notice for the author.
- Modules
  The application modules contained in the MTA deployment archive
- Resources
  The external resource that the application modules (defined in the “modules” section) depend on
- Properties
  Key-value pairs that must be made available to the respective module at run time
- Parameters
  Reserved variables that affect the behavior of the MTA-aware tools, such as the deploy service.
- Metadata
  Provide additional information about the declared parameters and properties.
- Cross-MTA Dependencies
  Modules can have dependencies on modules in other MTAs and subscribe to MTA configuration details (and changes) stored in the configuration registry managed by the XS advanced (and CF) deploy service.
- Sizing Restrictions
  The restrictions for the MTA artifacts handled by the deploy service both on-premise (XS advanced model) and SAP Business Technology Platform (Cloud Foundry).
- Transporting XS advanced applications with CTS

Modules

The `modules` section of the deployment descriptor lists the names of the application modules contained in the MTA deployment archive; the following elements are mandatory:

- `name`
  Must be unique within the MTA it identifies

Optional module attributes include: `path`, `type`, `description`, `properties`, and `parameters`, plus `requires` and `provides` lists.

Order of Deployment

Modules and therefore applications are deployed in an order that is based on the dependencies they have on each other. This is done in order to ensure that an application that depend on other applications is deployed only after any dependent applications are already up and running.

In some cases, it is crucial that modules and therefore applications are deployed in a predictable and consistent order. To ensure this, the module-level attribute deployed-after has been introduced. It contains a list of module names. If a module has this attribute, it will be deployed only after the modules specified in the attribute have already been deployed. The relations described through this attribute are transitive, so if module A should be deployed after module B, and module B should be deployed after module C, then it means that module A should be deployed after module C.

In the following example, the `deployed-after` attributes guarantee that the `ui` module is deployed after the `backend` and `metrics` modules, and they in turn are deployed after the `hdi-content` module. Note that the

deployment order of the `backend` and the `metrics` modules is not specified in the attributes. This means that they can be deployed in any order.

Sample Code

MTA Deployment Descriptor (`mtad.yaml`)

```
ID: com.sap.sample
version: 0.1.0
_schema-version: "3.2.0"
parameters:
  enable-parallel-deployments: true
modules:
  - name: ui
    type: javascript
    deployed-after: [ backend, metrics ]
  - name: backend
    type: java
    deployed-after: [ hdi-content ]
    requires:
      - name: metrics
        properties: METRICS_URL: ~{url}
  - name: metrics
    type: javascript
    deployed-after: [ hdi-content ]
    provides:
      - name: metrics
        properties: url: ${default-url}
  - name: hdi-content
    type: hdi
```

Parallel Deployment

In the order-of-deployment example above, the global MTA parameter `enable-parallel-deployments` is set to `true`. This setting activates the parallel deployment of MTA modules that do not have any dependencies regarding the deployment order. If the parameter is missing or its value is `false`, the modules are deployed sequentially.

Circular dependencies

If two modules have a circular dependency (for example `m1` depends on `m2`, but `m2` also depends on `m1`, the parameter "`dependency-type`" can be specified to control the order in which the modules are deployed. Modules with parameter `dependency-type: hard` (`m1` in the following example) are always deployed first.

Note

The default setting for the dependency type `dependency-type: soft` for all modules. The obvious consequence of this default dependency setting is that the deployment order of modules with circular dependencies is arbitrary and cannot be relied upon.

Sample Code

MTA Deployment Descriptor (`mtad.yaml`)

```
ID: com.sap.xs2.sample
version: 0.1.0
modules:
  - name: m1
    type: java.tomcat
    requires:
```

```
      - name: m2
    parameters:
      dependency-type: hard
  - name: m2
    type: java.tomcat
    requires:
      - name: m1
```

Shared Module Entries

It is possible for multiple MTA modules to reference a single deployable application, for example, a code bundle in the MTA archive. This means that, during the deployment operation, the same piece of code can be executed separately in multiple application (or application instances) but with different parameters and properties. This results in multiple, deployed run-time modules based on the same source module, which are usually started with different configuration parameters. A development project can have one source folder which is referenced by multiple module entries in the MTA deployment descriptor `mtad.yaml`, as illustrated in the following example:

Code Syntax

Multiple MTA Module Entries in the Deployment Descriptor (`mtad.yaml`)

```
...
modules:
- name: fileloader-master
  type: nodejs
  path: js
  properties:
    FL_ROLE: master
- name: fileloader-worker
  type: nodejs
  path: js
  parameters:
    instances: 3
  properties:
   FL_ROLE: worker
...
```

If deployment is based on an MTA archive, it is not necessary to duplicate the code to have two different deployable modules; the specification for the MTA-module entry in `MANIFEST.MF` is extended, instead. The following (incomplete) example of a `MANIFEST.MF` show how to use a comma-separated list of module names to associate one set of deployment artifacts with all listed modules:

Code Syntax

Multiple MTA Modules Listed in the `MANIFEST.MF` Deployment Manifest

Resources

The application modules defined in the “modules” section of the deployment descriptor depend on Manifest-Version: 1.0 ... Name: js/ MTA-Module: fileloader-master, fileloader-worker ... **resources**. In the `resources` section, the following elements are mandatory:

- `name`
  Must be unique within the MTA it identifies

The MTA deployment descriptor also supports the use of the following **optional** resource attributes:

- `type`
  The resource **type** is one of a reserved list of resource types supported by the MTA-aware deployment tools, for example: `com.sap.xs.uaa`, `com.sap.xs.hdi-container`, `com.sap.xs.job-scheduler`; the **type** indicates to the HDI deployer how to discover, allocate, or provision the resource, for example, a managed service such as a database, or a user-provided service. A resource type is associated with a particular backing service (for example, "`hana`" or "`xsuaa`") and a corresponding service plan (for example, "`sbss`" or "`default`").

  > → Tip
  >
  > For the complete list of resource **types** and the associated services and service plans, see *MTA Deployment Descriptor Syntax* in *Related Information* below.

- `description`
- `properties`
- `parameters`
- `active`
  The resource type `active: [true | false]` determines whether the resource should be considered (or ignored) for provisioning or for service binding. Resources that are not active (`"false"`) are not processed and are ignored even if the "inactive" resource is defined in an application's `requires` list.

> ! Restriction
>
> System-specific parameters for the deployment descriptor must be included in a so-called MTA deployment extension descriptor.

Properties

The MTA deployment descriptor can contain two types of properties, which are very similar, and are intended for use in the **modules** or **resources** configuration, respectively. Properties can be declared in the deployment description both in the `modules:` configuration (for example, to define `provides:` or `requires:` dependencies), or in the `resources` configuration to specify `requires:` dependencies. Both kinds of properties (`modules:` and `requires:`) are injected into the module's environment. In the `requires:` configuration, properties can reference other properties that are declared in the corresponding `provides:` configuration, for example, using the `~{}` syntax.

The values of properties can be specified at design time, in the deployment description (`mtad.yaml`). More often, however, a property value is determined during deployment, where the value is either explicitly set by

the administrator, for example, in an deployment-extension descriptor file (`myDeployExtension.mtaext`), or inferred by the MTA deployer from a target-platform configuration. When set, the deployer injects the property values into the module's environment. The deployment operation reports an error, if it is not possible to determine a value for a property.

> → Tip
>
> It is possible to declare metadata for properties defined in the MTA deployment description; the mapping is based on the parameter or property keys. For example, you can specify if a property is **required** (`optional; false`) or can be modified `overwritable: true`.

Cross-References to Properties

To enable resource properties to resolve values from a property in another resource or module, a resource must declare a dependency. However, these "requires" declarations do not affect the order of the application deployment.

> ! Restriction
>
> It is not possible to reference **list** configuration entries either from resources or "subscription" functionalities (deployment features that are available to subscriber applications).

> ‹≡› Code Syntax
>
> Cross-References between Properties in the MTA Deployment Descriptor

```
modules:
  - name: java
    ...
    provides:
      - name: backend
        properties:
          url: ${default-url}/foo
resources:
  - name: example
    type: example-type
    properties:
      example-prop: my-example-prop
  - name: uaa
    type: uaa-type
    requires:
      - name: example
      - name: backend
          properties:
            prop: ~{url}
          parameters:
            param: ~{url}
    properties:
      pro1: ~{example/example-prop}
    parameters:
      config:
        app-router-url: ~{backend/url}        #See modules: (java) provides:
        example-prop: ~{example/example-prop} #See resources: (example)
properties:
```

Parameters and Placeholders

Parameters are reserved variables that affect the behavior of the MTA-aware tools, such as the deployer. Parameters can be “system”, “write-only”, or “read-write” (default value can be overwritten). Each tool publishes a list of system parameters and their (default) values for its supported target environments. All parameter values can be referenced as part of other property or parameter value strings. To reference a parameter value, use the placeholder notation `${<parameter>}`. The value of a system parameter cannot be changed in descriptors. Only its value can be referenced using the placeholder notation.

Examples of common read-only parameters are `user`, `app-name`, `default-host`, `default-uri`. The value of a writable parameter can be specified within a descriptor. For example, a module might need to specify a non-default value for a target-specific parameter that configures the amount of memory for the module’s runtime.

> → Tip
>
> It is also possible to declare metadata for parameters and properties defined in the MTA deployment description; the mapping is based on the parameter or property keys. For example, you can specify if a parameter is **required** (`optional; false`) or can be modified `overwritable: true`.

> ‹≡› Sample Code
>
> Parameters and Placeholders

```
modules:
  - name: node-hello-world
    type: javascript.nodejs
    path: web/
    requires:
      - name: nodejs-uaa
      - name: nodejs
        group: destinations
        properties:
          name: backend
          url: ~{url}
          forwardAuthToken: true
    parameters:
      host: ${user}-node-hello-world
```

Descriptors can contain so-called placeholders (also known as substitution variables), which can be used as sub-strings within property and parameter values. Placeholder names are enclosed by the dollar sign ($) and curly brackets ({}). For example: `${host}` and `${domain}`. For each parameter “`P`”, there is a corresponding placeholder `${P}`. The value of `<P>` can be defined either by a descriptor used for deployment, or by the deploy service itself.

> → Tip
>
> Placeholders can also be used without any corresponding parameters; in this scenario, their value cannot be overridden in a descriptor. Such placeholders are Read only.

Placeholders can be used to read the value of parameters. For example, the following placeholder can be used in a descriptor to get the CF / XS API URL: ${xs-api-url}. Placeholders can also be used in map and list values in properties and parameters sections of modules and resources, as illustrated in the following example:

Sample Code

```
resources:
  - name: uaa
    type: com.sap.xs.uaa
    parameters:
      config:
        users: [ "${generated-user}", "XSMASTER" ]
```

Metadata for Properties and Parameters

It is possible to declare metadata for parameters and properties defined in the MTA deployment description, for example, using the "`parameters-metadata:`" or "`properties-metadata:`" keys, respectively; the mapping is based on the keys defined for a parameter or property. You can specify if a property is required (`optional; false`) or can be modified (`overwritable: true`), as illustrated in the following (incomplete) example:

The `overwritable:` and `optional` keywords are intended for use in extension scenarios, for example, where a value for a parameter or property is supplied at deployment time and declared in a deployment-extension descriptor file (`myMTADeploymentExtension.mtaext`).

You can declare metadata for the parameters and properties that are already defined in the MTA deployment description. However, any parameters or properties defined in the `mtad.yaml` deployment descriptor with the metadata value `overwritable: false` cannot be overwritten by a value supplied from the extension descriptor. In this case, an error would occur in the deployment.

Code Syntax

Metadata for MTA Deployment Parameters and Properties

```
modules:
 - name: frontend
   type: javascript.nodejs
   parameters:
     memory: 128M
     domain: ${default-domain}
   parameters-metadata:
     memory:
       optional: true
       overwritable: true
     domain:
       overwritable: false
   properties:
     backend_types:
       order_management: sap-erp
       data_warehouse: sap-bw
   properties-metadata:
     backend_types:
       overwritable: false
       optional: false
```

> **Note**
>
> Parameters or properties can be declared as sensitive. Information about properties or parameters flagged as “sensitive” is not written as plain text in log files; it is masked, for example, using a string of asterisks (`********`).

Example Deployment Description File

In many cases, an MTA will consist of multiple modules, which have interdependencies. The following, slightly more complex example shows an MTA deployment description including three modules:

- A database model
- An SAP UI5 application (hello world)
- An application written in node.js

In this example, the UI5 application, “hello-world”, will use the environment-variable `<ui5_library>` as a logical reference to some version of UI5 on a public Website.

> **Sample Code**
>
> Complex MTA Deployment Description (`mtad.yaml`)

```
ID: com.acme.xs2.samples.javahelloworld
version: 0.1.0
modules:
  - name: hello-world
    type: javascript.nodejs
    requires:
      - name: uaa
      - name: java_details
        properties:
          backend_url: ~{url3}/
    properties:
      ui5_library: "https://sapui5.hana.acme.com/"

  - name: java-hello-world-backend
    type: java.tomee
    requires:
      - name: uaa
      - name: java-hello-world-db        # deploy ordering
      - name: java-hdi-container
    provides:
      - name: java_details
        properties:
          url3: ${url}                   # the value of the place-holder $
{url}
                                         # will be made known to the deployer
  - name: java-hello-world-db
    type: com.sap.xs.hdi
    requires:
      - name: java-hdi-container
resources:
  - name: java-hdi-container
    type: com.sap.xs.hdi-container

  - name: uaa
    type: com.sap.xs.uaa
    parameters:
```

```
      name: java-uaa                    # the name of the new service
instance
```

The example describes three modules, which define a specific deployment order: `hello-world` requires `java-details`, which is provided by `java-hello-world-backend`, which in turn requires `java-hello-world-db`. These modules depend on a several resources, which are provisioned by the deployer as service instances in XS advanced, for example a database service instance `java-hdi-container` and a user authentication and authorization (UAA) service instance`uaa`.

→ Tip

`java-uaa` specifies the name of the service instance to be created on deployment; it provides the `hello_world` application with an OAuth cllient ID and OAuth client secret which enable XS Advanced to support authentication and authorization for this application. If your application does not require authentication services, you do not need to include this type of dependency.

The `hello_world` UI application will need to know the "access-point" for the `java-hello-world-backend` module. However, since this type of URL is often platform-specific, the UI application uses a property value (`backend_url` in this example) which is evaluated by the deployer as a value-substitution of the variable `<~{url3}>`; the variable `<~{url3}>` is provided as a property of the backend module `java-hello-world-backend`.

ℹ Note

The value of `url3` is specified via the system parameter `default-url`, whose value is known to the deployer at deployment time.

Cross-MTA Dependencies

In addition to having dependencies on modules in the same MTA, modules can have dependencies on modules from **other** MTAs, too. For these so-called cross-MTA dependencies to work, the MTA that **provides** the dependencies must declare them as "public" (this is true by default). Use the `configuration` resource to declare that one module consumes configurations from other MTAs. Bear in mind that the declaration method requires the addition of a resource in the deployment descriptor; the additional resource defines the provided dependency from the other MTA.

⚠ Caution

It is not recommended to use this feature to store sensitive content (for example, user credentials) as the stored data is not encrypted.

There are two methods that you can use to declare that one module has a dependency on a module in a different MTA:

- `mta-provided` [page 157]
- `configuration` [page 158]

> **Note**
>
> Both declaration methods require the addition of a resource in the deployment descriptor; the additional resource defines the provided dependency from the other MTA.

Cross MTA Dependency Method 1: Resource Type “mta-provided”

The resource defined in this method contains the special parameters `mta-id` (the ID of the provider MTA), `mta-version` (the version of the provider MTA) and `mta-provides-dependency` (the name of the provides dependency in the provider MTA). In addition, the `mta-version` parameter can contain version ranges as described in the Semver specification.

You can declare metadata for the parameters and properties defined in the MTA deployment descriptor. Bear in mind that this method can only be used for cross-MTA dependency resolution, and does not allow the specification of the organization and space in which the “providing” MTA resides. As a result, a provided dependency will be “found” by this syntax, only if it is published by an MTA that is deployed in the same space as the consuming MTA.

> **Caution**
>
> Because of the limitations listed above, it is not recommended to use the “`mta-provided`” dependency-declaration method; whenever possible, use the alternative declaration method (“`configuration`”) instead.

The following example shows the dependency declaration in the deployment descriptor of the “consumer” MTA :

> **Sample Code**
>
> Consumer MTA Deployment Descriptor (`mtad.yaml`)

```
_schema-version: "2.0.0"
ID: com.sap.sample.mta.consumer
version: 0.1.0
modules:
  - name: consumer
    type: java.tomee
    requires:
      - name: message-provider
        properties:
          message: ~{message}
resources:
  - name: message-provider
    type: mta-provided
    parameters:
      mta-id: com.sap.sample.mta.provider
      mta-version: ">=1.0.0"
      mta-provides-dependency: message-provider
```

The following example shows the dependency declaration in the deployment descriptor of the “provider” MTA :

> **Sample Code**
>
> Provider MTA Deployment Descriptor (`mtad.yaml`)

```
_schema-version: "2.0.0"
ID: com.sap.sample.mta.provider
version: 2.3.0
```

```
modules:
  - name: provider
    type: javascript.nodejs
    provides:
      - name: message-provider
        public: true
        properties:
          message: "Hello! This is a message provided by application \"${app-
name}\", deployed in org \"${org}\" and space \"${space}\"!"
```

Cross-MTA Dependency Method 2: Resource Type “configuration”

This method can be used to access any entry that is present in the configuration registry. The parameters used in this cross-MTA declaration method are `provider-nid`, `provider-id`, `version`, and `target`. The parameters are all optional and are used to filter the entries in the configuration registry based on their respective fields. If any of these parameters is not present, the entries will not be filtered based on their value for that field. The `version` parameter can accept any valid Semver ranges.

When used for cross-MTA dependency resolution the `provider-nid` is always “`mta`”, the `provider-id` follows the format `<mta-id>:<mta-provides-dependency-name>` and the `version` parameter is the version of the provider MTA. In addition, as illustrated in the following example, the `target` parameter is structured and contains the name of the organization and space in which the provider MTA is deployed. In the following example, the placeholders `${org}` and `${space}` are used, which are resolved to the name of the organization and space of the consumer MTA. In this way, the provider MTA is deployed in the same space as the consumer MTA.

> **i** Note
>
> By default, all provided dependencies are public.

The following example shows the dependency declaration in the deployment descriptor of the “consumer” MTA :

> **‹≡›** Sample Code
>
> Consumer MTA Deployment Descriptor (`mtad.yaml`)

```
_schema-version: "2.0.0"
ID: com.sap.sample.mta.consumer
version: 0.1.0
modules:
  - name: consumer
    type: java.tomee
    requires:
      - name: message-provider
        properties:
          message: ~{message}
resources:
  - name: message-provider
    type: configuration
    parameters:
      provider-nid: mta
      provider-id: com.sap.sample.mta.provider:message-provider
      version: ">=1.0.0"
      target:
        org: ${org}     # Specifies the org of the provider MTA
        space: ${space}
```

→ Tip

If no target organization or space is specified by the consumer, then the current organization and space are used to deploy the provider MTA. If no match is found for the provider MTA in the specified target organization or space, then `org: <current-org>` and `space: SAP` is searched.

The following example shows the dependency declaration in the deployment descriptor of the “provider” MTA :

Sample Code

Provider MTA Deployment Descriptor (`mtad.yaml`)

```
_schema-version: "2.0.0"
ID: com.sap.sample.mta.provider
version: 2.3.0
modules:
  - name: provider
    type: javascript.nodejs
    provides:
      - name: message-provider
        public: true
        properties:
          message: "Hello! This is a message provided by application \"${app-
name}\", deployed in org \"${org}\" and space \"${space}\"!"
```

Cross-MTA Configuration Visibility

A “consumer” module must explicitly declare the organizations and spaces in which a “provider” is expected to be deployed, except if it is the same space as the consumer. The “provider” can define an allowlist that specifies the organizations and spaces from which the consumption of configuration data is permitted. It is not required to place on an allowlist all the provider's spaces in its own organization.

i Note

Previously, registry entries were visible from all organizations by default. Now, the default visibility setting is "visible within the current organization and all the organization's spaces".

Allowlists can be defined on various levels. For example, a visibility allowlist could be used to ensure that a provider's configuration data is visible in the local space only, in **all** organizations and spaces, in a defined list of organizations, or in a specific list of organization and space combinations.

The options for allowlists and the visibility of configuration data are similar to the options available for managed services. However, for visibility allowlists, space developers are authorized to extend the visibility of configuration data beyond the space in which they work, without the involvement of an administrator. An administrator is required to release service plans to certain organizations.

Visibility is declared on the provider side by setting the parameter `visibility:` (of type 'sequence'), containing entries for the specified organization (`org:`) and space (`space:`). If no `visibility:` parameter is set, the default visibility value `org: ${org}, space: '*'` is used, which restricts visibility to consumers deployed into all spaces of the provider's organization. Alternatively, the value `org: '*'` can be set, which allows to bindings from all organizations and spaces. The allowlist can contain entries at the organization level only. This, however, releases configuration data for consumption from all spaces within the specified organizations, as illustrated in the following (annotated) example.

→ Tip

Since applications deployed in the same space are always considered " friends", visibility of configuration data in the local space is always preserved, no matter which visibility conditions are set.

Sample Code

```
provides:
  - name: backend
    public: true
    parameters:
      visibility:          # a list of possible settings:
       - org: ${org}       # for local org
         space: ${space}   # and local space
       - org: org1         # for all spaces in org1
       - org: org2         # for the specified combination (org2,space2)
         space: space2
       - org: ${org}       # default: all spaces in local org
       - org: '*'          # all orgs and spaces
       - org: '*'
         space: space3     # every space3 in every org
```

The authorization model ensures the following rules apply:

- Only those users in the white-listed spaces can read or consume the provided configuration data.
- Only users with the role “SpaceDeveloper” in the configuration-data provider's space can modify (edit or delete) configuration data.

Plugins

The deployment service also supports a method that allows an MTA to consume **multiple** configuration entries per `requires` dependency; the configuration is provided in a `list` property, as illustrated in the following example:

i Posting Instructions

Sample Code

Multiple “requires” Dependencies in the MTA Deployment Descriptor (`mtad.yaml`)

```
_schema-version: "2.1"
ID: com.acme.framework
version: "1.0"
modules:
  - name: framework
    type: javascript.nodejs
    requires:
      - name: plugins
        list: plugin_configs
        properties:
          plugin_name: ~{name}
          plugin_url: ~{url}/sources
        parameters:
          managed: true # true | false. Default is false
resources:
```

```
  - name: plugins
    type: configuration
    parameters:
      target:
        org: ${org}
        space: ${space}
      filter:
        type: com.acme.plugin
```

The MTA deployment descriptor shown in the example above contains a module that specifies a "`requires`" dependency to a configuration resource. Since the `requires` dependency has a `list` property, the deploy service will attempt to find multiple configuration entries that match the criteria specified in the configuration resource.

→ Tip

It is possible to create a subscription for a **single** configuration entry, for example, where no "`list:`" element is defined in the required dependency.

The resource itself contains a `filter` parameter that is used to filter entries from the configuration registry based on their content. In the example shown above, the filter only matches entries that are provided by an MTA deployed in the current space, which have a `type` property in their content with a value of `com.acme.plugin`.

ℹ Note

The filter parameter can be used in combination with other configuration resource specific parameters, for example: `provider-nid`, `provider-id`, `target`, and `version`.

If the "`list`" element is missing, the values matched by the resources filter are **single** configuration entries – not the usual list of multiple configuration entries. In addition, if either no value or multiple values are found during the deployment of the consuming (subscribing) MTA, the deployment operation fails. If a provider (plug-in) contributes additional configuration details after subscriber applications have been deployed, the subscriber applications will not receive the new information immediately; they will be made aware of the new configuration details only when they are updated. Note, however, that the update operation will fail because multiple configuration entries will be available at that point.

The XML document in the following example shows some sample configuration entries, which would be matched by the filter if they were present in the registry.

⁖≡ Sample Code

MTA Configuration Entries Matched in the Registry

```
<configuration-entry>
  <id>8</id>
  <provider-nid>mta</provider-nid>
  <provider-id>com.sap.sample.mta.plugin-1:plugin-1</provider-id>
  <provider-version>0.1.0</provider-version>
  <target-space>2172121c-1d32-441b-b7e2-53ae30947ad5</target-space>
  <content>{"name":"plugin-1","type":"com.acme.plugin","url":"https://
xxx.mo.sap.corp:51008"}</content>
</configuration-entry>
<configuration-entry>
  <id>10</id>
  <provider-nid>mta</provider-nid>
  <provider-id>com.sap.sample.mta.plugin-2:plugin-2</provider-id>
  <provider-version>0.1.0</provider-version>
```

```
  <target-space>2172121c-1d32-441b-b7e2-53ae30947ad5</target-space>
  <content>{"name":"plugin-2","type":"com.acme.plugin"}</content>
</configuration-entry>
```

The JSON document in the following example shows the environment variable that will be created from the `requires` dependency defined in the example deployment descriptor above, assuming that the two configuration entries shown in the XML document were matched by the filter specified in the configuration resource.

> **i** Note
>
> References to non-existing configuration entry content properties are resolved to "`null`". In the example above, the configuration entry published for `plugin-2` does not contain a `url` property in its content. As a result, the environment variable created from that entry is set to "`null`" for `plugin_url`.

> Sample Code
>
> Application Environment Variable

```
plugin_configs: [
  {
    "plugin_name": "plugin-1",
    "plugin_url": "https://xxx.mo.sap.corp:51008/sources"
  },
  {
    "plugin_name": "plugin-2",
    "plugin_url": null
  }
]
```

Requires dependencies support a special parameter named "`managed`", which registers as a "subscriber" the application created from the module containing the `requires` dependency. One consequence of this registration is that if any new configuration entries are published in the configuration registry during the deployment of another MTA, and those new entries match the filter specified in the subscription of an application, then that application's environment would be updated, and the application itself would be restarted in order for it to see its new environment's state.

> → Tip
>
> When starting the deployment of an MTA (with the `xs deploy` command), you can use the special option --no-restart-subscribed-apps to specify that, if the publishing of configuration entries created for that MTA result in the update of a subscribing application's environment, then that application should **not** be restarted.

Service-Creation Parameters

Some services support additional configuration parameters in the `create-service` request; these parameters are passed in a valid JSON object containing the service-specific configuration parameters. The deploy service supports the following methods for the specification of service creation parameters:

- Method 1
  An entry in the MTA deployment descriptor (or the extension)

- Method 2
  A JSON file with the required service-configuration parameters

> **Note**
>
> If service-creation information is supplied both in the deployment (or extension) descriptor **and** in a supporting JSON file, the parameters specified directly in the deployment (or extension) descriptor override the parameters specified in the JSON file.

Service-Creation Parameters in the MTA Deployment Descriptor

The following example shows how to define the service-configuration parameters in the MTA deployment descriptor (`mtad.yaml`). If you use this method, all parameters under the special `config` parameter are used for the service-creation request.

> **Sample Code**
>
> Service-Configuration Parameters in the MTA Deployment Descriptor

```
resources:
  - name: java-uaa
    type: com.sap.xs.uaa
    parameters:
      config:
        xsappname: java-hello-world
```

Service-Creation Parameters in a JSON File

The following example shows how to define a service-configuration parameter for a service-creation request in a separate JSON file, in this case, `xsappname` in the MTA security descriptor (`xs-security.json`). Bear in mind that, with this method, there are dependencies on further configuration entries in other configuration files. For example, if you use this JSON method, an entry must be added to the application's `MANIFEST.MF` file, which defines the path to the JSON file containing the parameters as well as the name of the resource for which the parameters should be used.

> **Sample Code**
>
> Service-Configuration Information in the MTA Deployment Descriptor

```
resources:
  - name: java-uaa
    type: com.sap.xs.uaa
```

The MTA's security descriptor (`xs-security.json`) defines the service-creation parameter (`xsappname`) to use, as illustrated in the following example:

> **Sample Code**
>
> MTA Service-creation Parameters Defined in the `xs-security.json` File

```
{
  "xsappname": "java-hello-world"
}
```

The Java application's deployment manifest (`MANIFEST.MF`) should contain the path to the JSON file containing the service-creation parameter to resolve, in this case `xs-security.json`, as illustrated in the following example:

Sample Code

Service-Configuration Parameters in the MANIFEST.MF

```
Name: xs-security.json
MTA-Resource: java-uaa
Content-Type: application/json
```

The following example of an MTA deployment descriptor shows how to combine both methods to achieve the desired application-service creation on deployment:

Sample Code

Combined Service-Configuration Parameters in the MTA Deployment Descriptor

```
resources:
  - name: java-uaa
    type: com.sap.xs.uaa
    parameters:
      config:
        xsappname: java-hello-world
```

Service-Binding Parameters

Some services support additional configuration parameters in the `create-bind` request; these parameters are passed in a valid JSON object containing the service-specific configuration parameters. The deployment service supports the following methods for the specification of service-binding parameters:

- Method 1
  An entry in the MTA deployment descriptor (or the extension)
- Method 2
  A JSON file with the required service-configuration parameters

Note

If service-binding information is supplied both in the MTA's deployment (or extension) descriptor **and** in a supporting JSON file, the parameters specified directly in the deployment (or extension) descriptor override the parameters specified in the JSON file.

In the MTA deployment descriptor, the `requires` dependency between a module and a resource represents the binding between the corresponding application and the service created from them (if the service has a `type`). For this reason, the `config` parameter is nested in the `requires` dependency parameters, and a distinction must be made between the `config` parameter in the `modules` section and the `config` parameter used in the `resources` section (for example, when used for service-creation parameters).

Service-Binding Parameters in the MTA Deployment Descriptor

The following example shows how to define the service-binding parameters in the MTA deployment descriptor (`mtad.yaml`). If you use this method, all parameters under the special `config` parameter are used for the service-bind request.

Sample Code

Service-Binding Parameters in the MTA Deployment Descriptor

```
modules:
  - name: node-hello-world-backend
    type: javascript.nodejs
    requires:
      - name: node-hdi-container
        parameters:
          config:
            permissions: debugging
```

Service-Binding Parameters in a JSON File

The following example shows how to define the service-binding parameters for a service-bind request in a JSON file, in this case, the file `xs-hdi.json`. With this method, there are dependencies on entries in other configuration files. For example, if you use this JSON method, an additional entry must be included in the `MANIFEST.MF` file which defines the path to the JSON file containing the service-binding parameters as well as the name of the resource for which the parameters should be used.

Sample Code

Service-Binding Configuration in the MTA Deployment Descriptor

```
modules:
  - name: node-hello-world-backend
    type: javascript.nodejs
    requires:
      - name: node-hdi-container
```

The MTA's `xs-hdi.json` file contains the definition for the `"permissions"` parameter, as illustrated in the following example:

Sample Code

MTA Service-binding Parameters Defined in the `xs-hdi.json` File

```
{
  "permissions": "debugging"
}
```

The multi-target application's deployment manifest (`MANIFEST.MF`) should contain the reference to the `"requires"` dependency (`node-hdi-container`), as illustrated in the following example:

Note

To avoid ambiguities, the name of the module is added as a prefix to the name of the `requires` dependency; the name of the manifest attribute uses the following format: `<module-name>/<requires-dependency-name>`.

Sample Code

Service-Binding Parameters in the MANIFEST.MF

```
Name: xs-hdi.json
MTA-Requires: node-hello-world-backend/node-hdi-container
Content-Type: application/json
```

The following example of an MTA deployment descriptor shows how to combine both methods to achieve the desired application-service binding on deployment:

Sample Code

Combined Service-Binding Configuration and Parameters in the MTA Deployment Descriptor

```
modules:
  - name: node-hello-world-backend
    type: javascript.nodejs
    requires:
      - name: node-hdi-container
        parameters:
          config:
            permissions: debugging
```

Service Keys

You can use the `service-keys` parameter in the MTA deployment descriptor to configure the deployment service to create and update the corresponding service keys during the creation or update of a service instance. The special `service-keys` parameter must be added to the `resources` in the MTA deplyoment descriptor which represent services that support service keys.

The following example shows how to set up the configuration of the service keys in the applications deployment descriptor (`mtad.yaml`) file. Each entry for the `service-keys` parameter can include optional configuration parameters, which must be placed under the `config` parameter.

Sample Code

Service-Key Configuration in the MTA Deployment Descriptor

```
resources:
  - name: my-service-instance
    type: org.cloudfoundry.managed-service
    parameters:
      service-keys:
      - name: tool-access-key
        config:
          permissions: read-write
      - name: reader-endpoint
        config:
          permissions: read-only
```

Consuming Existing Service Keys

You can configure an application to use an existing service key as an alternative to a service binding. In this way, it is possible to enable an application to use service key's credentials and consume the corresponding service.

The existing service keys are modeled as a resource of type `org.cloudfoundry.existing-service-key`. An MTA module can depend on the defined service resource which results in the injection of the reused service-key credentials into the consuming application's environment.

In the following example, the Node.js application `myApp` has the environment variable `keycredentials` whose details are specified in the existing service key `test-key-1` which belongs to the existing service `service-a`. In this reuse-service-key configuration, the only parameter that is mandatory is `service-name` in the `resources` section.

Sample Code

Consuming an Existing Service-Key in the MTA Deployment Descriptor

```
modules:
  - name: myApp
    type: javascript.nodejs
    requires:
      - name: service-key-1
        parameters:
          env-var-name: keycredentials # Default value is service-key name
...
resources:
  - name: service-key-1
    type: org.cloudfoundry.existing-service-key
    parameters:
      service-name: service-a # service the key belongs to
      service-key-name: test-key-1 # Default value is the resource name
```

Service Tags

Some services provide a list of tags that are later added to the `<VCAP_SERVICES>` environment variable. These tags provide a more generic way for applications to parse `<VCAP_SERVICES>` for credentials. You can provide custom tags when creating a service instance, too. To inform the deployment service about custom tags, you can use the special `service-tags` parameter, which must be located in the `resources` definition that represent the managed services, as illustrated in the following example:

Sample Code

Defining Service Tags in the MTA Deployment Descriptor

```
resources:
  - name: nodejs-uaa
    type: com.sap.xs.uaa
    parameters:
      service-tags: ["custom-tag-A", "custom-tag-B"]
```

Note

Some service tags are inserted by default, for example, `xsuaa`, for the XS User Account and Authentication (UAA) service.

Namespaces

To prevent name clashes for applications and services contained in different MTAs, but deployed in the same space, the deployment service provides an option that enables you to add the MTA IDs in front of the names of the applications and services contained in those MTAs. For example, an application named "`com.sap.xs2.sample.app`" and a service named "`com.sap.xs2.sample.service`" are created when the --use-namespaces option is specified during the deployment of an MTA with the following deployment descriptor:

Sample Code

MTA Deployment Descriptor

```
ID: com.sap.xs2.sample
version: 0.1.0
modules:
  - name: app
    type: java.tomcat
resources:
  - name: service
    type: com.sap.xs.uaa
```

By default the "use namespaces" feature is disabled. However, specifying the --use-namespaces option as part of the `xs deploy` command allows you to enable it. If you want namespaces to be used for applications, but not for services, you can use the --no-namespaces-for-services option in combination with the --use-namespaces. For example, if you use the --use-namespaces and --no-namespaces-for-services options when deploying an MTA with the deployment descriptor shown in the sample code above, the deployment operation creates an application named "`com.sap.xs2.sample.app`" and a service named "`service`".

Application Versions and Deployment Consistency

The deployment service follows the rules specified in the Semantic Versioning Specification (Semver) when comparing the versions of a deployed MTA with the version that is to be deployed. If an MTA submitted for deployment has a version that is lower than or equal to an already deployed version of the same MTA, the deployment might fail if their is a conflict with the version rule specified in the deployment operation. The version rule is a parameter of the deployment process, which specifies which relationships to consider between the existing and the new version of an MTA before proceeding with the deployment operation. The version rules supported by the deploy service are as follows:

- HIGHER
  Only MTAs with versions that are bigger than the version of the currently deployed MTA, are accepted for deployment.
- SAME_HIGHER
  Only MTAs with versions that are higher than (or the same as) the version of the currently deployed MTA are accepted for deployment. This is the **default** setting for the version rule.
- ALL
  All versions are accepted for deployment.

Registration of Service URLs

An application can declare a service URL for automatic registration with the XS advanced controller; the URL is made available so that other applications can find out where (which URL) the application can be accessed. The following example shows the parameters to use in the MTA's deployment (or extension) descriptor to ensure that the corresponding application specifies that a service URL should be registered as part of its deployment process:

→ Tip

You can use placeholders `${}` in the service-URL declaration.

‹≡› Sample Code

```
modules:
  - name: jobscheduler-dashboard
    type: javascript.nodejs
    parameters:
      register-service-url: true
      service-name: job-scheduler-service-dashboard
      service-url: ${default-url}
```

Service-Broker Creation

Service brokers are applications that advertise a catalog of service offerings and service plans, as well as interpreting calls for creation, binding, unbinding, and deletion. The deploy service supports automatic creation (and update) of service brokers as part of an application deployment process. For example, an application can declare that a service broker should be created as part of its deployment process, by using the following parameters in its corresponding module in the MTA deployment (or extension) descriptor:

→ Tip

You can use placeholders `${}` in the service-URL declaration.

‹≡› Sample Code

```
- name: jobscheduler-broker
  properties:
    user: ${generated-user}
    password: ${generated-password}
  parameters:
    create-service-broker: true
    service-broker-name: jobscheduler
    service-broker-user: ${generated-user}
    service-broker-password: ${generated-password}
    service-broker-url: ${default-url}
```

The value of the "`${generated-user}`" and "`${generated-password}`" placeholders in the `properties` section is the same as in the `parameters` section. The service-broker application uses this mechanism to inject the user-password credentials.

The `create-service-broker` parameter should be set to true if a service broker must be created for the specified application module. You can specify the name of the service broker with the `service-broker-name` parameter; the default value is `${app-name}`The `service-broker-user` and `service-broker-password` are the credentials that will be used by the controller to authenticate itself to the service broker in order to make requests for creation, binding, unbinding and deletion of services. The `service-broker-url` parameter specifies the URL on which the controller will send these requests.

> **i** Note
>
> During the creation of the service broker, the XS advanced controller makes a call to the service-broker API to inquire about the services and plans the service broker provides. For this reason, an application that declares itself as a service broker must implement the service-broker application-programming interface (API). Failure to do so might cause the deployment process to fail.

Module-Based User-Provided Service Creation

SAP HANA XS advanced allows you to expose an application as a user provided service which other applications can bind to. The following example of the application's the MTA deployment (or extension) descriptor shows the syntax required:

> Sample Code

```
modules:
  - name: provider
    properties:
      userID: ${generated-user}
      password: ${generated-password}
    parameters:
      create-user-provided-service: true
      user-provided-service-name: techical-user-provider
      user-provided-service-config:
        userID: ${generated-user}
        password: ${generated-password}
        url: ${default-url}
```

The `create-user-provided-service` parameter must be set to "`true`" if you want a user-provided service to be created for the specified module. In addition, you can specify custom parameters for the application (for example, `userID`, `password`, or `url`); these custom parameters are used for the creation of the user-provided service. The custom parameters are exposed to any other applications that bind to the user-provided service that is created during the application-deployment process.

> **!** Restriction
>
> The names of environment variables already defined at the platform level (for example, <XS_*>) cannot be used to define the name of a custom parameter, too. In addition, since "`user`" is the (protected) name of a platform environment variable, it cannot be used as the name of a custom parameter. To specify the name of a user in a custom parameter, use "`userID`", "`username`", or something similar instead.

Alternative Grouping of MTA Properties

The deploy service supports the extension of the standard syntax for references in module properties; this extension enables you to specify the name of the `requires` section inside the property reference. You can use this syntax extension to declare implicit groups, as illustrated in the following example:

Sample Code

Syntax Extension: Alternative Grouping of MTA Properties

```
modules:
  - name: pricing-ui
    type: javascript.nodejs
    properties:
      API: # equivalent to group, but defined in the module properties
       - key: internal1
         protocol: ~{price_opt/protocol} #reference to value of protocol
defined in price_opt of module pricing-backend
       - key: external
         url: ~{competitor_data/url} # reference to string value of property
'url' in required resource 'competitor_data'
         api_keys: ~{competitor_data/creds} # reference to list value of
property 'creds' in 'competitor_data'
    requires:
     - name: competitor_data
     - name: price_opt
  - name: pricing-backend
    type: java.tomcat
    provides:
     - name: price_opt
       properties:
          protocol: http ...
  resources:
    - name: competitor_data
      properties:
        url: "https://marketwatch.acme.com/"
        creds:
          app_key: 25892e17-80f6
          secret_key: cd171f7c-560d
```

MTA Deployment from a Directory

It is possible to deploy an MTA directly from directory, provided that directory structure is suitably defined. When deploying an MTA from a directory structure, the deploy service's command-line client (`xs deploy`) builds an MTA archive (`myMTA.mtar`) as a first step and then uses the prepared archive to complete the deployment operation. To ensure a successful deployment of an MTA from a directory structure, the following prerequisites apply:

- `mtad.yaml`
  The `mtad.yaml` file should be located in the root of the MTA directory one would like to deploy.
- `MANIFEST.MF`
  To enable the deploy service to create a valid `MANIFEST.MF` entry for each MTA element with the associated content, you must ensure that the conditions listed in the following table are met:

Required Entries in the MANIFEST.MF File for Directory Deployment

Element with Associated Content	Element/Parameter	Example mtad.yaml	Manifest Entry

MTA module	Element: `path`	`modules:`
		`- name: jobscheduler-backend`
		`  type: javascript.nodejs`
		`  path: backend/`
			`Name:`
			`  backend/`
			`MTA-Module:`
			`  jobscheduler-backend`

MTA resource that requires creation configuration specified in an external file	Parameter: `config-path`	`- name: jobscheduler-uaa`
		`  type: com.sap.xs.uaa`
		`  parameters:`
		`  config-path: security/xs-security.json`
			`Name:`
			`  security/xs-security.json`
			`MTA-Resource:`
			`  jobscheduler-uaa`
			`Content-Type:`
			`  application/json`

Module required dependency, pointing to the resource that needs binding configuration specified in an external file	Parameter: `config-path`	`modules:`
		`- name: backend`
		`  requires: db`
		`  parameters:`
		`  config-path: cfg/backend-db-params.json`
			`Name:`
			`  cfg/backend-db-params.json`
			`MTA-Requires:`
			`  backend/db`
			`Content-Type:`
			`  application/json`

> **i** Note
>
> The values specified for `path` and `config-path` values should be the **relative** path from the root directory of the MTA to be deployed to the file or directory containing the specified content.

Sizing Restrictions

The following table lists the restrictions that apply to the size of MTA **deployment** components that are handled by the SAP HANA deploy service during application deployment both on-premise (XS advanced model) and SAP Business Technology Platform (Cloud Foundry):

MTA Descriptor File Size Restrictions

MTA Component	Maximum Size
MTA archive	4 GB
MTA **module** content	4 GB
MTA **resource** content	1 GB
MTA deployment descriptor (`mtad.yaml`)	1 MB

MTA Component	Maximum Size
MTA manifest (`MANIFEST.MF`)	1 MB

Transporting XS Advanced Applications with CTS

You can enable SAP HANA extended application services, advanced model (XS advanced), to use the Change and Transport System (CTS) to transport applications that are deployed to the XS advanced run-time environment. The setup process includes the following high-level steps:

- Configure CTS
  This includes configuring transport destinations, setting up the XS advanced deploy service, and maintaining the trust relationship between the transport source and target systems, etc.
- Configure the transport-related application type
- Configure the transport landscape in XS advanced
  This includes configuring the transport systems in the transport management system, for example: development system (export), and the test and production systems (import)
- Using CTS with XS advanced
  This includes selecting and attaching objects to transport requests, importing transport requests with XS advanced content, etc.

For more information about the configuration steps required to set up CTS for XS advanced, see *Related Information* below.

Related Information

MTA Deployment Descriptor Syntax [page 173]
Multi-Target XS Advanced Applications [page 139]
Create the MTA Description Files [page 137]
How To Configure XS Advanced Model for CTS

4.1.3.1 MTA Deployment Descriptor Syntax

Description of an MTA's deployment-related prerequisites and dependencies.

> **i Note**
>
> The following syntax example contains links to more detailed information. Hover the mouse cursor over the property you want to find out about and click the link, if available, to display details of the required syntax.

Sample Code

MTA Deployment Description (`mtad.yaml`)

```
ID: com.sap.xs2.samples.nodehelloworld
version: 0.1.0
parameters:
  enable-parallel-deployments: true
modules:
  - name: node-hello-world
    type: javascript.nodejs
    path: web/   # Only for use in mta.yml
    deployed-after: [ node-hello-world-backend, node-hello-world-db]
    requires:
      - name: nodejs-uaa
      - name: nodejs
        group: destinations
        properties:
          name: backend
          url: ~{url}
          forwardAuthToken: true
        properties-metadata:
          name:
           optional: false
           overwritable: false
          url:
           overwritable: false
    parameters:
      host: !sensitive ${user}-node-hello-world
      memory: 128MB
    parameters-metadata:
      memory:
        optional: true
        overwritable: true
  - name: node-hello-world-backend
    type: javascript.nodejs
    path: js/   # Only for use in mta.yml
    provides:
      - name: nodejs
        properties:
          url: "${default-url}"
    requires:
      - name: nodejs-uaa
      - name: nodejs-hdi-container
      - name: node-hello-world-db
    parameters:
      host: ${user}-node-hello-world-backend
  - name: node-hello-world-db
    type: com.sap.xs.hdi
    path: db/   # Only for use in mta.yml
    requires:
      - name: nodejs-hdi-container
    parameters:
        tasks:
          - name: task-1
            command: node deploy.js
            env:
              env1: value1
resources:
  - name: nodejs-hdi-container
    type: com.sap.xs.hdi-container
    parameters:
      config:            # Only for use in mtad.yaml
        schema: ${default-container-name}
        xsappname: MyAppName01

  - name: nodejs-uaa
```

```
    type: com.sap.xs.uaa
    parameters:
      config-path: xs-security.json          # Only for use in mta.yaml
  - name: log
    type: application-logs
    optional: true
```

ID

Use the `ID` property to specify a unique identifier for the MTA. For example, the ID could be a reverse-URL dot-notation string that uniquely identifies the run-time namespace of the multi-target application to be deployed.

Code Syntax

```
ID: com.acme.mta.samples.javahelloworld
```

version

Use the `version` property to specify a version of the MTA defined in the `mtad.yaml` deployment descriptor.

The version value must follow the standard format for semantic versioning, for example, 2.0.0 (`<major>.<minor>.<patch>`). The restriction ensures a reliable contract between a development environment and the deployed system components.

Sample Code

```
"version": "<Major_Version>.<Minor_Version>.<Patch_Version>"
```

Note

It is also possible to declare a "partial" or incomplete version, for example, "`1`" or "`1.3`".

You can specify all the information used to define the MTA version, for example: "`<Major_Version>.<Minor_Version>.<Patch_Version>`", or any combination of the information, for example, `<Major_Version>.<Minor_Version>`, or simply `<Major_Version>`. If the version declaration does not specify a value for the `<Minor_Version>` or the `<Patch_Version>`, the deployer assumes the most recent known minor or patch version.

modules

`modules` section of the MTA deployment descriptor, you can define a set of modules of a certain type, the contents of which reside in the MTA archive or in a file system.

Sample Code

Module Definition in the MTA Deployment Description

```
modules:
  - name: node-hello-world
    type: javascript.nodejs
    path: web/   # Only for use in mta.yml
    requires:
      - name: nodejs-uaa
      - name: nodejs
        group: destinations
        properties:
          name: backend
          url: ~{url}
          forwardAuthToken: true
    parameters:
      host: ${user}-node-hello-world
      tasks:
        - name: task-1
          command: node deploy.js
          env:
            env1: <value1>
            env1: <value2>
        - name: task-2
```

→ Tip

It is also possible to declare metadata for parameters and properties defined in the MTA deployment description; the mapping is based on the parameter or property keys. For example, you can specify if a parameter is **required** (`optional; false`) or can be modified `overwritable: true`.

Use the following attributes to define each new dependent module:

MTA Deployment Descriptor Modules Keys

Attribute	Description	Example Value
`name`	Name of the module to be deployed	java-hello-world-backend
`type`	Determines the run-time to which the specified module is deployed.	java nodejs python

Attribute	Description	Example Value
`path`	The file-system path relative to the root of the MTA directory tree. The content of the file is used for the creation (or update) of the XS advanced multitarget application **! Restriction** The specified `path` is used only during the MTA build, and used to create an entry in the `MANIFEST.MF` file. In existing (already built) MTA archives (MTAR), `path` is ignored, and the corresponding entry for the module in the `MANIFEST.MF` file is used. For the deployment descriptor, `mtad.yaml`, see `config-path` in parameters.	`web/`

`group`	Combine properties from multiple providers into one look-up object (for example, an environment variable). This reduces the number of look ups performed by the code of the requiring module.	`DESTINATIONS`
		
		`API`

`provides`	Used to specify the names of `provides` sections, each containing configuration data; the data **provided** can be **required** by other modules in the same MTA.	`<provides_section_name>`
`requires`	Used to specify the names of "`requires`" sections that are provided"resource" that has been declared for the same MTA. Tools check if all required names are provided within the MTA. The deployment mechanism for the requiring module will access properties of the matching `provides`Metadata for MTA Deployment Parameters and section and provide them to the module runtime, for example, in environment variables. The `requires` section can declare a **group** assignment. Groups are used to combine properties from multiple providers into one lookup object (for example, an environment variable). This reduces the number of lookups performed by the code of the requiring module.	`<requires_section_name>`

Attribute	Description	Example Value

`properties`	A structured set of name-value pairs, which contain values that are injected into the environment of requiring modules at run time.
	
	A `properties` property section has a map structure at its first level; any deeper levels can be structured in any way, as long as they can be transformed into a valid JSON object or array.
	
	Keys defined at the first level map are used as lookup names. If environment variables are used, one variable is created per map key, and the name of the variable equals the key value.
	
	→ Tip
	It is possible to declare metadata for parameters and properties defined in the MTA deployment description; the mapping is based on the parameter or property keys. For example, you can specify if a parameter is **required** (`optional; false`) or can be modified `overwritable: true`.
		`properties:`
		`  name: backend`
		`  url: ${default-url}`
		`  forwardAuthToken: true`

`parameters`	Reserved variables that affect the behavior of the MTA-aware tools, such as the deployer. Parameters can be “read-only” values, “write-only”, or “read-write” (current/defined value can be overwritten). Modules and resources can read parameters by referencing them with place-holder notation, for example, enclosed by `${}`.	`host: ${user}-node-hello-world`

> ### i Note
> In the context of the module `type`, the “deployment environment” is any required set of configuration tools, deployment orchestration tools, and target-platform-specific services used to deploy an MTA.

The following table shows how the modules `type` defined in the MTA deployment descriptor is mapped, where appropriate, to a corresponding build pack and which, if any parameters and properties can be used.

> ### ! Restriction
> In the following table, “platform” is either XS (XS advanced) or CF (Cloud Foundry). For details of individual parameters, see MTA Development and Deployment Parameters [page 194] below.

MTA Default Modules Types (XS Advanced/Cloud Foundry)

Module Type	Platform	Module Parameters (default)	Module Properties	Result
`javascript.nodejs`	XS	None	None	Node.js run time
	CF			Automatic build-pack detection
`nodejs`	CF	`buildpack(nodejs_buildpack)`	None	Node.js run time
`native`	XS	None	None	Native C++ run time
`custom`	XS	None	None	Custom run time
	CF			Automatic build-pack detection
`application`	CF	CF	None	Automatic build-pack detection
`java`	XS	None	None	Java run time **i Note** Default is Tomcat
	CF	`buildpack(sap_java_buildpack)`		
`java.tomcat`	XS	None	`TARGET_RUNTIME` (tomcat)	Tomcat run time of `sap_java_buildpack`
	CF	`buildpack(sap_java_buildpack)`		
`java.tomee`	XS	None	`TARGET_RUNTIME` (tomee)	TomEE run time of `sap_java_buildpack` **i Note** In July 2021, we plan to remove the

Module Type	Platform	Module Parameters (default)	Module Properties	Result
	CF	`buildpack(sap_java_buildpack)`		outdated TomEE 1.7 version from the SAP Java Buildpack for SAP HANA XS Advanced. Although TomEE 1.7 will continue to be supported until the end of the deprecation period, it is recommended to migrate to TomEE 7 as soon as possible, as described in the *TomEE Migration Guide* listed in *Related Information* below. TomEE 7 is already included in previous versions of the SAP Java Buildpack for SAP HANA XS Advanced.

Module Type	Platform	Module Parameters (default)	Module Properties	Result
`python`	XS	None	None	Python run time
	CF	`buildpack(python_buildpack)`		
`com.sap.xs.hdi`	XS/CF	• `no-route: true` • `no-start: true` • `memory: 256MB` • `tasks: true` In this case, the one-off task executed on the application (after it has been deployed) deploys the entire HDI content. `[{ "name": "deploy", "command": "npm start", "env": { "EXIT": 1 } }]` • `dependency-type: hard`	None	HDI content activation
`com.sap.xs.hdi-dynamic`	XS/CF	`buildpack(nodejs_buildpack)`	None	Node.js run time
`com.sap.xs.hdi-zdm`	XS/CF	• `zdm-mode: true` • `no-route: true` • `memory: 256MB` • `execute-app: true` • `success-marker: STDOUT:.*ZDM deployment done.*` • `failure-marker: STDERR:.*ZDM deployment failed.*` • `stop-app: true` • `check-deploy-id: true` • `dependency-type: hard` • `health-check-type: none`	`"HDI_DEPLOY_MODE": "ZDM"`	HDI content activation
`com.sap.xs.sds`	XS/CF	• `no-route: true` • `dependency-type: hard`	None	Streaming Analytics

Module Type	Platform	Module Parameters (default)	Module Properties	Result
`com.sap.xs.dwf`	XS/CF	• `no-route: true` • `memory: 256M` • `execute-app: true` • `success-marker: STDOUT:The deployment of DWF content to .* container container finished successfully .*` • `failure-marker: STDERR:The deployment of DWF content to .* container failed .*` • `stop-app: true` • `check-deploy-id: true` • `dependency-type: hard`	None	Data Warehousing Foundation
`com.sap.portal.site-content`	XS/CF	• `no-route: true` • `memory: 256M` • `execute-app: true` • `success-marker: "STDOUT:The deployment of html5 application content done.*"` • `failure-marker: "STDERR:The deployment of html5 application content failed.*"` • `check-deploy-id: true` • `dependency-type: hard` • `health-check-type: none`	None	Portal content activation

Module Type	Platform	Module Parameters (default)	Module Properties	Result
`com.sap.html5.application-content`	CF	• `no-route: true` • `memory: 256M` • `execute-app: true` • `success-marker: "STDOUT:The deployment of html5 application content done.*"` • `failure-marker: "STDERR:The deployment of html5 application content failed.*"` • `stop-app: true` • `check-deploy-id: true` • `dependency-type: hard` • `health-check-type: none`	None	HTML5 application content activation **! Restriction** For use only in SAP Cloud Platform (Cloud Foundry) scenarios, to make the Node.js Application Router available as a service, for example, to applications that have no graphical front end.
`business-logging`	CF	• `no-route: true` • `memory: 256M` • `execute-app: true` • `success-marker: "STDOUT:Deployment of content deployer done."` • `failure-marker: "STDERR:The deployment of content deployer failed."` • `stop-app: true` • `health-check-type: none`	None	Deploys the Business Logging for configuring text resources
`staticfile`	CF	`buildpack(staticfile_buildpack)`	None	Static file run time
`ruby`	CF	`buildpack(ruby_buildpack)`	None	Ruby run time
`go`	CF	`buildpack(go_buildpack)`	None	Go run time

Module Type	Platform	Module Parameters (default)	Module Properties	Result
`php`	CF	`buildpack(php_buildpack)`	None	PHP run time
`binary`	CF	`buildpack(binary_buildpack)`	None	Binary run time
`dotnet_core`	CF	`buildpack(dotnet_core_buildpack)`	None	Dotnet run time

You can use the `buildpack` parameter to specify a particular build pack to use for the application build and deployment, as illustrated in the following example:

Sample Code

```
modules:
  - name: my-binary-app
    type: custom
    parameters:
      buildpack: binary_buildpack
```

resources

A resource is something which is required by the MTA at run time but not provided by the MTA, for example, a managed service instance, an instance of a user-provided service, or an external Web resource.

Sample Code

Resource Definition in the MTA Deployment Description

```
resources:
  - name: nodejs-hdi-container
    type: com.sap.xs.hdi-container
    parameters:
      config:               # Only for use in mtad.yaml
        schema: ${default-container-name}
        xsappname: MyAppName01
  - name: nodejs-uaa
    type: com.sap.xs.uaa
    parameters:
      config-path: security/xs-security.json    # Only for use in mta.yaml
  - name: log
    type: application-logs
    optional: true
```

In the `resources` section of the MTA deployment descriptor, the following attributes are used to define each resource:

MTA Deployment Descriptor Resources Keys

Attributes	Description	Example Value
`name`	The name of the required resource	`<Module_Name>`
`type`	Determines the way in which the resource is provisioned by the deploy service. In XS advanced, typed resources are provisioned as service instances. **i Note** The resource type is mapped to a service and a corresponding service plan. For more information, see "parameters" below and service mappings.	com.sap.xs.hana-sbss com.sap.xs.hdi-container com.sap.xs.managed-hdi-container com.sap.xs.uaa com.sap.xs.fs com.sap.portal.site-content org.cloudfoundry.managed-service *
`description`	Non-translatable, free-text string; the string is not meant to be presented on application user interfaces (UI)	Resource provides features that ...
`parameters`	Reserved variables that affect the behavior of the MTA-aware tools, such as the deployer. Parameters can be "read-only" values, "write-only", or "read-write" (current/defined value can be overwritten). Modules and resources can read parameters by referencing them with place-holder notation, that is; enclosed by `${}`.	`service-plan: hdi-shared`

`properties`	A structured set of name-value pairs, which contain values that are injected into the environment of requiring modules at run time.	`properties:`
		`  name: backend`
		`  url: ~{url}`
		`  forwardAuthToken: true`

`optional`	An application can declare required resources to be `optional`, if it can compensate for their non-existence. If a required resource is declared as `optional: true` and the deployer cannot allocate the required resource, the deployer issues a warning and continues processing. If a required resource is **not** optional (`optional: false`), the deployer indicates an error and stops processing.	`optional: true` (default = `false`)

Attributes	Description	Example Value
`active`	Determines whether the resource should be considered (`true`) or ignored (`false`) for provisioning or for service binding. Resources that are not active (`"false"`) are not processed and are ignored even if the "inactive" resource is present in another application's `requires` list. If you previously deployed the same application with the resource type attribute `active` set to `true`, the binding to the resource will be removed if `active` is reset to `false`. If the resource is set to `active: false`, no binding is created for deployments with the resource type `existing-service`, or `user-provided-service`; no binding **or** service is created for `managed-service`. For deployments with the resource type `org.cloudfoundry.existing-service-key`, no binding to an application environment is made in the following cases: • If the `org.cloudfoundry.existing-service-key` resource itself is set to `active: false` • If the `org.cloudfoundry.existing-service-key` resource is set to `active: true`, but refers to a resource that is set to `active: false` For deployments with configuration resources that are dependent on other Multi-Target Applications and	`active: false` (default = `true`)

Attributes	Description	Example Value
	the configuration resource is set to `active:false`: • If the `requires` section of the module **type** expects a list, then the environment variable assigned for this list is empty and no subscriptions are created between the modules. • If the `requires` section does **not** expect a list, then there will be no environment variable.	

> **i** Note
>
> The resource type is mapped to a service and a corresponding service plan.

For example, the resource type "`com.sap.xs.hana-sbss`" is mapped to the "`hana`" service plan "`sbss`"; the resource type "`com.sap.xs.uaa-devuser`" is mapped to the "`xsuaa`" service plan "`devuser`". The command `xs marketplace` displays a list of all currently available services and service plans.

The following table shows how the **resource type** defined in the MTA deployment descriptor is mapped to a corresponding service and service plan. Bear in mind that the availability of features and functionality in the Cloud environment can change at short notice. For information about the latest changes and additions to the support for Cloud Foundry applications in the SAP Cloud Platform, see *MTA Resource Types in the SAP CP Cloud Foundry Environment* in *Related Information* below.

> **!** Restriction
>
> In the following table, "platform" is either XS (XS advanced) or CF (Cloud Foundry).

MTA Default Resource Types and Mapped Services

Resource Type	Service	Platform	Service Plan	Created Service
com.sap.xs.hana-sbss	hana	XS/CF	sbss	Service-broker security
com.sap.xs.hana-schema	hana	XS/CF	schema	Plain schema
com.sap.xs.hana-securestore	hana	XS/CF	securestore	SAP HANA secure store
com.sap.xs.hdi-container	hana	XS/CF	hdi-shared	HDI container
com.sap.xs.managed-hdi-container	managed-hana	XS	hdi-shared	Managed HDI container

Resource Type	Service	Platform	Service Plan	Created Service
com.sap.xs.managed-hana-schema	managed-hana	XS	schema	Managed plain schema
com.sap.xs.managed-hana-secure-store	managed-hana	XS	securestore	Managed HANA secure store
com.sap.xs.job-scheduler	jobscheduler	XS	default	Job Scheduler
		CF	lite	
com.sap.xs.fs	fs-storage	XS	free	File-system storage
		CF	lite	
com.sap.xs.uaa	xsuaa	XS/CF	default	Global UAA *
com.sap.xs.uaa-application	xsuaa	CF	application	PaaS-enabled application UAA service plan for CF scenarios
com.sap.xs.uaa-broker	xsuaa	CF	broker	PaaS-enabled broker UAA service for CF scenarios
com.sap.xs.uaa-devuser	xsuaa	XS/CF	devuser	Development-user UAA service
com.sap.xs.uaa-space	xsuaa	XS/CF	space	UAA service for a space
com.sap.xs.uaa-system	xsuaa	CF	system	PaaS-enabled system UAA service for CF scenarios
com.sap.xs.sds	sds	XS/CF	default	Streaming Analytics
com.sap.xs.auditlog	auditlog	XS	free	Audit-log service
auditlog		CF	standard	
com.sap.xs.ui5	sapui5_sb	XS	sapui5-1.52	SAPUI5 1.0.1
com.sap.xs.messaging	messaging-service	XS	free	Messaging Service

Resource Type	Service	Platform	Service Plan	Created Service
com.sap.portal.site-content **! Restriction** Only for use with the SAP Node.js module `@sap/site-content-deployer`	portal-services	XS/CF	site-content	Portal services
com.sap.portal.site-host **! Restriction** Only for use with the SAP Node.js module `@sap/site-entry`	portal-services	XS/CF	site-host	Portal services
application-logs **i Note** Replaces deprecated `com.sap.xs.application-logs`	application-logs	CF	lite	Streams logs of bound applications to a central application logging stack
autoscaler	autoscaler	CF	lite	Automatically increase or decrease the number of application instances based on a policy you define.
connectivity	connectivity	CF	lite	Establishes secure, reliable connectivity between Cloud applications and on-premise systems
destination	destination	CF	lite	Provides a secure and a reliable access to destination configurations
feature-flags	feature-flags	CF	lite	Feature Flags service for the control of feature rollout

Resource Type	Service	Platform	Service Plan	Created Service
ml-foundation-services	ml-foundation-services	CF	lite	Provides access to foundation services by provisioning and deprovisioning“ ml foundation” services
mongodb **Note** Replaces deprecated `org.mongodb`	mongodb	CF	v3.0-container	MongoDB document-oriented database system
objectstore	objectstore	CF	s3-standard	Provides a highly available, distributed, consistent object store
postgresql **Note** Replaces deprecated `org.postresql`	postgresql	CF	v9.4-container	Provides a postgreSQL object-relational database system
rabbitmq **Note** Replaces deprecated `com.rabbitmq`	rabbitmq	CF	v3.6-container	Provides RabbitMQ messaging
redis **Note** Replaces deprecated `io.redis`	redis	CF	v3.0-container	Redis in-memory data structure store

You can use the `service-plan` parameter to specify a custom service plan in place of the predefined plan. The following example shows how to replace the default service plan `v9.4-container` for the `postgresql` resource with the custom service plan `v9.4-container-xsmall`:

Sample Code

Specifying a Custom Service Plan

```
resources:
  - name: my-postgre-service
    type: postgresql
```

```
    parameters:
      service-plan: v9.6-dev
```

To use a managed service (or service plan) in a Cloud Foundry scenario which is not listed in the resource-types tables above (and the special resource-types table below), you can define the new managed service with the resource type `org.cloudfoundry.managed-service`, as illustrated in the following example:

Sample Code

```
resources:
  - name: my-postgre-service
    type: org.cloudfoundry.managed-service
    parameters:
      service: spark
      service-plan: shared
```

The `shared` parameter can only be used with resource types that are interpreted as services, for example, all `org.cloudfoundry.[managed|user-provided]-service` resource types as well as all default resource types such as `com.sap.xs.hdi-container`.

The deployment service also supports a number of "special" resource types, as illustrated in the following table:

Additional Special MTA Resource Types: Service Mapping

Resource Type	Resource Parameter ID	Created Service
org.cloudfoundry.user-provided-service	• `service-name` **Optional**. Name of the service to create. Default value: the resource name. • `config` **Required**. Map value, containing the service creation configuration, for example, `url` and user credentials (`user` and `password`) • `shared` **Optional** (true | false (default)). Share service between multiple MTAs.	Create or update a user-provided service configured with the specified resource parameters
org.cloudfoundry.managed-service	• `service` **Required**. Name of the service to create. • `service-plan` **Required**. Name of the service plan to use; this can be any service plan available in the service market place (`xs marketplace`). • `shared` **Optional** (true | false (default)). Share service between multiple MTAs.	Create a managed service
org.cloudfoundry.existing-service	• `service-name` **Optional**. Name of the service to ignore. Default value: the resource name.	Assume that the named service exists and do not try to manage its life-cycle

Resource Type	Resource Parameter ID	Created Service
org.cloudfoundry.existing-service-key	• `service-name` **Required**. Name of the existing service instance whose key is to be used. • `service-key-name` **Optional**. Name of the service key. Default value is the resource `name`.	Check and use credentials for an existing service key.
mta-provided	• `mta-id` **Required**. The ID of the provider MTA. Can also contain version ranges as described in the Semantic Version (Semver) specification. • `mta-version` **Required**. The version of the provider MTA. • `mta-provides-dependency` **Required**. The name of the `provides` dependency in the provider MTA.	Define dependencies between different MTAs

Resource Type	Resource Parameter ID	Created Service

configuration	• `provider-nid`
	Optional. Name space ID of the provider MTA. For cross-MTA dependency resolution this value is always "`mta`"
	• `provider-id`
	Optional. The ID of the provider MTA. Requires the format `<mta-id>:<mta-provides-dependency-name>`.
	• `version`
	Optional. Version number of the provider MTA; required format is defined in the Semver specification.
	• `target`
	Optional. Name of the provider MTA's target "space"; a structured parameter that must contain the "`org`" and "`space`" of the provider MTA.
	→ Tip
	If you specify a wild-card value (*) for organization or space of the provider MTA, the provider would be searched in all organization or spaces for which the wild-card value is provided. If no match is found for the provider MTA in the specified target organization or space, then `org: <current-org>` and `space: SAP` is searched.
	• `filter`
	Optional. Map value, containing the content entries used to match configuration entries, for example:
	`filter:`
	`  type: com.acme.plugin`
	`  name: some-cool-plugin`
		Define the mechanism for filtering configuration entries based on their configuration content

Parameters

Module, resource, and dependency parameters have special, platform-specific semantics. In general, many of these parameters are the same and have the same syntax and semantics as those that can be specified in manifest files for XS advanced application-deployment, for example, with the `xs push` command. All parameter values can be referenced as part of other property or parameter value strings. To reference a parameter value, use the placeholder notation `${<parameter>}`, for example, `${default-host}`.

> → Tip
>
> It is also possible to declare metadata for parameters and properties defined in the MTA deployment description; the mapping is based on the parameter or property keys. For example, you can specify if a parameter is **required** (`optional; false`) or can be modified `overwritable: true`.

The following parameters are supported:

MTA Development and Deployment Parameters

Parameter	Scope	Read-Only (System)	Default Value	Example
`app-name`	modules	Yes	The name of the application in the XS advanced or Cloud Foundry environment to be deployed for this module, based on the module name (with or without a name-space prefix)	`node-hello-world` `com.sap.xs2.samples.xsjshelloworld.node-hello-world`
`authorization-url`	All	Yes	The authorization end point for the XS advanced or Cloud Foundry environment Default: N/A → Tip Use in place of the deprecated `xs-auth-url`.	`https://localhost:9999/uaa-security`
`buildpack`	modules		Empty, or as specified in the deploy service configuration	`buildpack: git://github.acme.com/xs2-java/xs2javabuildpack`
`check-deploy-id`	modules		Check the deployment (process) id when checking the application execution status	`check-deploy-id: true`
`command`	modules		Empty, or as specified in the deploy service configuration	`command: node index.js`

Parameter	Scope	Read-Only (System)	Default Value	Example

`config`	resources		Define service creation parameters in the MTA deployment descriptor **inline**. If the option is present in both the `MANIFEST.MF` (`"xsappname": "default-xsappname"`) and the `config` parameter (`xsappname: MyAppName01`), the MTA deployment service merges both definitions and gives precedence to the `config` parameter (`MyAppName01`).
			! Restriction
			For use in the **deployment** descriptor (`mtad.yaml`) only!
				`resources:`
				`  - name: nodejs-hdi-container`
				`type: com.sap.xs.hdi-container`
				`    parameters:`
				`      config:`
				`# Only for use in mtad.yaml`
				`        xsappname: MyAppName01`

`config-path`	resources		The relative path from the MTA root directory to a file in the same application project, which contains information used to create or update parameters of an XS advanced service represented by the MTA resource.
			! Restriction
			The specified `config-path` is used only during the MTA build, and a corresponding entry is created in the `MANIFEST.MF` file. In existing (already built) MTA archives (MTAR), the path is ignored, and the entry for the module in the `MANIFEST.MF` file is used instead.
				`resources:`
				`  - name: nodejs-uaa`
				`    type: com.sap.xs.uaa`
				`    parameters:`
				`      config-path: ./xs-security.json`
				`# Only for use in mta.yaml`

Parameter	Scope	Read-Only (System)	Default Value	Example
`controller-url`	All	Yes	The URL for the application programming interface (API) in the XS advanced or Cloud Foundry environment Default: N/A → Tip Use in place of the deprecated `xs-api-url`.	CF: `https://api.cf.acme.ondemand.com` XS: `http://localhost:30030`

`database_id`	resources	Yes	The unique ID of a database instance (other than the default) that hosts the run-time to which you want to deploy an application.
			→ Tip
			Also useful in Cloud scenarios where multiple tenant databases are mapped to the same Cloud Foundry organization or space and you need to specify the target tenant DB explicitly.
				`resources:`
				`  - name: hdi_db`
				`    parameters:`
				`      config:`
				`        database_id: 007c3a2f-5f4a-42e2-bd18-c00d7ec007f`

`default-container-name`	resources	Yes	Default value for the container-name parameter that is used during HDI creation. It is based on the organization, space and service name, which are combined in a way that conforms to the container-name restrictions for length and legal characters. Default: N/A	`INITIAL_INITIAL_SERVICE_NAME`
`default-domain`	All	Yes	The default domain (configured in the XS advanced or Cloud Foundry environment)	`accra6024` `cfapps.acme.com`

Parameter	Scope	Read-Only (System)	Default Value	Example
`default-host`	modules	Yes	The default host name, composed based on the target platform name and the module name, which ensures uniqueness. Used with host-based routing to compose the default URI, see below.	`trial-a007007-node-hello-world`
`default-port`	modules		The default, automatically allocated port value. Used with port-based routing to compose the default URI (explained below). If a previous version of the application is already deployed, the default port is the same as the port in the first URL used by that application. If a previous version has not been deployed, a new port is reserved that is not in use by any other application	52001
`default-uri`	modules	Yes	The default URI, composed either as `${domain}:${port}` with port-based routing or as `${host}.${domain}` with host-based routing. Note that `${host}` will be the same as `${default-host}`, unless specified explicitly as a parameter. Similarly, `${domain}` will be the same as `${default-domain}`, and `${port}` will be the same as `${default-port}`, unless specified explicitly.	XS advanced: `accra6024:52001` Cloud Foundry: `trial-a007007-node-hello-world.cfapps.acme.ondemand.com`
`default-url`	modules	Yes	The default URL, composed as `${protocol}://${default-uri}`	`${protocol}://${default-uri}`

Parameter	Scope	Read-Only (System)	Default Value	Example
`default-xsappname`	resources	Yes	Default value for the `xsappname` parameter that is used during UAA creation. It is based on the service name, which is modified in a way that conforms to the `xsappname` restrictions for length and legal characters. Default: N/A	`xs_-deploy-service-database` (if the service name is "`xs@-deploy-service-database`"
`dependency-type`	modules		Deployment order of modules with circular dependencies. Default: "`soft`"	`dependency-type: hard` `dependency-type: soft`
`deployed-after`	modules		Indicates the point in the deployment sequence when the specified modules must be deployed	`deployed-after: [ backend, metrics ]`
`deploy-url`	All	Yes	The URL of the deploy service for the XS advanced or Cloud Foundry environment	`https://mo-de0825faa.mo.sap.corp:51002`
`disk-quota`	modules		The disk space that will be available to the application. **i Note** Requires a unit of measurement (M, MB, G, or GB) in upper or lower case. XS Advanced: -1 (unlimited), or as specified in module-type. Cloud Foundry: 1GB, or as specified in module-type.	`disk-quota: 1G`
`domain`	modules		`${default-domain}` **→ Tip** It is recommended to use `routes` to define application routes.	`domain: ${default-domain}.acme.com`

Parameter	Scope	Read-Only (System)	Default Value	Example
`domains`	modules		`domains: - ${default}`	`domains: - ${default-domain}.acme.com - test-${default-domain}.acme.com`
`enable-ssh`	modules		Enables (`true`) the use of SSH within an application. **! Restriction** For the Diego container run-time environment only.	`enable-ssh: true`
`enable-parallel-deployments`	All		Activates `true`) the parallel deployment of MTA modules that do not have any dependencies regarding the deployment order. If the parameter is missing or its value is `false`, the modules are deployed sequentially.	`enable-parallel-deployments: true`
`execute-app`	modules		After start and upon completion, application sets [success | failure]-marker in a log message	`execute-app: true`
`env-var-name`	modules		Specified in required dependencies and defines the name of the environment variable that will contain the service key credentials. The parameter itself is optional Default: the name of the service key.	`modules: - name: foo type: javascript.nodejs requires: - name: service-key-1 parameters: env-var-name: SERVICE_KEY_CREDENTIALS`
`failure-marker`	modules		The failure marker in the log files for application execution, for example: `(STDERR): The deployment of <DWF> content to .* container failed.*` Default: `"STDOUT:FAILURE"`	`failure-marker: "STDERR:Failure!"`

Parameter	Scope	Read-Only (System)	Default Value	Example
`generated-password`	All	Yes	A generated user id that is composed of 16 characters that may contain upper and lower case letters, digits and special characters (_, -, @, $, &, #, *). Default: N/A	`IG@zGg#2g-cvMvsW`
`generated-user`	All	Yes	A generated user id that is composed of 16 characters that may contain upper and lower case letters, digits and special characters (_, -, @, $, &, #, *). Default: N/A	`uYi$d41TzM1-Dm6f`
`health-check-http-endpoint`	modules		Application is considered healthy if the response to `GET` request is "200 OK". Default: • `health-check-type: http`: "/" • Otherwise: N/A	`health-check-type: http` `health-check-http-endpoint: /health`
`health-check-timeout`	modules		Health-check timeout in seconds	`health-check-timeout: 120`

`health-check-type`	modules		`.http`	`health-check-type: port`
				`health-check-type: http`
				`health-check-type: process`

`host`	modules		`${default-host}` → Tip It is recommended to use `routes` to define application routes.	`host: ${space}-node-hello-world`
`hosts`	modules		`hosts: - ${host}`	`hosts: - ${space}-node-hello-world` `- test-${space}-node-hello-world`

Parameter	Scope	Read-Only (System)	Default Value	Example
`instances`	modules		1, or as specified in module-type	`instances: 2`

`keep-existing-routes`	Global, modules		At the **global** level, it ensures that the existing routes of **all** applications within an MTA are retained. At the **modules** level, it ensures the existing routes of the module's corresponding application are retained, even if the routes are not defined within the deployment and/or extension descriptors.
			
			i Note
			The module-level variant of the parameter takes priority over the global parameter. Default is `false`.
			
			`keep-existing-routes` is typically used to retain manually mapped routes (for example, with the "`[xs|cf] map-route`" command). However, manually mapped routes can lead to inconsistent deployment results or errors that are hard to trace. The recommended approach is to define all routes in the deployment and/or extension descriptors and let the deployer create and manage them.
				`parameters:`
				`  keep-existing-routes: true`
				`modules:`
				`  - name: foo`
				`  type: nodejs`
				`  parameters:`
				`  keep-existing-routes: false`
				`  - name: bar`
				`  type: nodejs`
				`  - name: baz`
				`  type: javascript.nodejs`

`memory`	modules		256M, or as specified in module-type	`memory: 128M`
`no-route`	modules		Do not assign a route to the application	`no-route: true`

Parameter	Scope	Read-Only (System)	Default Value	Example

`no-start`	modules		Start/do not start the application during deployment. Default: `false`.
			
			→ Tip
			This parameter setting overrides the command-line option --no-start.
			
			If you explicitly set the `no-start` to `false` for the module `foo` in the example provided, then the module `foo` **is** started on deployment, even if you also specify the command-line option `--no-start` with the `[xs | cf] deploy` command.
				`modules:`
				`  - name: foo`
				`  type: foo`
				`  parameters:`
				`  memory: 1G`
				`  no-start: true`
				`  - name: bar`
				`  type: bar`
				`  parameters:`
				`  memory: 2G`

`org`	All	Yes	Name of the target organization	`initial, trial`
`platform`	All	Yes	Name of the target platform. If not specified explicitly, a target platform is created implicitly as "`<org> <space>`". **! Restriction** For XS advanced on-premise only: not for use with Cloud Foundry.	`XSA-INITIAL`,"`initial initial`", "`trial a007007`"
`port`	modules		`${default-port}`	`port: 52001`
`ports`	modules		`ports: - ${port}`	`ports: - 52001 - 52002`
`protocol`	modules	Yes	The protocol used by the XS advanced or Cloud Foundry environment, for example: "http" or "https"	`https`

Parameter	Scope	Read-Only (System)	Default Value	Example

`restart-on-env-change`	modules		Restart application on update if `env` has changed in one of the environment variables: `vcap-application`, `vcap-services` and `user-provided`.
			
			i Note
			Default is `true`. `false` means that running applications cannot consume any environment changes.
				`restart-on-env-change:`
				`  vcap-application: false`
				`  vcap-services: true`
				`  user-provided: true`

`route-path`	modules	Yes	The context "route-path" which is part of the application default URI. Context path routing is based not only on domain names (host header) but also the path specified in the URL
			
			→ Tip
			It is recommended to use `routes` to define application routes.
				Host-based routing:
				`${host}.${domain}:<router-port>${route-path}`
				Port-based routing:
				`${domain}:${port}${route-path}`

`routes`	modules	Yes	An aggregation of the parameters `host(s)`, `domain(s)`, and `no-hostname`, which encompasses the full addresses to which to bind a module.
			
			→ Tip
			If both the `routes` parameter and the other (host, domain) parameters are defined, the `routes` value is used and the values specified for the other parameters are ignored.
				`modules:`
				`- name: my-app`
				`  type: application`
				`  parameters:`
				`  routes:`
				`  - route: host.my.domain/eg`
				`  - route: host.his.domain/eg2`
				`  - route: host.her.domain2/eg3`

`service`	resources		Empty, or as specified in resource-type	`service: hana`

Parameter	Scope	Read-Only (System)	Default Value	Example

`service-alternatives`	resources		Empty, or as specified in the deploy service configuration (`resource-type`)
			List of alternatives for a **default** service offering defined in the deploy service configuration. If a default service offering does not exist for the current org/space or creating a service fails (with a specific error), service alternatives are used. The order of service alternatives is considered.
				`parameters:`
				`  service: managed-hana`
				`  service-plan: hdi-shared`
				`  service-alternatives: ["hana"]`

`create-service-broker`	modules	Yes	Specifies whether [true|false] a service broker should be registered for the application module. Default: false	`create-service-broker: true`
`service-broker-name`	modules	Yes	The name of the service broker in the XS advanced or Cloud Foundry environment to be created and registered for the specified application module. Default: `${app-name}`	`service-broker-name: jobscheduler` `service-broker-name: ${app-name}`
`service-broker-password`	modules	Yes	The password used for authentication by the XS controller at the service broker when performing service-related requests. The parameter is mandatory if `create-service-broker: true`.	`service-broker-password: ${generated-password}`
`service-broker-url`	modules	Yes	Specifies the value of the service broker universal resource locator (URL) to register; service requests are sent to this URL. The parameter is mandatory if `create-service-broker: true`.	`service-broker-url: ${default-url}`

Parameter	Scope	Read-Only (System)	Default Value	Example
`service-broker-user`	modules	Yes	The name of the user required for authentication by the XS controller at the service broker when performing service-related requests. The parameter is mandatory if `create-service-broker: true`.	`service-broker-user: ${generated-user}`

`service-key-name`	resources		For resource type `org.cloudfoundry.existing-service-key`. Resource parameter `service-name` is mandatory, if no `service-key-name` is defined, the deploy service uses the name of the resource.	`parameters:`
				`  service-name: myservice-a`
				`  service-key-name: myservice-key`

`service-keys`	resources		The name of the service key to be created or updated.	`parameters:`
				`  service-keys:`
				`  - name: tool-access-key`
				`  config:`
				`  permissions: read-write`
				`  - name: reader-endpoint`
				`  config:`
				`  permissions: read-only`

`service-name`	resources	Yes	The name of the service to be created for this resource in the XS advanced or Cloud Foundry environment, based on the resource name (with or without a name-space prefix)	`nodejs-hdi-container` `com.sap.xs2.samples.xsjshelloworld.nodejs-hdi-container`
`service-plan`	resources		Empty, or as specified in resource-type	`service-plan: hdi-shared`
`service-tags`	resources		A list of custom service tags that are added to the `<VCAP_SERVICES>` environment variable and assigned to the specified service instance	`parameters:` `  service-tags: ["custom-tag-A", "custom-tag-B"]`

Parameter	Scope	Read-Only (System)	Default Value	Example
`siteId`	resources		A globally unique ID (GUID) for your Fiori LaunchPad site	`siteId=4c736e0c-a096-45f1-9ae5-a613eb24b2b9`
`space`	All	Yes	Name of the target organizational space	`initial, a007007`
`stop-app`	modules		Stop the application after execution.	`stop-app: true`
`success-marker`	modules		The success marker in logfiles for application execution, for example: `(STDOUT): The deployment of <appname> content done` Default: `"STDOUT:SUCCESS"`	`success-marker: "STDOUT:Success!"`

`tasks`	modules	Yes	Specify tasks which are available for execution (by `command:`) in the current droplet of the application. The tasks are executed after the deployment of the application in which they are specified.
			→ Tip
			Use `env:` to define environment variables which will be available during the task execution.
				`tasks:`
				`- name: task-1`
				`command: some-script.sh`
				`env:`
				`env1: value1`
				`env2: value2`

`tcp`	modules		Use TCP type routes for all application routes if there is `domains` parameters with multiple values. Default: `false`	`tcp:true`

Parameter	Scope	Read-Only (System)	Default Value	Example
`tcps`	modules		Create a TCPS route (and use TLS with SSL termination at the platform router) for all app routes where there are '`hosts`' and `domains` parameters with multiple values. **i Note** Only relevant if SSL is enabled globally in XS advanced, for example, with "SSL mode"). Default: `false`	`tcps:true`
`user`	All	Yes	Name of the current user	

`visibility`	modules		Specify in an "allowlist" the organizations and spaces from which configuration data can be consumed. An allowlist is not required for all the provider's spaces in its own organization. A consumer must explicitly declare in which organization or space a "provider" is expected to be deployed (if it is not in the consumer's space).	`provides:`
				`  - name: backend`
				`    parameters:`
				`      visibility:`
				`        - org: ${org}`
				`          space: $`
				`{space}`
				`        - org: '*'`
				`        - org: '*'`
				`          space:`
				`space3`

`xs-api-url`	All	Yes	The URL for the application programming interface (API) in the XS advanced or Cloud Foundry environment Default: N/A **→ Tip** Deprecated: use `controller-url` instead.	CF: `https://api.cf.acme.ondemand.com` XS: `http://localhost:30030`

Parameter	Scope	Read-Only (System)	Default Value	Example
`xs-auth-url`	All	Yes	The authorization end point for the XS advanced or Cloud Foundry environment Default: N/A → Tip Deprecated: use `authorization-url` instead.	`https://localhost:9999/uaa-security`
`xs-type`	All	Yes	The XS type, for example, Cloud Foundry (CF) or XS advanced (XSA). Default: N/A	`CF`, `XSA`
`zdm-mode`	modules		Run blue-green deployment in zero-downtime-maintenance mode	`zdm-mode: true`

Metadata for Properties and Parameters

It is possible to declare metadata for parameters and properties defined in the MTA deployment description, for example, using the "`parameters-metadata:`" or "`properties-metadata:`" keys, respectively; the mapping is based on the keys defined for a parameter or property. You can specify if a property is required (`optional; false`) or can be modified (`overwritten: true`), as illustrated in the following (incomplete) example:

Code Syntax

```
modules:
 - name: frontend
   type: javascript.nodejs
   parameters:
     memory: 128M
     domain: !sensitive ${default-domain}
   parameters-metadata:
     memory:
       optional: true
       overwritable: true
     domain:
       overwritable: false
   properties:
     backend_types:
       order_management: sap-erp
       data_warehouse: sap-bw
   properties-metadata:
     backend_types:
       overwritable: false
```

```
        optional: false
        sensitive: false
```

The `overwritable:` and `optional` keywords for defining metadata are intended for use in extension scenarios, for example, where a value for a parameter or property is supplied at deployment time and declared in a deployment-extension descriptor file (`myDeploymentExtension.mtaext`). Parameters or properties defined in the `mtad.yaml` deployment descriptor with the metadata value `overwritable: false` cannot be overwritten by a value supplied from the extension descriptor. In this case, an error would occur in the deployment.

> → Tip
>
> Parameters or properties can also be declared as sensitive.

overwritable:

The effect of the metadata key `overwritable: [true | false]` differs depending on the value type of the property or parameter it modifies. For example, if a value is declared in a deployment-**extension** descriptor for a value that in the original deployment descriptor is declared with the metadata parameter `overwritable: false`, then the value in the extension descriptor is ignored. The following rules apply for the modification of the specified parameter or property:

- For parameters and properties of value type scalar (or an array), the overwritable operation replaces the value declared
- If a parameter (or property) value type is structured (for example, as a map or object), then "overwritable" means "extendable" or "merged"
- A parameter (or property) value type with no declared value can be overwritten by a either a scalar value or a structured value
- If there is mismatch in the value type declared for the original parameter (or property) in the MTA deployment descriptor and a corresponding value for the same parameter (or property) in an MTA deployment-**extension** descriptor, an error occurs with notification that the overwrite operation was not successful

sensitive:

The `sensitive` keyword enables you to hide sensitive information from public view, for example, if it is written to log or audit files. If a property or parameter is flagged as "sensitive", any related information is not written as plain text in log files; it is masked, for example, using a string of asterisks (`********`). You can declare a parameter or property as "sensitive" in either of the following ways:

- `!sensitive`
  Add the string `!sensitive` before the parameter value, for example: `domain: !sensitive ${default-domain}`
- `sensitive: true`
  Add the `sensitive: true` key to the list of `parameter-metadata`

Related Information

Create the MTA Description Files [page 137]

The MTA Deployment Descriptor [page 147]
Maintaining Application Services in XS Advanced [page 826]
Deploy a Multi-Target Application with Zero-Downtime Maintenance [page 221]
MTA Resource Types in the SAP CP Cloud Foundry Environment
Change Application Settings for the XS Advanced Java Run Time [page 737]

4.1.4 The MTA Deployment Extension Descriptor

Provide system-specific details that are not known until deployment time.

The MTA deployment descriptor provides the input required by the MTA deployer; it contains the necessary configuration for deployment and for passing values to the applications. However, in some instances, the final values are not known to the developer of the application. This type of system-specific information is **only** known to the administrator of the system on which the application is deployed.

For the cases where system-specific information is required at deployment time, it is possible to define extensions to the deployment configuration; the modifications must be specified in an MTA **extension**. An MTA extension is a file with the suffix `.mtaext`, for example, `myMTADeploymentExtension.mtaext`. The information defined in the deployment extension is used during deployment; the file name is specified as an option in the deployment command, for example:

Sample Code

```
xs deploy target/<MTA_archive>.mtar -e myMTADeployExtension.mtaext
```

Extension descriptors can be applied in the following scenarios:

- Deployment:
  In theory, the developer-created deployment descriptor can contain all the information required to deploy the corresponding MTA, but this is unlikely in reality. Usually, some deployment information will not be stored with the application code in a version-control system; it is added at deployment time. An extension descriptor provides a convenient way to declare such last-minute deployment-related information. In addition, multiple, alternative deployments can be configured by using multiple extension descriptors for the same application.

You use the MTA deployment **extension** to specify information that cannot be included in the MTA deployment descriptor, for example, because the details are not yet known, or differ from system to system.

- Proxy information
  For example, host name and port number for connections to a service
- Memory requirements
- URLs
  For connections to application modules
- User-provided service credentials

The syntax required in an MTA deployment extension is illustrated in the following example:

Note

The syntax used in the MTA deployment extension (`myMTAextension.mtaext`) is the same as the syntax required to create the MTA development descriptor (`mta.yaml`).

Sample Code

MTA Deployment Extension Description (`MyMTADeployExtension.mtaext`)

```
_schema-version: "2.0.0"
ID: com.sap.xs2.samples.javahelloworld.config1
extends: com.sap.xs2.samples.javahelloworld
modules:
  - name: java-hello-world
    parameters:
      memory: 128M
  - name: java-hello-world-backend
    parameters:
      memory: 512M
      instances: 1
```

You can use the MTA deployment extension descriptor to add the following information (at deployment time) to an MTA deployment descriptor:

- A property element, for example, a new property or parameter.
- A value for a property that does not already have a valued defined

Note

It is not possible to use the MTA deployment extension descriptor to add new `modules` or `resources`, or to add new `requires` or `provides` nodes.

Related Information

MTA Deployment Descriptor Syntax [page 173]
Create the MTA Description Files [page 137]

4.2 Deploy a Multi-Target Application (with Command-Line Tools)

Install a multi-target application (MTA) on SAP HANA.

Prerequisites

Deploying a multi-target application (MTA) to SAP HANA requires **some or all** of the following components and tools; which components or tools you need depends on what you are deploying: an MTAR archive or a folder structure:

- A file system containing the source files for the multi-target application you want to deploy

- A deployment description for the multi-target application; the mandatory file name is `mtad.yaml`
- An optional deployment description **extension** for the multi-target application deployment description, for example, `myMTADeploymentExtension.mtaext`
- An application's archive manifest `MANIFEST.MF`: required only if you create the MTAR deployment archive manually, for example, with the `jar` command
- An MTA archive, for example, `MTApp-Name.mtar` (optional)
- Knowledge of the `xs deploy` command as well as its parameters and options

Context

To deploy a multi-target application (MTA) to SAP HANA using the `xs` command-line tools, perform the following steps:

Procedure

1. Navigate to the parent folder of the file-system containing the application that you want to deploy.
2. Create a deployment descriptor (`mtad.yaml`) for your multi-target application.
3. Create an application manifest file (`MANIFEST.MF`) for the multi-target application.
4. Create an MTA archive.

   If you want to deploy a folder structure (including all sub-folders), you can skip this step.

   > **!** Restriction
   >
   > The MTA archive **must** have the extension `.mtar`, for example, `myMTApp.mtar`.

   1. Change directory to the parent folder containing **all** the application modules that you want to deploy.
   2. Create an MTA archive (MTAR) that contains the complete MTA folder structure (including all sub-folders).
      To create the MTA archive, you can use the following tools:
      - The MTA Archive (MTAR) Builder; for more information, see *Related Links*
      - Java's `jar` command, for example:
        ```
        jar cvMf <myMTApp>.mtar -C <myMTApp> .
        ```
      - Any archiving tool that produces Zip output
5. Deploy the multi-target application to SAP HANA.

   You can use either of the following deployment methods:

   - Deploy a **file system** (or part thereof) that includes MTA modules and the MTA deployment descriptor (`mtad.yaml`):
     1. Change directory to the folder containing the application you want to deploy.
     2. Use the `xs deploy` command to deploy the complete MTA folder structure (including all sub-folders) to the default target SAP HANA container.
        ```
        xs deploy <relative/path/to/MTA_folder>
        ```

- Deploy an **MTA archive**, which you create using Zip (or Java) tools:
  1. Use the `xs deploy` command to deploy the MTAR archive.
     `xs deploy <myMTApp_Name>.mtar`
  2. Check the MTA was successfully deployed.
     The information printed to the console should indicate the successful completion of the deployment process, as illustrated in the following example.

     Output Code

     ```
     SSL context configured!
     ...
     Upload finished successfully
     Starting process xs-deploy ...
      Processing MTA archive...
      MTA archive processed successfully.
      Building model...
      ...
     Process finished successfully
     ```

> → Tip
>
> If your deployment requires system-specific information that is only available at deployment time, you can include the information in a so-called MTA deployment **extension**. The file containing the extension details can then be specified as a deployment parameter, for example, `xs deploy --archive <myMTApp_Name>.mtar -e myMTADeploymentExtension.mtaext`

6. Check the deployment results.

   You can check that an MTA was successfully deployed by listing the corresponding services and applications and accessing the deployed application at the assigned URL.

   a. Log on to the XS controller.

   Sample Code

   ```
   xs login -a https://<hana-host>:3<instance-nr>30 -u <org-master-user>
   -p <org-master-pwd> --skip-ssl-validation
   ```

   b. List all running services.

   Sample Code

   ```
   xs services
   ```

   You should see the HDI service instance used by the application (for example, `nodejs-hdi-container`), and also the UAA service instance (for example, `nodejs-uaa`

   Output Code

   ```
   xs services
   name                      service  plan        bound apps
   ------------------------------------------------------------------------
   -------
   auditlog-db-container     hana     hdi-shared  auditlog-db, auditlog-
   server, ...
   auditlog-sbss             hana     sbss        auditlog-server, auditlog-
   broker, ...
   ```

```
...
hdi-container            hana     hdi-shared
...
nodejs-hdi-container     hana     hdi-shared  node-hello-world-db, node-
hello...
nodejs-uaa               xsuaa    default     node-hello-world-backend,
node-hello...
...
```

c. List all applications.

Sample Code

```
xs apps
```

A list of applications is displayed showing all applications that are part of the deployed MTA, for example, `node-hello-world-db`, `node-hello-world-backend`, and `node-hello-world`. The status of the `db` application should be `STOPPED`; the status of the other applications should be `STARTED`. The `db` application is stopped by the deployment process after the HDI deployment finishes, to prevent it from wasting resources.

Output Code

```
xs apps
name                        requested state   instances   memory   [...]
-------------------------------------------------------------------------
hdi-broker                  STARTED           1/1         256 MB   [...]
uaa-security-db             STARTED           1/1         256 MB   [...]
uaa-security                STARTED           1/1         2.00 GB  [...]
jobscheduler-db             STARTED           1/1         256 MB   [...]
jobscheduler-service        STARTED           1/1         1.00 GB  [...]
jobscheduler-broker         STARTED           1/1         1.00 GB  [...]
xsa-monitor                 STARTED           1/1         512 MB   [...]
deploy-service              STARTED           1/1         512 MB   [...]
product-installer           STARTED           1/1         512 MB   [...]
node-hello-world-db         STOPPED           0/1         256 MB   [...]
node-hello-world-backend    STARTED           1/1         256 MB   [...]
node-hello-world            STARTED           1/1         256 MB   [...]
```

→ Tip

The information displayed by the `xs apps` command (but not shown in the example above) also contains the URL and port number to use to access the deployed application, for example, `http://xsa007.acme.com:40003`. You can log on to the listed application at the assigned URL with the name and password of the user with the appropriate credentials.

d. List all deployed MTAs.

Sample Code

```
xs mtas
```

You should see the MTA that you just deployed, for example, `com.sap.xsa.samples.nodehelloworld`.

e. Display the URL assigned to a specific application.

Sample Code

```
xs app <AppName>
```

With the correct application-user credentials, you should be able to access the deployed application at the specified URL.

Output Code

```
xs app node-hello-world
Showing status and information about "node-hello-world"
  name:              node-hello-world
  requested state:   STARTED
  instances:         1
  memory:            128 MB
  disk:              <unlimited>
  buildpack:         sap_nodejs_buildpack
  urls:              https://host.acme.com:40009
Instances of droplet 1 created at May 7, 2017 11:35:52 AM:
index   created                  state     host            internal
port
------------------------------------------------------------------------
-
0       May 7, 2017 11:37:56 AM   RUNNING   host.acme.com   40323
```

Related Information

The MTA Deployment Descriptor [page 147]
The MTA Deployment Extension Descriptor [page 210]
The MTA Archive (MTAR) Builder [page 217]

4.2.1 The MTA Archive

An MTA archive is a Jar file containing deployable application modules.

A multi-target application (MTA) archive is used to deploy an application that comprises multiple modules in one deployment operation. An MTA archive contains all the application modules to be deployed; MTA archives have the file extension `.mtar`. MTA archives can be built either with the Java SDK `jar` command or with Zip-tools.

→ Tip

It is also possible to build an MTA archive with the tools included in the SAP Web IDE. Simply right-click the application project for which you want to create an MTA archive and choose *Build* > *Build* in the context menu displayed.

An MTA archive contains the following minimum mandatory content:

- One or more MTA modules
- Two mandatory configuration files in the `META-INF/` folder:
  - `MANIFEST.MF`
    The deployment manifest
  - `mtad.yaml`
    The deployment description

The MTA deployment tools assume that the content of an MTA archive has the following mandatory structure:

Sample Code

SAP HANA MTA Archive

```
/java-backend/
+--- backend.war
/ui5-frontend/
+--- ui.zip
/META-INF/
+--- MANIFEST.MF
+--- mtad.yaml
```

The MTA archive can be deployed using the MTA deployment tools, for example, the `xs deploy` command in a command shell or the build/run tools available in the SAP Web IDE.

4.2.2 The MANIFEST.MF File

The `MANIFEST.MF` file contains a list of the files in the standard Jar archive.

In the context of the SAP HANA MTA archive, the (`manifest.mf`) file contains the usual elements of the standard JAR manifest file plus a `Name` section for the deployment descriptor (`mtad.yaml`). For each MTA module, the `name` section links the module file (or directory name) to the corresponding MTA module name in the deployment descriptor. In the following example of a simple `manifest.mf` file one of the modules (`utils`) is a directory; the other two (`runner.war` and `builder.war`) are single files.

Sample Code

```
Manifest-Version: 1.0
Trusted-Library: true
Created-By: Apache Maven 3.2.5
Build-Jdk: 1.8.0_11

Name: META-INF/mtad.yaml
Content-Type: text/plain

Name: devx/runner.war
MTA-module: di-runner
Content-Type: application/war

Name: devx/utils
MTA-module: di-utils
Content-Type: content-type text/directory

Name: devx-di/builder.war
MTA-module: di-builder
```

```
Content-Type: application/war
```

The manifest file may also contain other standard jar-file information, for example, hash values and security signatures.

> **i Note**
>
> If the modules specified in the corresponding deployment description file (`mtad.yaml`) contain a `path` attribute, its value must be identical to the `Name` information in the `MANIFEST.MF` file.

Related Information

The MTA Deployment Descriptor [page 147]

4.2.3 The MTA Archive (MTAR) Builder

The MTAR builder is a command-line tool that packages a multi-target application (MTA) into an archive that you can deploy to the XS advanced platform.

The MTA archive builder is a standalone command-line tool that builds a deployment-ready MTAR (`.mtar` file) from the artifacts of an MTA project according to the project's MTA development descriptor (`mta.yaml` file). The MTA archive builder can be used on any file system and independently of the development environment in which the application project has been created.

> **i Note**
>
> The MTA archive builder is Java-based and runs on Windows and Linux operating systems. The MTA archive builder supports schema versions 2.0 and 2.1 in the MTA **development** descriptor (`mta.yaml`).

The build process and the resulting MTA archive depend on the target platform on which the archive will be deployed. The following table lists the target platforms that support the deployment of packages created with the MTA archive builder:

Target Platform	Description
CF	SAP Cloud Platform, Cloud Foundry development and run-time environment, which contains the Cloud Foundry application run time that is based on the open-source application platform managed by the Cloud Foundry Foundation
NEO	SAP Cloud Platform, NEO development environment, which enables you to develop HTML5 applications, Java (Tomcat) applications, and SAP HANA database applications running in the SAP HANA extended application services, classic model, run-time environment
XSA	SAP HANA extended application services (XS), advanced model, development and run-time environments

The following example shows how to use the MTAR builder to build an MTA archive called `myproject.mtar` that is intended for deployment on an XS advanced run-time platform (`--build-target=XSA`) using deployment-related information defined in the MTA extension descriptor `ext1.mtaext`:

Sample Code

Running the MTA Archive Build Tool

```
java -jar mta.jar -e=ext1.mtaext --build-target=XSA –-mtar=myproject.mtar
build
```

Tip

A deployment extension descriptor (for example, `ext1.mtaext`) is optional; it extends the details already defined in an application's deployment descriptor (`mtad.yaml`) and is only necessary in cases where system-specific details are not known to the application developer and need to be inserted at deployment time depending on the deployment target. For more information, see *Create the MTA Description Files* in *Related Information* below.

Each module in the MTA project is built using the build technology that is either associated with the module type by default (for example, `npm` for Node.js and `maven` for Java) or is configured explicitly in the `build-parameters` section of the MTA development descriptor (`mta.yaml`). The results of each module's build are packaged into an archive together with the corresponding MTA **deployment** descriptor (`mtad.yaml`). The deployment descriptor is created based on the following components:

- The structure of the MTA project or folder hierarchy
- The MTA development descriptor (`mta.yaml`)
- Any extension descriptors
  If any extension descriptors are supplied, the MTAR builder merges the information with the information provided in the `mta.yaml`.
- Any command-line options supplied when running the MTAR builder, for example, the build target (CF, Neo, or XSA)

In the MTA **deployment** descriptor, `mtad.yaml`, the design-time modules are converted into the module types that are recognized by the respective target platforms. The resulting MTA archive (`myarchive.mtar`) can then be deployed on the desired target platform.

Tip

The MTA Archive Builder is available for download from the SAP Development Tools Website as follows:

SAP Development Tools > *Multi-Target Application Archive Builder* > *mta_archive_builder-1.1.0.jar*

If the MTA archive builder encounters a module type in the **development** descriptor (`mta.yaml`) that it does not recognize, it copies the unrecognized module-type definition unchanged to the **deployment** descriptor (`mtad.yaml`) generated by the build process, and the `zip` builder is applied to the module. This means that archived module sources will be packaged as the build result.

All resource definitions specified in the development descriptor are passed to MTA archive unchanged; that is, without mapping and validations.

→ Tip

It is also possible to build an MTA archive with the tools included in the SAP Web IDE. For example, right-click the application project for which you want to create an MTA archive and choose *Build* > *Build* in the context menu displayed.

Related Information

SAP Software Download Center (Logon required)
The MTA Archive Builder User Guide
Create the MTA Description Files [page 137]

4.3 Deploy a Multi-Target Application with Blue-Green Deployment

In XS advanced, the deploy service supports the blue-green deployment of stateless applications modeled as a multi-target application (MTA).

Context

By running two identical production environments that are called “blue” and “green”, you can perform a blue-green deployment, which will eliminate the downtime and risk for your system.

! Restriction

There is no blue-green deployment for bound services. Blue and green applications are bound to the same service instances.

Procedure

1. Deploy your initial MTA (blue version).

   Run the following command in an XS CLI command shell:

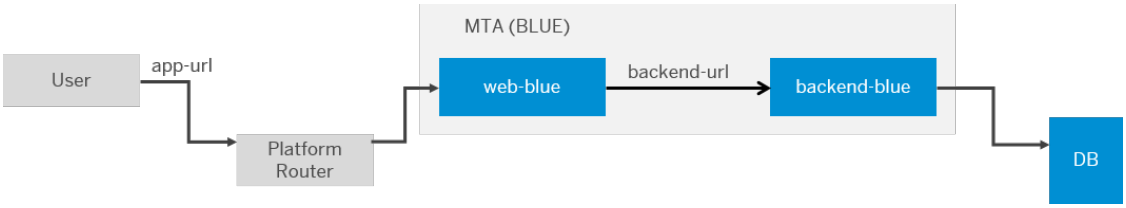
   ```
   $ [xs|cf] bg-deploy <your-mta-archive-v1>
   ```

   The deploy service performs the following actions:

   - Creates new applications

> **Note**
>
> If there are already installed applications, the deploy service will add “blue” to the existing application names

- Creates productive routes to the blue applications



2. Deploy your updated MTA (green version).

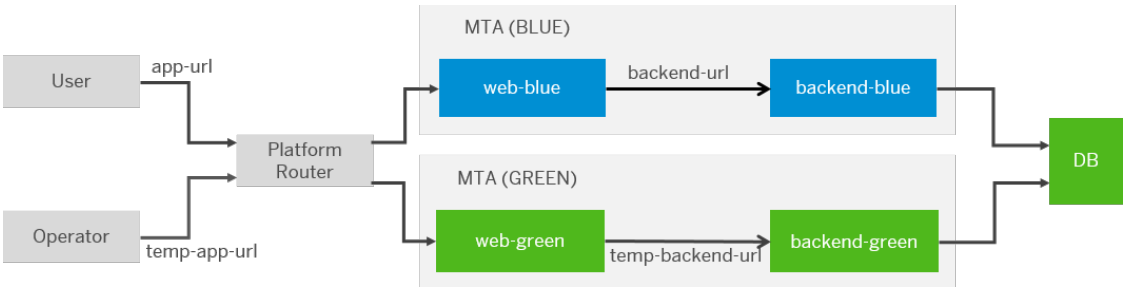
   Run the following command in an XS CLI command shell:

```
$ [xs|cf] bg-deploy <your-mta-archive-v2>
```

> **Note**
>
> The first action is that all MTA services are updated. The changes between the old and the new versions must be compatible. For example, the changes between the old and the new versions of the database tables, UAA configurations, and so on.

   The deploy service performs the following actions:

   - Creates new applications adding “green” to the existing application names
   - Creates temporary routes to the green applications



   - Interrupts the deployment process showing a message the one shown in the following example:

> **Output Code**
>
> User input in XS advanced (**xs**) or Cloud Foundry (**cf**)

```
Process needs additional user input
Use "[xs|cf] bg-deploy -i 469520 -a resume" to resume the process
Use "[xs|cf] bg-deploy -i 469520 -a abort" to abort the process
```

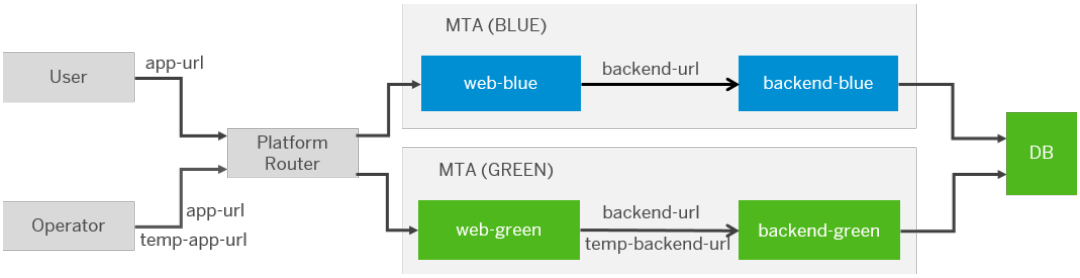
3. Test the green application version using the temporary routes.
4. Choose which MTA version you want to make available (blue or green).
   a. To keep the initial (blue) version of the MTA available in the productive system, stop the blue-green deployment process.

> **Note**
>
> Stopping the blue-green deployment process does not automatically remove the new green versions of the application or the temporary routes assigned to them.

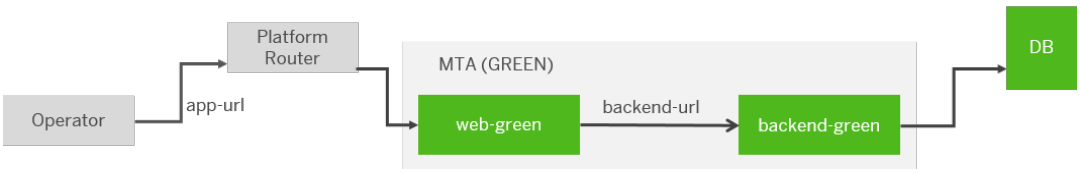
b. To make the new (green) version of the MTA available in the productive system, choose *Resume*.

This performs the following actions:

- Maps the productive routes to the new green versions of your MTA



- Deletes the temporary routes assigned to the new green version of the MTA
- Deletes the “blue” version of the MTA, which was productive before



Related Information

Deploy a Multi-Target Application with Zero-Downtime Maintenance [page 221]

4.4 Deploy a Multi-Target Application with Zero-Downtime Maintenance

Update an application that has database changes without any downtime.

Prerequisites

- The applications use HDI containers for persistence
- The application does not use a hard coded service name for the data source to the HDI service.

Context

When performing a blue-green deployment using the zero-downtime-maintenance (ZDM) deployment scenario, you can update an application that has database changes between the existing (blue) and the new (green) versions of the application. To update your application using the ZDM scenario, proceed as follows:

Procedure

1. Define the `com.sap.xs.hdi-container` resource type in the application's deployment descriptor.

   Sample Code

   ```
   _schema-version: 2.0.0
   ID: com.sap.xs2.samples.javahelloworld
   version: 0.1.0
   modules:
   ...
   resources:
     - name: java-hdi-container
       type: com.sap.xs.hdi-container
   ...
   ```

2. Run the deployment in ZDM mode.

   To enable zero-downtime-maintenance mode, you must declare the value `zdm-mode:true` as a parameter value of the `com.sap.xs.hdi` module type, as illustrated in the following example.

   Note

   You can define ZDM mode either in the application's deployment descriptor or in an extension descriptor.

   Sample Code

   ```
   _schema-version: 2.0.0
   ID: com.sap.xs2.samples.javahelloworld
   version: 0.1.0
   modules:
   ...
     - name: java-hello-world-db
       type: com.sap.xs.hdi
       path: db/
       parameters:
         zdm-mode: true
       requires:
         - name: java-hdi-container
   resources:
     - name: java-hdi-container
       type: com.sap.xs.hdi-container
   ...
   ```

3. To start the ZDM deployment, follow the steps described in *Deploy an MTA with Blue-Green Deployment*.

   For more information, see *Related Information*.

Related Information

Deploy a Multi-Target Application with Blue-Green Deployment [page 219]
Zero-Downtime Maintenance with Multi-Target Applications [page 223]
HDI Artifacts and Database Changes Supported by ZDM [page 226]

4.4.1 Zero-Downtime Maintenance with Multi-Target Applications

In XS advanced, the deploy service supports application updates using zero-downtime maintenance (ZDM).

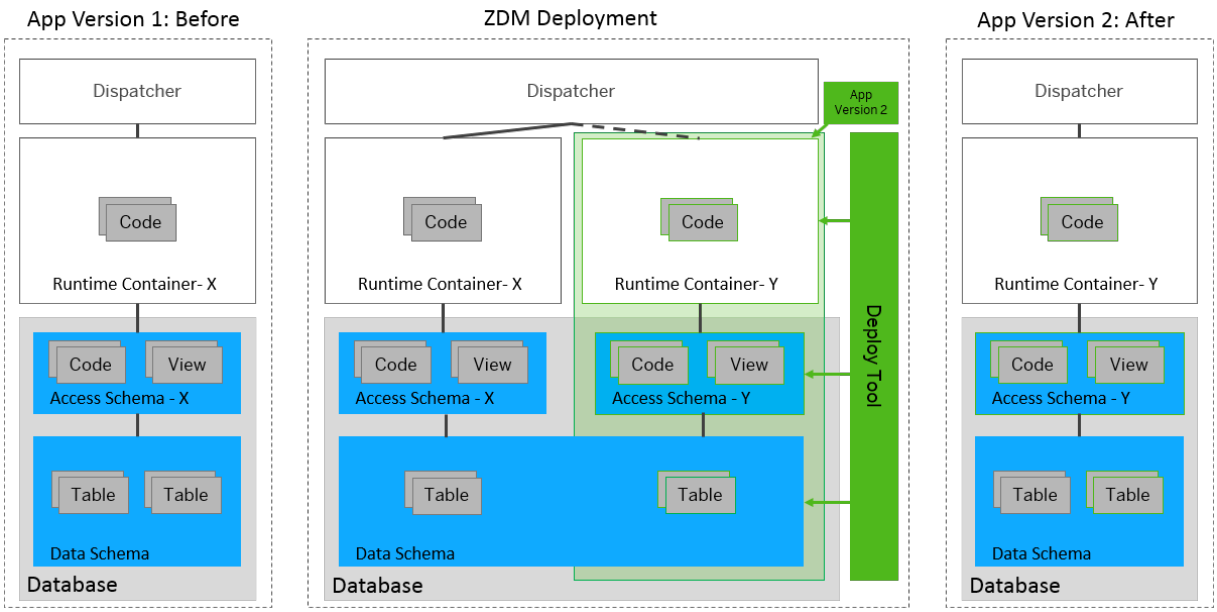
In XS advanced, zero-downtime updates of multi-target applications is achieved by deploying database artifacts in separate schemas, for example a “data” schema and an “access” schema.

- Data schema
  Contains all database objects that have persistence data, for example tables, sequences, and indexes.
- Access schema
  Contains the interface to database objects such as projection views and synonyms as well as database logic such as calculation views, database procedures, and functions.

> **i Note**
>
> Applications are indirectly bound to the access and data schemas. Only deployment and lifecycle management tools access the data schema directly.

The deploy-service supports the following operation modes:

- Normal
- ZDM
  Enables the update of applications using zero-downtime-maintenance. In this mode, the deploy service analyzes the application artifacts and deploys them into the `data` and `access` schemas by providing the corresponding roles in the `data` schema, granting any necessary permissions, and providing the corresponding interfaces to the `access` schema.



Depending on where and how the HDI artifacts are modeled in the XS advanced application's database `db/` module, the following methods are used to handle the ZDM deployment:

1. Default handling
   The artifacts are modeled in the default folders (`src/`, `cfg/`) of the application's database (`db/`) module. In this case, the deploy service handles the artifacts using the default rules and deploys them to the relevant default schema according to artifact type.
   During zero-downtime maintenance, the default deployment has the following limitations:
   - Some access schema objects (for example, interfaces and logic) are deployed in the `data` schema, which is unnecessary as this brings a performance reduction and violates the ZDM concept of schema separation.
     For example, CDS types should be separated: objects such as CDS entities should be used from the `data` schema and objects such as CDS views from the `access` schema. If these CDS types are defined in separate CDS files, all artifacts except the CDS view are deployed to both schemas.

     > → Tip
     >
     > It is not necessary to deploy `access` schema objects such as the CDS type used from the CDS view into the `data` schema.

   - `hdbtable` artifacts are deployed into the `data` schema by default.
     In certain situations, it is acceptable to deploy `hdbtable` artifacts into the `access` schema, for example, if there are corresponding `hdbtabledata` artifacts containing data that is used to fill the table defined in the `hdbtable` artifacts and, in addition, the data includes language text that is incompatible between versions.
     The default handling of ZDM deployment does not permit you to deploy `hdbtable` artifacts into an `access` schema; `hdbtable` artifacts are deployed into the `data` schema by default.
2. Separated handling using `data/` and `access/` folders.
   In this scenario the artifacts are modeled in the `data/` and `access/` folders within the `src/` and `cfg/` folders of the database module. The `data/` folder should contain the data-schema-related objects that have persistence data, for example, tables, sequences, or indexes. The `access/` folder should contain

access-schema-related objects (for example, interface-to-database objects such as projection views and synonyms) and the database logic (defined in calculation views, database procedures, and functions). The deploy service deploys the artifacts from the `data/` folder to the `data` schema and generates the corresponding interface objects in the `access` schema. Artifacts from the `access/` folder are deployed only to the `access` schema. This type of (separated) handling resolves the limitations listed above for the following reasons:

- Limitation 1 (objects in the `access` schema)
  Separate handling resolves this limitation because the `access` schema objects such as the CDS type used from CDS view are modeled in the `access/` folder and are deployed only to the `access` schema.
- Limitation 2 (`hdbtable` objects deployed by default to the `data` schema)
  Separate handling resolves this limitation because the `hdbtable` artifacts that should be deployed to the `access` schema are modeled in the `access/` folder and, as a result, are deployed only to the `access` schema.

Separated handling brings clarity to the database model as the location of database objects is more accurately defined. Separated handling also improves the separation of object types.

> **i** Note
>
> 1. Currently all CDS files modeled in the `data/` folder are deployed to both the `data` and `access` schemas.
> 2. Reusable database modules are supported.

To ensure that your applications support the ZDM update, follow the adoption guidelines described in *Database Artifacts and Changes Supported by ZDM* and model the HDI artifacts in `data/` and `access/` folders accordingly. ZDM is also supported with the default handling of HDI artifacts, but it has limitations when the data model is more complex. For more information about the database changes and database artifacts that are supported by ZDM in XS advanced, see *Related Information*.

Persistence in applications

ZDM deployment is possible for applications that use HDI containers for persistence. HDI containers are services that use the `hdi-shared` service plan.

> **i** Note
>
> Applications should not use a hard-coded service name for the data source to the HDI service, as during ZDM deployment the applications are bound to a new access HDI service with generated name, which could be different from the hard-coded name in the application code.

1. Java Applications
   Java applications can use dynamic data-source configuration in one of the following ways:
   1. By using an SAP JAVA buildpack
      Java applications can use a dynamic data-source-configuration feature that allows bound services described in the `manifest.yml` to appear as data sources available for look up in the `JNDI` of the application.
   2. By using the *Spring Cloud Spring Service Connector*
      For more details, see *Related Information*
2. Node.js applications

Should use `@sap/hana-client` for connection to the database.

Related Information

Deploy a Multi-Target Application with Zero-Downtime Maintenance [page 221]
HDI Artifacts and Database Changes Supported by ZDM [page 226]
Spring Cloud Spring Service Connector

4.4.2 HDI Artifacts and Database Changes Supported by ZDM

In XS advanced, zero-downtime maintenance (ZDM) supports the HDI artifacts and database changes listed in this topic.

If you are updating an XS advanced application using blue-green deployment in a zero-downtime-maintenance scenario, bear in mind the information included here about the possible impact the deployment can have on the HDI artifacts included in the application to be redeployed and any database changes required.

Supported database Changes in ZDM

The following database changes are supported during zero-downtime maintenance of multi-target applications in XS advanced:

- Modifications in the analytics layer (calculation views)
- Modifications of business logic (SQLScript procedures, table functions)
- Modifications in database abstraction Layer (database views)
- Modifications in the data model that do not require data transformation or migration, for example:
  - Adding or removing tables
  - Adding nullable table columns (without the `NOT NULL` constraint)
  - Adding table columns with default value (with the `DEFAULT` constraint)
  - Extending the length of text table columns

Supported Database Artifacts in ZDM

The table *Supported HDI Artifacts* below contains a description of the modeling of the supported HDI artifacts and the target schema where they should be deployed during the upgrade of an XS advanced application using zero-downtime maintenance.

The default target schema (data/access) is the target schema in which the artifact should be deployed most frequently (by default). The artifact is located in the default folder (`src/`, `cfg/`) of the `db` module, not into `data/` or `access/` folder and HDI Deployer applies default handling to the artifact.

Supported HDI Artifacts

HDI Artifact Type	Default target schema (data/access)	Adoption guidelines
`.hdbafllangprocedure`	access	-
`.hdbanalyticprivilege`	access	-
`.hdbapplicationtime`	data	-
`.hdbcalculationview`	access	-
`.hdbconstraint`	data	-

HDI Artifact Type	Default target schema (data/access)	Adoption guidelines
`.hdbcds`	access **and** data All `.hdbcds` artifacts that contain an entity, type, or table-type definition access (only) All `.hdbcds` artifacts that do **not** contain an entity, type, or table-type definition	1. Put CDS (temporary) entities into the `data/` folder, because in the access schema, CDS (temporary) entities only produce projection views. 2. Put CDS views only in the `access/` folder. CDS types and CDS table types should not be used by both entities and views or procedures. Separate CDS types and CDS table types for `data/` and `access/` folders respectively. 1. `data/` folder Define CDS types and CDS table types which are used only by CDS entities. Do not make changes to data types which are backwards-incompatible. 2. `access/` folder Define CDS types and CDS table types which are used by procedures, views, and/or table types, but not from entities. 3. Associations defined in CDS entities can be used only by CDS views, but not by `.hdbviews`. In the `data/` folder do not model associations to objects from `access/` folder. 4. Put CDS files containing DCL (Data Control Language) objects only in the `access/` folder.
`.hdbcollection`	data	-

HDI Artifact Type	Default target schema (data/access)	Adoption guidelines
`.hdbflowgraph`	access	1. Replication tasks (`.hdbreptask`) and flowgraphs (`.hdbflowgraph`) should replicate into tables from the `data/` folder. 2. Put replication tasks (`.hdbreptask`) and flowgraphs (`.hdbflowgraph`) into the `access/` folder, and make sure that they are not active from new and old version at the same time.
`.hdbfulltextindex`	data	-
`.hdbfunction`	access	-
`.hdbgraphworkspace`	access	-
`.hdbmrjob`	access	-
`.jar`	access	-
`.hdbindex`	data	-
`.hdblibrary`	access	-
`.hdblogicalschema`	access	-
`.hdbprocedure`	access	-
`.hdbprojectionview`	access	-
`.hdbprojectionviewconfig`	access	-
`.hdbpublicsynonym`	access	-
`.hdbreptask`	access	See `.hdbflowgraph.`
`.hdbresultcache`	access	-
`.hdbrole`	access	-
`.hdbroleconfig`	access	-
`.hdbsearchruleset`	access	-
`.hdbsequence`	data	-

HDI Artifact Type	Default target schema (data/access)	Adoption guidelines
`.hdbview`	access	-
`.hdbstatistics`	data	-
`.hdbstructuredprivilege`	access	-
`.hdbsynonym`	access	-
`.hdbsynonymconfig`	access	-
`.hdbsysbicsynonym`	access	-
`.hdbtable`	data	1. Tables (`hdbtable`) are usually put into the `data/` folder. In certain situations, it is valid to put them into the `access/` folder, for example, if there are corresponding `hdbtabledata` files that fill these tables with language texts that is different or incompatible between versions. Put CDS entities in the `data/` folder. **⚠ Caution** If you put CDS entities in the `access` schema, this would create projection views. 2. Do not model associations for tables (`hdbtable`). Associations are not visible in the `access` schema because they are not exposed through the generated projection views. Do not point to objects in the `access` schema. 3. Table name and column names should be enclosed in quotes, for example, (`"column_name"`).

HDI Artifact Type	Default target schema (data/access)	Adoption guidelines
`.hdbdropcreatetable`	data	1. Put `hdbdropcreatetable` temporary tables into the `access/` folder because they are related to application logic (for example, `procedures`). 2. If `hdbdropcreatetable` temporary tables are put into the `data/` folder, they will be deployed there but cannot be used. There is no projection view created for them in the `access` schema.
`hdbtabledata`	data	`hdbtabledata` files with language texts for calculation views must be put into the `access/` folder. If they are put into the `data/` folder, they do not do any harm, but cannot be used.
`.hdbmigrationtable`	data	-
`.csv`	data	See `.hdbtabledata`.
`.properties`	data	See `.hdbtabledata`.
`.tags`	data	See `.hdbtabledata`.
`.hdbtabletype`	access	-
`.hdbtextconfig`	access	-
`.hdbtextdict`	access	-
`.hdbtextrule`	access	-
`.hdbtextinclude`	access	-
`.hdbtextlexicon`	access	-
`.hdbtextminingconfig`	access	-
`.hdbtrigger`	access	Backwardly incompatible changes in triggers are not supported. Version 2 of the application has to be in read-only mode, which is not supported.
`.hdbvirtualfunction`	access	-
`.hdbvirtualfunctionconfig`	access	-

HDI Artifact Type	Default target schema (data/access)	Adoption guidelines
`.hdbvirtualpackage`	access	-
`.hdbvirtualpackagehadoop`	access	-
`.hdbvirtualpackagesparksql`	access	-
`.hdbvirtualprocedure`	access	-
`.hdbvirtualprocedureconfig`	access	-
`.hdbvirtualtable`	access	-
`.hdbvirtualtableconfig`	access	-
`.txt`	access	-
`.hdiconfig`	data and access	-
`.hdinamespace`	data and access	-
`.hdbgrants`	access	-

Related Information

Deploy a Multi-Target Application with Zero-Downtime Maintenance [page 221]
Zero-Downtime Maintenance with Multi-Target Applications [page 223]

5 Defining the Data Model in XS Advanced

The application data model comprises all the database artifacts used to store and provision data for your application's back end and user interface.

As part of the process of defining the database persistence model for your application, you create database design-time artifacts such as tables and views, for example using SQL Data Definition Language (DLL) or Core Data Services (CDS). However, you can also create procedures and functions, for example, using SQLScript, which can be used to insert data into (and remove data from) tables or views. For database applications, the structure of the design-time resources should look something like the `db/` module illustrated in the following example:

Sample Code

```
<MyAppName>
|- db/                              # Database deployment artifacts
|  |- package.json                  # Database details/dependencies
|  \- src/                          # Database artifacts
|     |- .hdiconfig                 # HDI build plug-in configuration
|     |- .hdinamespace              # HDI run-time name-space configuration
|     |- data/                      # Database artifacts
|     |  |- myTable.hdbtable        # SQL DDL table definition
|     |  |- myView.hdbview          # SQL DDL view definition
|     |  |- myEntity.hdbcds         # CDS table definition
|     |  |- myDataType.hdbcds       # CDS type definition
|     |  |- myDoc.hdbcds            # CDS data model definition
|     |  \- mySynonym.hdbsynonym    # SQL DDL synonym definition
|     \- procedures                 # Database procedures, functions, ...
|        |- myProc.hdbprocedure     # SQL DDL procedure
|        \- myFunc.hdbfunction      # SQL DDL function
|- web/
|  |- xs-app.json
|  \- resources/
|- js/
|  |- start.js
|  |- package.json
|  \- src/
|- security/
|  \- xs-security.json
\- mtad.yaml
```

The following steps describe the high-level steps you perform when defining, deploying, and consuming the database persistence model for an application deployed to the XS advanced model run time.

1. Define the data model.
   Set up the folder structure for the design-time representations of your database objects; this could include CDS documents that define tables, data types, views, and so on. But it could also include other database artifacts, too, for example: your stored procedures, synonyms, sequences, scalar (or table) functions, and so on.
   In addition, you can also define the analytic model, for example, the calculation views and analytic privileges that are to be used to analyze the underlying data model and specify who (or what) is allowed access.
2. Set up the SAP HANA HDI deployment infrastructure.

This includes the following components:

- The HDI configuration
  Map the design-time database artifact type to the corresponding HDI build plug-in in the HDI configuration file (`.hdiconfig`). The mapping is determined by the file extension, for example, artifacts with the `.hdbtable` extension must be mapped to the HDI build plug-in `com.sap.hana.di.table`; artifacts with the `.hdbcds` extension must be mapped to the `com.sap.hana.di.cds`.
- Run-time name space configuration
  Define rules that determine how the run-time name space of the deployed database object is formed. For example, you can specify a base prefix for the run-time name space and, if desired, specify if the name of the folder containing the design-time artifact is reflected in the run-time name space that the deployed object uses.
  Alternatively, you can specify the use of freestyle names, for example, names that do not adhere to any name-space rules.

3. Deploy the data model.
   Use the design-time representations of your database artifacts to generate the corresponding active objects in the database catalog.
4. Consume the data model.
   Reference the deployed database objects from your application, for example, using OData services bound to UI elements.

Related Information

Create the Data Persistence Artifacts with CDS in XS Advanced [page 311]
Design-Time Database Resources in XS Advanced [page 277]
Managing SAP HDI Containers and Artifacts [page 234]

5.1 Managing SAP HDI Containers and Artifacts

In SAP HANA Deployment Infrastructure (HDI), database development artifacts are deployed to so-called containers.

SAP HDI provides a service that enables you to deploy database development artifacts to HDI containers. This service includes a family of consistent design-time artifacts for all key SAP HANA platform database features which describe the target (run-time) state of SAP HANA database artifacts, for example: tables, views, or procedures. These artifacts are modeled, staged (uploaded), built, and deployed into SAP HANA.

> **i Note**
>
> The HDI focuses strictly on deployment; HDI does not include any version-control tools, nor does it provide any tools for life-cycle management.

In the context of XS advanced, the SAP HANA service broker is used to create and drop HDI containers for XS advanced applications; for pure HDI content development outside the scope of XS advanced, the HDI SQL APIs also enable the maintenance and management of HDI containers and the creation of content in the containers.

The HDI deployment tools deploy database artifacts to an HDI container. Each HDI container comprises a design-time container (DTC) and a run-time container (RTC). Design-time database objects are typically located in the `db` folder of the XS advanced application's design-time hierarchy, as illustrated in the following example:

Output Code

```
app1-hello-world
|- db
   \- src
      |- .hdiconfig                       # HDI build-plugin configuration
      |- .hdinamespace                    # Run-time name-space configuration
      |- myCDSdoc.hdbcds                  # CDS document
      |- myTable.hdbtable                 # DB table definition
      |- mySQLView.hdbview                # SQL view definition
      |- mySynonym.hdbsynonym             # DB synonym definition
      |- mySynoConfig.hdbsynonymconfig    # DB synonym configuration file
      |- myProcedure.hdbprocedure         # DB procedure definition
      \- myImportData.hdbtabledata        # Data-import definition
```

The deployment process in XS advanced and the build-and-make process in HDI API enable you to populate the database run-time with the specified catalog objects. In addition to database artifacts, HDI also enables you to import and export table content such as business configuration data and translatable texts.

! Restriction

HDI enables you to deploy database objects only; it is not possible (or necessary) to deploy application-layer artifacts such as JavaScript programs or OData objects.

As part of the maintenance and management of containers, system administrators typically perform the following operations:

- Configure access to the SAP HANA Deployment Infrastructure (including containers)
- Create and remove HDI containers, each consisting of both a design-time and a run-time container

The configuration of HDI containers involves the creation and configuration of the following design-time artifacts:

- Container deployment configuration (`.hdiconfig`)
  A JSON file containing a list of the bindings between database artifact types (for example, sequence, procedure, table) and the corresponding build and deployment plug-in (and version).
- Run-time container name-space rules (`.hdinamespace`)
  A JSON file containing a list of design-time file suffixes and the naming rules for the corresponding runtime locations.

  → Tip

  You can apply the rules defined in the `.hdinamespace` file either exclusively to the folder it is located in or to both the folder it is located in **and** its subfolders.

Related Information

SAP HDI Containers [page 240]
SAP HDI in the Context of XS Advanced
SAP HDI Container Content Development with the HDI API

5.1.1 Naming Conventions in SAP HDI

Rules and restrictions apply to the names of artifacts in the SAP HDI virtual file system

In SAP HANA Deployment Infrastructure (HDI), bear in mind the rules listed below when specifying the name of a database object that is deployed to a HDI container and written to the HDI virtual file system. For HDI run-time objects and their name spaces, valid characters are those that are also valid for catalog names (described in the *SAP HANA SQL and System Views Reference*), except for the following restrictions:

- Name space names cannot contain the forward-slash character (`/`).
- The backslash (\) can be used but must be escaped in the `.hdbnamespace` file with a second backslash.
- Name space and object names cannot contain the colon character (:)
- The double-colon (::) is used as separator between the name space and the object name.

  > **i** Note
  >
  > HANA 2 SPS01 Rev 11: For synonyms, the separator can also be the forward-slash character (`/`). The forward slash should only be used for migration scenarios where the double-colon cannot be used.

- The name space name can be empty, in which case no separator is used. If a name space **is** used, then object names must include the declared name space as a prefix, for example, `<namespace>::myObject`

> **!** Restriction
>
> Due to catalog rules, the maximum length of the fully qualified object name (including the name space) is 127 characters.

In the HDI virtual file system, the following restrictions apply to the names of files and folders:

- Names can include only a subset of ASCII, namely, the following characters:
  - Lower- or upper-case letters (aA-zZ)
  - Digits (0-9)
  - In addition to the standard characters listed above, HDI files and folders can also include the following special characters:
    ! # % & ' ( ) + , - . ; = @ [ ] ^ _ ` { } ~
- The maximum length of the object's name (including the path to the source file) is 255 characters in the HDI virtual file system.
- The forward slash (`/`) is used as the folder delimiter in the HDI path:
  `path/to/HDI/object.hdbsuffix`
- An HDI path must **not** start with a forward slash (`/`). Only relative paths are permitted, for example:
  `path/to/HDI/object.hdbsuffix`
- Trailing spaces ' ' or periods '.' are not allowed in the name of an HDI file or folder
  - Trailing space:
    `path/to/HDI/object.hdbsuffix<trailing space>`

    - Trailing period:
      `path/to/HDI/object.hdbsuffix.`
- For HDI folders:
    - An empty path denotes the root folder
    - Other paths must end with a forward slash (`/`)
        - `path/`
        - `path/to/`
        - `path/to/HDI/object/`
- Core Data Services (CDS)
  Additional restrictions apply to the names of CDS artifacts deployed to an HDI container. For more details, see *Related Information*.
- SAP Web IDE for SAP HANA
  If you are using SAP Web IDE for SAP HANA to develop applications for XS advanced, additional restrictions apply to the names of design-time artifacts. For more details, see *Related Information*.

Related Information

Naming Conventions in CDS (SAP HANA Developer Guide) [page 327]
SAP Web IDE for SAP HANA Reference [page 1060]

5.1.2 Set up an SAP HDI Container for XS Advanced Applications

Set up the environment required for the deployment to the SAP HANA Deployment Infrastructure (HDI) of an XS advanced application's database artifacts.

Context

In SAP HDI, the design-time artifact types for XS advanced applications are distinguished by the artifact's file suffix (for example, `.hdbtable`). These artifact types must be associated with a corresponding build plug-in to ensure the successful build and deployment. The binding between an artifact type and a corresponding build plug-in is established by the contents of the `.hdiconfig` file, which must be deployed into the target container, too.

In the context of XS advanced, the service broker is used to create HDI containers for the deployed application. An HDI container is a set of schemata and a set of users that, together, enable an isolated deployment of database artifacts. From an application perspective, it is just a technical user and a schema.

For backward compatibility with legacy applications, the deployed objects can also be assigned to a run-time name space, which is defined in the `.hdinamespace` file that also has to be deployed to the target HDI container.

→ Tip

If you are using SAP Web IDE, the configuration described here is available by default. If you are working in a command shell, use **either** the `xs` command (on-premise environments) **or** the `cf` command (SAP Cloud Foundry) in the following steps.

Procedure

1. Check that the HDI service plan `hdi-shared` is available for your organization or space.

```
[xs | cf] marketplace
Getting services from marketplace...
service       plans                                    description
------------------------------------------------------------------------------
---
fs-storage    free                                     xs file service
provides...
hana          hdi-shared, sbss, schema, securestore  SAP HANA database
xsuaa         default, devuser, space                Manage app
authorization...
managed-hana  hdi-shared, schema, securestore        Creates service
instance...
auditlog      free                                     Broker for Audit log
Service
```

2. Create an HDI container using the service broker.

```
[xs | cf] create-service hana hdi-shared <myHDIcontainer>
```

3. Bind the new HDI container to the application that you want to use the container.

```
[xs | cf] bind-service <myApplication> <myHDIcontainer>
```

4. Configure the build plug-ins required for the artifacts deployed to the HDI container.

   By default, a file suffix (for example, `.hdbprocedure`) is not bound to a specific version of a build plug-in. Instead, the binding is established by the contents of the `.hdiconfig` file, which must be deployed into a container along with the database artifacts it lists. An `.hdiconfig` file is a JSON file with the structure illustrated in the following example:

   ‹≡› Sample Code

   Example `.hdiconfig` File

```
{
"plugin_version": "2.0.30.0"
"file_suffixes" : {
    "hdbtable" : {
        "plugin_name" : "com.sap.hana.di.table"},
    "hdbview" : {
        "plugin_name" : "com.sap.hana.di.view"},
    "hdbprocedure" : {
        "plugin_name" : "com.sap.hana.di.procedure"},
    "<hdb_file_suffix_#>" : {
        "plugin_name"   : "<plugin_NAME>",
        "plugin_version": "<local_plugin_VERSION>"}
    }
}
```

> → Tip
>
> Add to the `.hdiconfig` file, the file-suffixes of the design-time artifacts deployed by your applications. The `.hdiconfig` file must be placed in the root folder of the database source-files, for example, `app1/db/.hdiconfig`.

5. Configure the run-time name spaces for the objects added to the run-time container during application deployment.

   In SAP HANA HDI, name-space rules are defined in one or more file resources named `.hdinamespace`, which must be located in the design-time folder to which the naming rules apply - or the root folder of a hierarchy to which the naming rules apply. The content of the `.hdinamespace` file is specified according to the JSON format with the following structure:

   > → Tip
   >
   > Change the path defined in `name` and, for `subfolder`, choose between `append` (add sub-folder name to object name space) and `ignore` (do **not** add sub-folder name to the object name space).

   > Sample Code
   >
   > `src/.hdinamespace`

   ```
   {
      "name"      : "com.sap.hana.example",
      "subfolder" : "<[append | ignore]>"
   }
   ```

6. Deploy the application.

   The deployment operation pushes the specified application's database objects to the HDI container to which it is bound.

   - Command-line interface:
     `[xs | cf] deploy`
   - SAP Web IDE:
     Build > *Build*

     > i Note
     >
     > You can use the *Database Explorer* tool to view the contents of the XS application's run-time container.

Related Information

The SAP HDI Container Configuration File [page 241]
SAP HDI Container Configuration File Syntax [page 243]
SAP HDI Artifact Types and Build Plug-ins Reference
Managing SAP HDI Containers and Artifacts [page 234]

5.1.2.1 SAP HDI Containers

An SAP HANA HDI container consists of a design-time container and a corresponding run-time container.

The SAP HANA Deployment Infrastructure (HDI) uses so-called containers to store design-time artifacts and the corresponding deployed run-time (catalog) objects. The HDI makes a strict separation between design-time and run-time objects by introducing the following, distinct container types:

- **Design-time** container (DTC)
  Provides an isolated environment for the storage of design-time files
- **Run-time** container (RTC)
  Stores the deployed objects built according to the specification stored in the corresponding design-time artifacts

> → Tip
>
> HDI uses so-called build plug-ins to create run-time objects from the corresponding design-time files, for example, a procedure object in the database catalog from a design-time definition of a stored procedure.

Design-Time Containers

In SAP HDI, the design-time container is used to store the design-time representations of the catalog objects that you want to create during the deployment process. The design-time container does not provide direct access to its database storage; it contains meta-data that can only be accessed by means of the HDI application-programming interface (API). The HDI API enables access to design-time resources inside a design-time container by means of the following file systems:

- work
  A read-write file system where developers can create, modify, and delete design-time resources. The resources in the **work** file system are not modified by HDI, so undeploying a resource from the **deployed** file system leaves the corresponding resource in the work file system untouched. This is because deployed resources are handled by the **deployed** file system.
- deployed
  A read-only file system containing the design-time resources that have been **deployed**. The deployed file system can only be changed by HDI during the deployment and undeployment process.

> ! Restriction
>
> For maximum isolation and increased security, content stored in a design-time container is owned by a dedicated technical user.

Build Plug-ins are used to transform a certain type of design-time file into the corresponding run-time objects. For this reason, each design-time resource type must be mapped to a build plug-in for each container that is created. It is not possible to create a catalog object from a design-time resource without having a plug-in which can read and transform this source definition during deployment.

A design-time container is always attached to a corresponding **target** run-time container.

Run-Time Containers

In SAP HANA, the database run-time container stores the deployed objects built according to the specification stored in the corresponding design-time artifacts. During the deployment and undeployment operations, all interaction with the run-time container is performed by build plug-ins, which are mapped to distinct design-time artifact types in the container-configuration file (`.hdiconfig`).

A run-time container does not know about the concept of design-time containers. As a consequence, the run-time container can be identified with a standard database schema without any need for HDI meta-data.

The creation of a run-time container results in a new technical database user and a schema with the same name as the one chosen for the name of the run-time container itself. Access to content in the run-time container is controlled by database privileges, for example, by granting SELECT or EXECUTE privileges.

Build Plug-ins

The transformation of design-time resources into corresponding run-time (catalog) objects as well as the extraction of dependency information is performed by so-called build plug-ins. A newly created design-time container cannot access any plug-ins; this access must be defined in the HDI configuration file (`.hdiconfig`). In addition, since there is no default behavior configured, it is not possible to deploy any design-time files (except the two HDI container-configuration and name-space configuration files `.hdiconfig` and `.hdinamespace` respectively) without configuring a build plug-in first. To configure a design-time container to use certain build plug-ins, you must set up and deploy a `.hdiconfig` container-configuration file.

> → Tip
>
> By default, a file suffix (for example, `.hdbprocedure`) is not bound to a specific version of a known build plug-in. Instead, the binding is established by the contents of the `.hdiconfig` file, which must be deployed into a container.

Related Information

The SAP HDI Container Configuration File [page 241]
Managing SAP HDI Containers and Artifacts [page 234]

5.1.2.2 The SAP HDI Container Configuration File

Bind design-time file types to the corresponding build plug-in required in the SAP HANA Deployment Infrastructure (HDI).

In SAP HDI, a design-time container defines an isolated environment for design-time files. To create run-time objects from the corresponding design-time files (for example, a catalog object from a source definition of

a stored procedure), HDI uses so-called Build Plug-ins. It is not possible to create a catalog object from a design-time resource without having a plug-in which can read and transform this source definition. For this reason, each design-time resource type must be mapped to a build plug-in for each container that is created. This mapping is configured in an HDI container-configuration file: a file with no name but the mandatory file suffix `.hdiconfig`.

> **Note**
>
> The `.hdiconfig` file must be located in the root folder of the database module. However, you can include additional `.hdiconfig` files further down the source-file hierarchy, for example, if you want to change (expand/restrict) the definitions at a certain point in the hierarchy.

By default, a file suffix (for example, `.hdbprocedure`) is not bound to a specific version of a known build plug-in. Instead, the binding is established by the contents of the `.hdiconfig` file, which must be deployed into the container along with the artifacts it describes. An `.hdiconfig` file is a JSON file with the following structure:

Sample Code

```
{
"plugin_version": "2.0.30.0",
"file_suffixes" : {
   "hdbsynonym" : {
      "plugin_name" : "com.sap.hana.di.synonym"},
   "hdbview" : {
      "plugin_name" : "com.sap.hana.di.view"},
   "hdbcalculationview" : {
      "plugin_name" : "com.sap.hana.di.calculationview"},
   "hdbprocedure" : {
      "plugin_name" : "com.sap.hana.di.procedure"},
   "<hdb_file_suffix_#>" : {
      "plugin_name" : "<plugin_NAME>" }
    }
}
```

It is also possible to define the plug-in version for individual artifact plug-ins, as illustrated in the following example:

Sample Code

```
{
"file_suffixes" : {
   "hdbsynonym" : {
      "plugin_name"   : "com.sap.hana.di.synonym",
      "plugin_version": "2.0.30.0"},
   "hdbview" : {
      "plugin_name"   : "com.sap.hana.di.view",
      "plugin_version": "2.0.30.0" },
   "hdbcalculationview" : {
      "plugin_name"   : "com.sap.hana.di.calculationview",
      "plugin_version": "2.0.30.0"},
   "hdbprocedure" : {
      "plugin_name"   : "com.sap.hana.di.procedure",
      "plugin_version": "2.0.30.0"},
   "<hdb_file_suffix_#>" : {
      "plugin_name"   : "<plugin_NAME>","<plugin_VERSION>" }
    }
}
```

> **i** Note
>
> If the plug-in version is defined at both the global and local (individual) level, as illustrated in the following example, then the version specified for an HDI build plug-in at the individual level takes precedence.

```
{
"plugin_version": "2.0.30.0",
"file_suffixes" : {
   "hdbsynonym" : {
      "plugin_name"   : "com.sap.hana.di.synonym"},
   "hdbview" : {
      "plugin_name"   : "com.sap.hana.di.view"},
   "hdbcalculationview" : {
      "plugin_name"   : "com.sap.hana.di.calculationview",
      "plugin_version": "2.0.20.0"},
   "hdbprocedure" : {
      "plugin_name"   : "com.sap.hana.di.procedure",
      "plugin_version": "2.0.20.0"}
    }
}
```

The plug-in **name** corresponds to the plug-in name in reverse-URL notation, for example, `com.sap.hana.di.procedure` or `com.sap.hana.di.view`.

The plug-in **version** is described as follows: **major** version, **minor** version, **revision** version, and an optional **patch** version, for example, `2.0.20.0` signifies the SAP HANA version 2.0 SPS 02. Similarly, `1.0.12.0` signifies the SAP HANA version 1.0 SPS 12.

> **i** Note
>
> The three level plug-in version format previously used in SAP HANA 1.0, for example, "`13.1.0`", is still valid. Values in the old format will be internally converted to the appropriate four level format for SAP HANA 1.0 revisions.

Related Information

Managing SAP HDI Containers and Artifacts [page 234]
SAP HDI Artifact Types and Build Plug-ins Reference

5.1.2.3 SAP HDI Container Configuration File Syntax

In SAP HANA Deployment Infrastructure (HDI), the JSON syntax is used to format the content of the HDI container-configuration file (`.hdiconfig`).

> Sample Code

```
{
  "plugin_version": "<plugin_VERSION>"
  "file_suffixes" : {
    "<file_suffix_1>" : {
      "plugin_name"   : "<plugin_NAME>",
```

```
      "plugin_version": "<plugin_VERSION>" // If different from global
version
    },
    "<file_suffix_2>" : {
      "plugin_name"   : "<plugin_NAME>"
    },
    "<file_suffix_#>" : {
      "plugin_name"   : "<plugin_NAME>",
      "plugin_version": "<plugin_VERSION>" // If different from global
version
    },
    {
      [...]
    }
  }
}
```

file_suffixes

Define one or more file suffixes which you want to bind to a build plug-in for deployment to an HDI container. The file suffix specifies the **type** of database artifact, for example, `.hdbsequence` or `.hdbview`.

Sample Code

```
{
  "file_suffixes" : {
    "<file_suffix_1>" : {
      "plugin_name"   : "<plugin_NAME>",
      "plugin_version": "<plugin_VERSION>" //optional
    }
  }
}
```

You can use the asterisk character (*) to define a **global** file suffix which binds all remaining, unbound file suffixes to a specific build plug-in, for example, to bind all unbound file suffixes to the `Copy Only` plug-in.

Restriction

All "`hdi*`" file suffixes (for example, `.hdiconfig` or `hdinamespace`) are reserved for use with HDI; these file suffixes cannot be (re)configured.

The following example of an abbreviated HDI configuration file illustrates how the design-time artifact types (for example, `.hdbsynonym`, `.hdbtable`, `.hdbcalculationview`) are mapped to their corresponding HDI build plug-in:

Code Syntax

`.hdiconfig`

```
{
  "plugin_version" : "2.0.20.0",   // global, optional, defaults to 0.0.0.0
  "file_suffixes" : {
      "hdbsynonym" : {
        "plugin_name" : "com.sap.hana.di.synonym"
      },
      "hdbsynonymconfig" : {
```

```
        "plugin_name" : "com.sap.hana.di.synonym.config"
      },
      "hdbtable" : {
        "plugin_name" : "com.sap.hana.di.table"
      },
      "hdbdropcreatetable" : {
        "plugin_name" : "com.sap.hana.di.dropcreatetable"
      },
      "hdbcalculationview" : {
        "plugin_name" : "com.sap.hana.di.calculationview"
      },
      "hdbcds" : {
        "plugin_name" : "com.sap.hana.di.cds",
        "plugin_version": "2.0.19.0"
      },
      "<file suffix 2>" : {
       "plugin_name"   : "<plugin name>",
       "plugin_version": "<plugin_version>" //local, optional
      }
}
```

plugin_name

Use the `plugin_name` key to specify the name of the HDI build plug-in that you want to associate with a specific HDI artifact type.

Sample Code

```
"plugin_name"   : "<plugin_NAME>",
```

For example, you can associate the artifact suffix for the HDB SQL view (`.hdbview`) with the build plug-in `com.sap.hana.di.view`.

Sample Code

```
"plugin_name"   : "com.sap.hana.di.view",
```

plugin_version (optional)

Use the optional `plugin_version` key to specify the version number of the HDI build plug-in that you want to associate with one or more HDI plug-ins.

Sample Code

```
"plugin_version":
"<Major_Version>.<Minor_Version>.<Revision_Version>.<Patch_Version>"
```

→ Tip

To specify one plug-in version for **all** configured plug-ins, define `"plugin_version"` globally at the start of the configuration file. To specify a specific version for a particular plug-in, define `"plugin_version"` locally alongside the plug-in name to which the specific version refers. The locally defined plug-in version takes precedence over the globally defined version.

The value of the `plugin_version` key is interpreted as a minimal required version of the build plug-in. Version numbers follow the format `<major>.<minor>.<Revision_Version>.<patch>`. An installed build plug-in with the same major version is considered to be compatible if its minor version is greater than the requested minor version, or its minor version is equal to the requested version and the revision or patch number is greater than (or equal to) the requested version.

Sample Code

HDI Plug-in Version for SAP HANA 2.0 SPS 00 Revision 0

```
"plugin_version": "2.0.0.0"
```

Note

The three-level plug-in format used in SAP HANA 1.0 (for example, "`11.1.0`") is still valid; this old, three-level version format is converted internally to the appropriate four-level format required for newer versions of SAP HANA revisions.

In SAP HANA 1.0 there was no top-level (global) `"plugin_version"` entry, and the bottom-level (local) `"plugin_version"` entries were mandatory. In SAP HANA 2.0, the version of **all** plug-ins shipped with SAP HANA is the same as (and equal to) the SAP HANA version. For this reason, there should normally be no need for the local `"plugin_version"` entries since the result of the compatibility check would depend only on the largest entry in use. This means that it is sufficient to provide the top-level (global) entry or no entry at all if no particular minimum SAP HANA version is required. However, if present, bottom-level (local) entries override the top-level (global) entry or the default `"plugin_version"` of 0.0.0.0, respectively.

Related Information

Set up an SAP HDI Container for XS Advanced Applications [page 237]
SAP HDI Artifact Types and Build Plug-ins Reference

5.1.3 Define Run-Time Name-Space Rules for SAP HDI

Define rules for run-time name spaces in SAP HANA Deployment Infrastructure (HDI) containers.

Prerequisites

For the definition and deployment of name space rules for run-time container objects in SAP HDI, the following prerequisites apply:

- The design-time folder hierarchy contains application artifacts
- The design-time containers exist
- The corresponding run-time containers exist

Context

In SAP HANA Deployment Infrastructure (HDI), the application objects deployed to an HDI container must also be assigned to a run-time name space, which is defined in the `.hdinamespace` file.

> → Tip
>
> In the following examples, use **either** the `xs` command (on-premise environments) **or** the `cf` command (SAP Cloud Foundry).

Procedure

1. Create the name-space configuration file (`.hdinamespace`) in the design-time location of your choice.

   Design-time database objects are typically located in the `db/` folder of the application's design-time hierarchy, as illustrated in the following example:

   Output Code

```
app1-hello-world
|- db
   \- src
      |- .hdiconfig                      # HDI build-plugin configuration
      |- .hdinamespace                   # Run-time name-space
configuration
      |- myCDSdoc.hdbcds                 # CDS document
      |- myTable.hdbtable                # DB table definition
      |- mySQLView.hdbview               # SQL view definition
      |- mySynonym.hdbsynonym            # DB synonym definition
      |- mySynoConfig.hdbsynonymconfig   # DB synonym configuration file
      |- myProcedure.hdbprocedure        # DB procedure definition
      \- myImportData.hdbtabledata       # Data-import definition
```

> **Note**
>
> If you are using the SAP HDI API to create content, you first use the `WRITE` API to create the `.hdinamespace` in the HDI container's "work" file system and then deploy the name-space configuration to the "deployed" file system of the same HDI container, for example, with the `MAKE` command as described below.

2. Configure the run-time name spaces for the objects added to the run-time container during application deployment.

   In SAP HANA, name-space rules are defined in one or more file resources named `.hdinamespace`, which must be located in the design-time folder to which the naming rules apply - or the root folder of a hierarchy to which the naming rules apply. The content of the `.hdinamespace` file is specified according to the JSON format with the following structure:

   > **Sample Code**
   >
   > `src/.hdinamespace`
   >
   > ```
   > {
   >    "name"      : "com.sap.hana.example",
   >    "subfolder" : "<[append | ignore]>"
   > }
   > ```

   > **Tip**
   >
   > Change the path defined in `name` and, for `subfolder`, choose between `append` (add sub-folder name to object name space) and `ignore` (do **not** add sub-folder name to the object name space).

3. Deploy the name-space configuration to the container where the rules apply.

   The deployment operation pushes the specified application's database objects to the HDI container to which it is bound.

   - Command-line interface:
     `[xs | cf] deploy`
   - SAP Web IDE:
     Build > *Build*

     > **Note**
     >
     > You can use the *Database Explorer* tool to view the contents of the XS application's run-time container.

   - HDI SQL API
     To deploy the newly created files with the HDI `MAKE` API, you must provide the file name in the `PATH` parameter.

     > **Restriction**
     >
     > This step requires the privileges of an HDI container administrator. Only files can be deployed; it is not possible to deploy folders.

Sample Code

Deploy Files to the Work File System of Container "C"

```
CREATE LOCAL TEMPORARY COLUMN TABLE #DEPLOY_PATHS LIKE
_SYS_DI.TT_FILESFOLDERS;
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('.hdinamespace');
CREATE LOCAL TEMPORARY COLUMN TABLE #UNDEPLOY_PATHS LIKE
_SYS_DI.TT_FILESFOLDERS;
CREATE LOCAL TEMPORARY COLUMN TABLE #PATH_PARAMETERS LIKE
_SYS_DI.TT_FILESFOLDERS_PARAMETERS;
CALL C#DI.MAKE(#DEPLOY_PATHS, #UNDEPLOY_PATHS, #PATH_PARAMETERS,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #DEPLOY_PATHS;
DROP TABLE #UNDEPLOY_PATHS;
DROP TABLE #PATH_PARAMETERS;
```

Related Information

The HDI Name-Space Configuration File [page 250]
The SAP HDI Name-space Configuration Syntax [page 251]

5.1.3.1 Run-Time Name Spaces in SAP HDI

SAP HDI defines a strict separation between the naming of run-time objects and the organization of design-time files.

In SAP HANA Deployment Infrastructure (HDI), there is no simple, one-to-one mapping between the file-system structure and the names of the resulting run-time objects. Instead, the names of run-time objects are determined by naming rules defined in HDI name-space files. If desired, the file-system structure can still be used to derive the run-time name space of an object. However, decoupling design-time file structure and run-time name-space names allows the file-system structure to reflect the organization of work, and it is easy to relocate design-time resources without changing run-time name spaces. Note that if design-time folder names do not contribute to run-time names, you avoid problems with limitations on name length.

In SAP HANA, name-space rules for run-time objects are defined in one or more files named `.hdinamespace`, which must be located in the folder to which the naming rules apply - or the root folder of a folder structure to which the naming rules apply. The name-space rules allow you to specify the following elements:

- A name-space prefix, for example, `com.acme.hana.example`
- The name of design-time folder(s) containing the deployed objects

Code Syntax

Run-Time Name Space with Appended Folder Name

```
com.acme.hana.example.<Design-Time_Folder_Name>::<Run-Time_Object_Name>
```

The following table illustrates how the rules specified in the name-space configuration file influence the way run-time objects are named. If the name-space rules specify that the name of the design-time sub-folder should be **appended** to the run-time name-space prefix, for example, `com.acme.hana.example`:

Folder Names Appended to the Run-Time Name Space

Design-time Resource Path	Name of Run-Time Object (append)
`/src/`	`com.acme.hana.example::<Object_Name>`
`/src/db`	`com.acme.hana.example.db::<Object_Name>`
`/src/data`	`com.acme.hana.example.data::<Object_Name>`

Related Information

The HDI Name-Space Configuration File [page 250]
The SAP HDI Name-space Configuration Syntax [page 251]

5.1.3.2 The HDI Name-Space Configuration File

The SAP HANA Deployment Infrastructure (HDI) uses a JSON resource to define naming rules for run-time objects.

In SAP HDI, name-space rules are defined in one or more file resource named `.hdinamespace`, which must be located in the folder to which the naming rules apply - or the root folder of a hierarchy to which the naming rules apply. The content of the `.hdinamespace` file is specified according to the JSON format with the following structure:

Sample Code

`src/.hdinamespace`

```
{
   "name"      : "com.sap.hana.example",
   "subfolder" : "<[append | ignore]>"
}
```

Every folder can contain its own name-space file. For a name-space file to take effect, it has to be deployed in the same way as any other design-time file. After deployment, a name-space file provides run-time name-space information for all files in the same folder.

The `.hdinamespace` file provides name-space information for all sub-folders where no additional name-space file is present. For those sub-folders without a name-space configuration file, the name space depends on the value of the `subfolder` key in the name-space-configuration file in the parent folder.

- `append`
  Add the name of the sub-folder to the run-time name space for the deployed objects

- `ignore`
  Do **not** add the name of the sub-folder to the run-time name space for the deployed objects

> **Note**
>
> Name spaces and the naming rules are configurable and do not directly relate to the folder structure of the design-time files.

Developers can place a name-space configuration file (`.hdinamespace`) in a design-time folder, which specifies how the generated database objects are named in the sub-hierarchy underneath this folder. For example, you can just use the plain design-time object name, or the object name with a common name-space prefix, or even the object name with a prefix that is a concatenation of a common name space and the design-time folder path. Choosing a folder-independent naming rule provides the freedom to change the design-time folder structure at a later point in time without causing incompatible changes to run-time names.

Related Information

Run-Time Name Spaces in SAP HDI [page 249]
The SAP HDI Name-space Configuration Syntax [page 251]
Define Run-Time Name-Space Rules for SAP HDI [page 247]

5.1.3.3 The SAP HDI Name-space Configuration Syntax

In SAP HANA Deployment Infrastructure (HDI), the contents of the `.hdinamespace` file are formatted with the JSON syntax

> **Sample Code**
>
> `src/.hdinamespace`

```
{
   "name"      : "com.sap.hana.example",
   "subfolder" : "<[append | ignore]>"
}
```

name

Use the `name` property in the `.hdinamespace` configuration file to specify the name of run-time space to map to the name of (and objects in) the file-system folder where the `.hdinamespace` file is located.

> **Sample Code**

```
"name" : "com.sap.hana.example",
```

subfolder

Use the `subfolder` key in the `.hdinamespace` configuration file to specify if the name of the file-system folder where the `.hdinamespace` file is located should be **appended** to the name of the corresponding run-time space or **ignored**.

Sample Code

```
{
"name" : "com.sap.hana.example",
"subfolder" : "<[append | ignore]>"
}
```

For those sub-folders without a name-space configuration file, the name space depends on the value of the `subfolder` key in the name-space configuration file in the parent folder, for example: `append` or `ignore`.

- `"subfolder" : "ignore"`
  The sub-folder inherits the name-space information from the parent folder. In this case, all sub-folders use the same run-time name space.

  Design-Time Folder Names Ignored in Run-Time Name Space

Design-time Resource Path	Name of Run-time Object (ignore)
`/src/`	`com.sap.hana.example::<Object_Name>`
`/src/db`	`com.sap.hana.example::<Object_Name>`
`/src/db/proc`	`com.sap.hana.example::<Object_Name>`
`/src/data`	`com.sap.hana.example::<Object_Name>`
`/src/data/import`	`com.sap.hana.example::<Object_Name>`

- `"subfolder" : "append"`
  The run-time name-space for a sub-folder is derived by appending the sub-folder name to the name-space name of the parent folder.

  Design-Time Folder Names Appended to Run-time Name Space

Design-time Resource Path	Name of Run-Time Object (append)
`/src/`	`com.sap.hana.example::<Object_Name>`
`/src/db`	`com.sap.hana.example.db::<Object_Name>`
`/src/db/proc`	`com.sap.hana.example.db.proc::<Object_Name>`
`/src/data`	`com.sap.hana.example.data::<Object_Name>`
`/src/data/import`	`com.sap.hana.example.data.import::<Object_Name>`

Related Information

Run-Time Name Spaces in SAP HDI [page 249]
The HDI Name-Space Configuration File [page 250]
Define Run-Time Name-Space Rules for SAP HDI [page 247]

5.2 Configuring the HDI Deployer

Learn how to set up and use the Node.js-based SAP HANA Deployment Infrastructure (HDI) Deployer for XS advanced.

The HDI Deployer is a Node.js application (`@sap/hdi-deploy`) that is included in an XS advanced Multi-Target Application (MTA) and is used to deploy HDI design-time artifacts to the respective HDI containers. When an MTA is deployed, the HDI Deployer application is pushed first so that it can "prepare" the SAP HANA persistence; by the time defined services are started in XS Advanced, the HDI container is ready for use.

In the XS advanced HDI setup, each module of the MTA (for example `db/`, `js/`, or `java/`) is assigned its own technical database user for accessing the run-time schema of the HDI container.

The HDI Deployer is packaged into the `db/` module of the MTA. Consequently, in order to use a new HDI Deployer, you need to reference a new version of the HDI Deployer in the `db/` module's `package.json` file. The HDI Deployer supports HANA 1.0 SPS11, HANA 1.0 SPS12, and the current HANA 2.0 development codeline.

> **i** Note
>
> The HDI Deployer assumes ownership of the `src/`, `cfg/`, and `lib/` folders in the bound target HDI container. It is not possible to bind more than one (1) instance of the HDI Deployer to the same HDI container as the target container. For example, you cannot bind the `db/` modules of two different MTAs to the same HDI container as the target container; this results in undefined behavior.

Related Information

Integrate the HDI Deployer into an XS Advanced MTA Database Module [page 254]
Download and Consume SAP Node.js Packages [page 633]
Managing SAP HDI Containers and Artifacts [page 234]

5.2.1 Integrate the HDI Deployer into an XS Advanced MTA Database Module

Install the HDI Deployer for use by an XS advanced Multi-Target Application.

Prerequisites

The following prerequisite apply for the installation and use of the SAP HANA HDI Deployer for XS advanced:

- Node.js package manager (`npm`)
  Ensure that `npm` is configured appropriately, for example, with a working URL to the package registry

Context

The HDI Deployer is an application that is included in a Multi-Target Application (MTA) and is used to deploy HDI design-time artifacts to the respective HDI containers. When an MTA is deployed, the HDI Deployer application is pushed first so that it can “prepare” the SAP HANA persistence; by the time the services defined in the MTA's deployment descriptor are started in XS advanced, the HDI container is ready for use.

Procedure

1. Ensure that `npm` is configured appropriately.

   The following example indicates some of the configuration parameters that might be required (not all indicated parameters are mandatory):

   ```
   npm config set registry "http://npm.acme.com:8081/registry/content/groups/
   build.milestones.npm/"
   npm config rm proxy
   npm config rm https-proxy
   npm config set strict-ssl false
   ```

   → Tip

   For more information about how to include the public NPM Registry (`https://registry.npmjs.org`) as the upstream NPM registry, see *Related Information* below.

2. Install the HDI Deployer for use by the XS advanced application's database module (`db/`).

You can install the HDI Deployer by including the Node.js application "`@sap/hdi-deploy`" in the `package.json` file that defines the dependencies inside your XS advanced application's `db/` module, as illustrated in the following example:

Code Syntax

`db/package.json`

```
{
  "name": "deploy",
  "dependencies": {
    "@sap/hdi-deploy": "3.11.6"
  },
  "scripts": {
    "start": "node node_modules/@sap/hdi-deploy/"
  }
}
```

Related Information

Configuring the HDI Deployer [page 253]
Download and Consume SAP Node.js Packages [page 633]
The NPM Registry [page 687]

5.2.1.1 The XS Advanced Database Module's File Structure

The HDI Deployer expects the following file system structure for your the HDI content in your `db/` module:

- `src/`
  A folder containing the HDI source artifacts required by the XS advanced application
- `cfg/`
  An optional folder containing HDI configuration artifacts
- `package.json`
  The configuration file used by the Node.js package manager (`npm`) to bootstrap and start the XS advanced application

> **Note**
>
> All other files in the root directory of the database module (`db/`) are ignored by the HDI Deployer (`@sap/hdi-deploy`).

Related Information

Integrate the HDI Deployer into an XS Advanced MTA Database Module [page 254]

Configuring the HDI Deployer [page 253]
Managing SAP HDI Containers and Artifacts [page 234]

5.2.1.2 Database Connection Details and Options

Configure the HDI deployer's connection to the database.

The HDI Deployer can find details of service bindings by consulting the standard `<VCAP_SERVICES>` environment variable used in Cloud Foundry and XS advanced. The `<VCAP_SERVICES>` environment variable contains details of the database connection, for example: host name, port number, logon credentials, certificates, and so on, as illustrated in the following example. For the purposes of mutual authentication, the properties `client_authentication_private_key` and `client_authentication_certificate` can be supplied in the service binding.

For service bindings based on the SAP HANA Service Broker in XS Advanced or Cloud Foundry, the details of the bound service usually look something like the following example:

Sample Code

```
{
  "name" : "foo",
  "label" : "hana",
  "tags" : [ "hana", "database", "relational" ],
  "plan" : "hdi-shared",
  "credentials" : {
    "schema" : "FOO",
    "user" : "FOO_345999596729_RT",
    "password" : "<password>",
    "hdi_user" : "FOO_645927945801_DT",
    "hdi_password" : "<password>",
    "host" : "srv1234567.host.name",
    "port" : "30115",
    "db_hosts" : [ {
      "port" : 30115,
      "host" : "srv1234567.host.name" } ],
    "url" : "jdbc:sap://srv1234567.host.name:30115/?currentschema=FOO",
    "driver" : "com.sap.db.jdbc.Driver"
    }
}
```

In the example above, the credentials section contains all the data needed by the HDI Deployer to establish a connection to the database. The HDI Deployer uses the `hdi_user` and `hdi_password` credentials from a direct service binding. The `hdi_user` should have only those privileges required to perform the deployment. In most cases, access to a container FOO's API in the `FOO#DI` schema is sufficient.

For local testing, the HDI Deployer supports default configurations defined in the following configuration files:

- `default-env.json`
  A JSON file that defines a set of environment variables and corresponding values
- `default-services.json`
  A JSON file that specifies a set of service bindings

Sharing Passwords Between Services

The `password` and `hdi_password` properties can be specified as a combination of passwords from other bound services, as illustrated in the following example of a service binding:

Sample Code

```
{
  "hana" : [],
  "user-provided" : [
    {
    "name" : "split_password_service",
    "label" : "user-provided",
    "tags" : [],
    "credentials" : {
      "user" : "user",
      "schema": "schema",
      "password": ["password_and_hdi_password_service",
"password_only_service"],
      "hdi_password": ["password_and_hdi_password_service",
"hdi_password_only_service"],
      "tags" : [ "hana" ]
    }
  },
  {
    "name" : "password_and_hdi_password_service",
    "label" : "user-provided",
    "tags" : [],
    "credentials" : {
      "password" : "PASSWORD",
      "hdi_password": "HDI_PASSWORD",
      "tags" : [ "password" ]
    }
  },
  {
    "name" : "hdi_password_only_service",
    "label" : "user-provided",
    "tags" : [],
    "credentials" : {
      "hdi_password": "123",
      "tags" : [ "password" ]
    }
  },
  {
    "name" : "password_only_service",
    "label" : "user-provided",
    "tags" : [],
    "credentials" : {
      "password" : "456",
      "tags" : [ "password" ]
    }
  } ]
}
```

When the service `split_password_service` is used, the services specified in `password` and (or) `hdi_password` are checked and the respective `password` and (or) `hdi_password` are concatenated. The services are accessed in the order they are defined. The resulting `split_password_service` has the password "`PASSWORD456`" and the `hdi_password` "`HDI_PASSWORD123`".

> **Note**
>
> You can specify 0 (zero) to n services. Specifying 0 services results in the `password` or `hdi_password` '<>', where '<>' is an empty string.

Related Information

The SAP HDI Deployer [page 258]

5.2.1.3 The SAP HDI Deployer

SAP HDI provides dedicated tools to enable the deployment of design-time database artifacts to the SAP HANA database.

For XS advanced and Cloud Foundry environments, the SAP HANA Deployment Infrastructure (HDI) includes a so-called "Deployer", which is a Node application that is included in a Multi-Target Application (MTA) for the purpose of deploying an XS advanced application's HDI database-related **design-time** artifacts to the application's corresponding HDI containers. When an MTA is deployed to the target run-time platform, for example, XS advanced or SAP Cloud Platform @ Cloud Foundry, the HDI Deployer application `@sap/hdi-deploy` is deployed first so that it can "prepare" the SAP HANA persistence; by the time the services defined in the MTA's deployment descriptor are started in the XS advanced (or Cloud Foundry) run-time environment, the HDI container is ready for use.

You can install the HDI Deployer by including the Node.js application "`@sap/hdi-deploy`" in the `package.json` file that defines the dependencies inside your XS advanced application's `db/` module, as illustrated in the following example:

> **Code Syntax**
>
> `myApp/db/package.json`

```
{
  "name": "deploy",
  "dependencies": {
    "@sap/hdi-deploy": "3.11.6"
  },
  "scripts": {
    "start": "node node_modules/@sap/hdi-deploy/"
  }
}
```

> **Note**
>
> Java applications also require the HDI Deployer `@sap/hdi-deploy` for the deployment to HDI of database-related design-time artifacts. However, after deployment Java applications typically connect to the run-time schema of the application's corresponding HDI container using JDBC. Access to the HDI run-time containers is also possible with the SAP SQL APIs for SAP HANA DI (HDI).

The approach used for the HDI deployer is also applied with other backing services in XS advanced. For example, the SDS deployer takes care of the deployment of Streaming Analytics artifacts, and the LRep deployer takes care of deploying FLP content to the Layered Repository. The common approach is that dedicated XS advanced applications initialize backing service instances for MTAs and are coordinated by the deployment process as defined in the MTA deployment descriptor.

In addition to the stand-alone deployer application, HDI also provides an API that enables applications to create database objects at run time by generating design-time artifacts within the application and deploying them into their container with the HDI. In this way, it is possible to extend the persistence in run-time authoring scenarios, for example, field extensibility or dynamic rule generation.

Dynamic Deployment to HDI

The Node.js module `@sap/hdi-dynamic-deploy` provides a Node.js-based HTTP server that can be used to deploy database content to dynamically created HDI containers, for example, containers created by means of the XS advanced Instance Manager. The HDI Dynamic Deployer is intended for use in not only in XS Advanced on-premise scenarios but also in the SAP Cloud Platform Cloud Foundry (CF) environment.

> **i** Note
>
> `@sap/hdi-dynamic-deploy` is based on `@sap/hdi-deploy` which should be used if a static deployment at deployment time is sufficient.

Typically, `@sap/hdi-dynamic-deploy` is installed by including a dependency in the `package.json` file of your XS advanced application's database module, as illustrated in the following example:

> Sample Code
>
> `myApp/db/package.json`

```
{
  "name": "deploy",
  "dependencies": {
     "@sap/hdi-dynamic-deploy": "1.5.9"
  },
  "scripts": {
     "start": "node node_modules/@sap/hdi-dynamic-deploy/"
  }
}
```

For more detailed information about using the `@sap/hdi-dynamic-deploy`, see the `README.md` file included in the application's corresponding NPM package.

Run-time Access to HDI Containers

If XS advanced applications need access to their HDI run-time containers outside the scope of the HDI deployer, they can use the SAP SQL APIs for SAP HANA DI, which include an API specifically intended for content development in HDI containers.

> **Note**
>
> When working with the SAP SQL API for HDI, the HDI Deployer "`@sap/hdi-deploy`" described above is **not** required.

For example, for Node.js applications that need to create database objects at run time, the Node.js module `@sap/hdi` provides an HDI client library that includes a selection of asynchronous methods which enable access to SAP HANA deployment infrastructure functionality, for example: `connect()` and `disconnect()`, `createContainer()` and `dropContainer()`, and `configureLibraries()`.

Similarly, the `sap-java-hdi` client library enables access to the SAP HANA Deployment Infrastructure (HDI) for Java applications that need to create database objects at run time. Version 2.0 of the SAP HANA DI (HDI) client library for Java applications wraps the HDI SQL APIs in Java classes and methods which enable access to the APIs that are provided for managing HDI, its containers, and any container groups. Assuming the `hdiUser` has the privileges required to access `_SYS_DI`, the `sap-java-hdi` client library can be used to connect to the SAP HANA database, configure HDI, create and drop HDI containers, configure container build plug-in libraries, and so on.

> **Note**
>
> For more information about the standard and additional Java and Node.js libraries that you can use in your XS advanced applications, see *Related Information* below.

Related Information

Deployment Options in HDI [page 260]
Using Ignore Lists to Prevent Deployment [page 266]
Standard Node.js Packages for XS Advanced [page 635]
Standard Java Client Libraries for XS Advanced [page 720]

5.2.1.4 Deployment Options in HDI

A list of options for configuring custom deployment scenarios with SAP HDI.

Usage

The following example shows how to provide interactive options for the Node.js-based HDI deployer `@sap/hdi-deploy`:

> **Output Code**

```
node deploy --<option> [<component> [<component> [...]]]
```

→ Tip

Interactive deployment options can also be passed to `@sap/hdi-deploy` using the environment variable `HDI_DEPLOY_OPTIONS`, in JSON format, for example:

```
{ "verbose" : true, "exit" : true, "include_filter" : [ "src/", "cfg/" ] };
```

Deployment Options

The following table lists the options supported by the HDI deployer (`@sap/hdi-deploy`) for the configuration of interactive deployment scenarios, for example, for orchestration using the SAP WEB IDE or continuous integration (CI) scripts.

→ Tip

Options can also be passed to `@sap/hdi-deploy` by means of the `HDI_DEPLOY_OPTIONS` environment variable.

Option	Description
`--version`	Show the version of the currently installed HDI deployer and exit
`-t, --trace`	Enable tracing
`--use-hdb`	Enable the `hdb` client to connect to the database instead of `@sap/hana-client`; by default, the `@sap/hana-client` is used ⓘ Note Requires `@sap/hdi-deploy` 4.4.0.
`--hana-client-trace`	Enable tracing for the SAP HANA client. ⓘ Note Enabling this option means that **all** interactions with the SAP HANA server will be traced, which can lead to a large amount of trace information being written.
`--hana-client-packet-trace`	Enable `PACKET` tracing for the SAP HANA client. ⓘ Note This option must only be used in combination with `--hana-client-trace`.
`--[no-]verbose`	[Do not] print detailed log messages to the console

Option	Description
`--structured-log <file>`	Write log messages as JSON objects into the given file. If the log file already exists, messages are appended.
`--[no-]exit`	[Do not] exit after deployment of artifacts
`--[no-]lock-container`	[Do not] acquire the container lock while working with the container
`--root <path>`	Use the given root path for artifacts
`--working-set [<path> ...]`	Define the given paths (directories and files) as the working set; a non-default working set applies additional restrictions, for example, other options might be disallowed
`--include-filter [<path> ...]`	Only include the given paths (directories and files) during delta detection
`--deploy [<file> ...]`	Explicitly schedule the given files for deployment and extends the include-filter for collecting local files. Instead of a real path, a path pattern like `src/**/*.hdbtable` can be used as well.
`--[no-]treat-unmodified-as-modified`	[Do not] treat unmodified files during delta detection as modified files
`--undeploy [<file> ...]`	Explicitly schedule the given files for undeployment
`--parameter [<key>=<value>> ...]`	Pass the given list of key-value parameters to the deployment
`--path-parameter [<path>:<key>=<value> ...]`	Pass the given list of **path** key-value parameters to the deployment
`--[no-]auto-undeploy`	[Do not] undeploy artifacts automatically based on delta detection and ignore the `undeploy.json` file
`--[no-]treat-warnings-as-errors`	[Do not] treat warnings as errors
`--[no-]simulate-make`	[Do not] simulate the `make`, and skip post-`make` activities; pre-`make` activities still take effect, for example, grants
`--connection-timeout <ms>`	The number of milliseconds to wait for the database connection(s) to succeed
`--delete-timeout <ms>`	The number of milliseconds to wait for the `DELETE` call
`--write-timeout <ms>`	The number of milliseconds to wait for the `WRITE` call
`--lock-container-timeout <ms>`	The number of milliseconds to wait for the container lock to be released

Option	Description
`--exclude-filter [<path> ...]`	Exclude the given paths during: file walk, delta detection and when explicitly scheduled using `--(un)deploy` → Tip The exclude filter works like (and can be used either in place of or in combination with) the `.hdiignore` file, whose content is similar in structure to the `.gitignore` file. Like the `undeploy.json` file, the `.hdiignore` file must be located in the root folder of the project.
`--[no]-optimise-file-upload`	[Do not] perform delta detection by means of local SHA256 calculation instead of `DELETE` and `WRITE` calls. Will not have any positive effect when used along with `--treat-unmodified-as-modified`. ℹ Note Requires `@sap/hdi-deploy` 4.5.0.
`--[no-]treat-wrong-ownership-as-errors`	[Do not] treat wrong ownership of objects as errors ℹ Note Default: Not enabled.
`--[no-]migrationtable-development-mode`	[Do not] pass the development mode flag for migration tables to HDI, if the parameter is supported by the server ℹ Note Default: Not enabled.
`--[no-]liveness-ping`	[Do not] send a sign of life (availability) at a specified interval ℹ Note Default: Send a sign of availability
`--[no-]live-messages`	[Do not] display the `make` messages while the `make` operation is still in progress. ℹ Note Default: Display `make` messages during the `make` operation.

Environment Variables for HDI Configuration

You can use environment variables to configure the HDI Deployer, `@sap/hdi-deploy`, for example, for use by applications or for infrastructure or development tools such as the deploy service or the internal build tools used by the SAP Web IDE.

HDI Environment Variables for Applications

You can configure the Node.js module `@sap/hdi-deploy` to use the following environment variables which are exposed to applications, for example, in the the MTA development descriptor (`mta.yaml`) or Cloud Foundry manifest (`manifest.yml`):

- `TARGET_CONTAINER`:
  (optional) The service name that specifies the HDI target container. This name is required if more than one HDI service is bound to the HDI deployer.
- `SERVICE_REPLACEMENTS`:
  (optional) A list of service replacements structured as JSON, for example, `[ { "key": "logical-service-name-1", "service":"real-service-name-1"}, { "key": "logical-service-name-2", "service":"real-service-name-2"} ]`, where the logical service names refer to the names in the HDI content, and the real service names refer to the services which are bound to the HDI deployer in `VCAP_SERVICES`. Note that if the HDI content references a service name which is not listed in the replacements, then this name is used as a real service name.

The structure of the `SERVICE_REPLACEMENTS` environment variable is based on the MTA specification in order to enable MTA group assignments.

Sample Code

Example `manifest.yml` File

```
applications:
- name: app-db
  path: db
  health-check-type: process
  services:
    - app-database
    - real-grantor-service
    - real-external-service
  env:
    TARGET_CONTAINER: app-database
    SERVICE_REPLACEMENTS: >
    [
      {
        "key" : "logical-grantor-service",
        "service" : "real-grantor-service"
      },
      {
        "key" : "logical-external-service",
        "service" : "real-external-service"
      }
     ]
```

Environment Variables for HDI Infrastructure and Development Tools

The following variables can be used to configure the HDI deployment infrastructure and development tools such as the SAP Web IDE:

Variable	Description	Comments
`EXIT`	If set, the HDI Deployer exits when the deployment is completed. → Tip Using the `EXIT` environment variable is equivalent to passing the option `--exit` on the command line.	Optional
`DEPLOY_ID`	If set, the given deployment ID will be written to the final application log entry (custom ID, to support processes in parsing log output)	Optional

`HDI_DEPLOY_OPTIONS`	A JSON-structured set of options for the HDI Deployer, for example:
	
	`{ "auto_undeploy" : true,`
	`  "exit" : true,`
	`  "root" : "/volumes/A/workspaces/B/db/",`
	`  "include_filter" : [ "src/", "cfg/" ]`
	`};`
	
	Command line options can be translated to `HDI_DEPLOY_OPTIONS` options by replacing the "-s" in the option names with "_s". Options which can accept multiple values require a JSON array with the required values, for example, path options such as `include-filter`.
	
	ⓘ Note
	Options defined in `HDI_DEPLOY_OPTIONS` override any options that are passed on the command line.
		Optional

`APPLICATION_ID`	Used in conjunction with the `space_name` and the `organization_name` of the `<VCAP_APPLICATION>` to set the session variable `<APPLICATION>` for all connections to the database. ⓘ Note The `APPLICATION_ID` setting can only be used by applications from SAP.	Optional, fallback = `SAP_HDI`
`APPLICATION_VERSION_INFO`	Logged to the command line, to allow logging of some additional information about the application.	Optional

Dynamic Deployment

The standard way for XS advanced or Cloud Foundry to deploy HDI content at run time is by making use of the HDI dynamic deployer (`@sap/hdi-dynamic-deploy`) instead of using the HDI deployer (`@sap/hdi-deploy`) directly.

Based on `@sap/hdi-deploy`, the Node.js module `@sap/hdi-dynamic-deploy` provides an HTTP server that calls `@sap/hdi-deploy` when it receives a corresponding HTTP POST request and can be used to deploy database content to dynamically created HDI containers, for example, containers created by means of an Instance Manager. The HDI dynamic deployer is intended for use in both on-premise and Cloud scenarios.

> **i** Note
>
> For more information about the HDI dynamic deployer, see *Related Information* below.

Related Information

Configuring the HDI Deployer [page 253]
The SAP HDI Deployer [page 258]
@sap/hdi-dynamic-deploy (NPM Repository)
Standard Node.js Packages for XS Advanced [page 635]

5.2.1.5 Using Ignore Lists to Prevent Deployment

Specify a list of artifacts that should be ignored during deployment.

The HDI deployer allows you to define a list of artifacts that should be ignored for the purposes of deployment; the artifacts are specified in a file that must have the name `.hdiignore` and are not considered for any deployment operation in HDI.

> **i** Note
>
> The `.hdiignore` file must be placed at the root of the application project folder, just like the `undeploy.json` file.

The format of the `.hdiignore` file is similar to the structure of a `.gitignore` file; namely, lines of text specifying the relative path to any artifact you want to exclude from an HDI deployment operation. The syntax required in the `.hdiignore` file enables you to use either real paths or path patterns, as illustrated in the following example:

> **Code Syntax**
>
> The `.hdiignore` File

```
src/table_1.hdbtable
src/*_2.hdbtable
```

The `.hdiignore` file works in the same way as the HDI deployer's --exclude-filter option, and it is possible to use both features at the same time. For more information about the exclude filter as well as details of other deployment options, see *Deployment Options in HDI* in *Related Information* below.

Related Information

Deployment Options in HDI [page 260]

5.2.1.6 HDI Delta Deployment and Undeploy Allowlist

The HDI Deployer implements a delta-based deployment strategy including an optional allowlist.

On start up, the HDI Deployer recursively scans the local `src/` and `cfg/` folders, processes configuration templates, checks the HDI container on the server-side, and calculates the set of added, modified, and deleted files based on the difference between the local file system state and the state of the deployed file system of the server-side HDI container.

In normal operation, the HDI Deployer will schedule only the set of added and modified files for deployment.

The set of deleted files is not scheduled for automatic undeployment. To undeploy deleted files, an application must include an undeploy "allowlist", for example, by creating an `undeploy.json` file in the root directory of the `db/` module (alongside the `src/` and `cfg/` folders). The undeploy allowlist `undeploy.json` file is a JSON document with a top-level array of file names, as illustrated in the following example:

Code Syntax

Example `undeploy.json` File

```
[
    "src/Table.hdbcds",
    "src/Procedure.hdbprocedure"
]
```

The `undeploy.json` file must list all artifacts to be undeployed. In addition, the file path specified for the artifacts to remove must be **relative** to the root directory of the `db/` module and must use the HDI file path forward-slash delimiter (`/`).

For interactive scenarios, it is possible to pass the "auto-undeploy" option to the HDI Deployer, as illustrated in the following example:

```
node deploy --auto-undeploy
```

In this case, the HDI Deployer ignores the undeploy allowlist defined in the `undeploy.json` file and schedules all deleted files in the `src/` and `cfg/` folders for undeployment.

Related Information

Integrate the HDI Deployer into an XS Advanced MTA Database Module [page 254]
The XS Advanced Database Module's File Structure [page 255]

5.2.1.7 The Default Access Role for HDI Containers

An overview of the access role required for HDI containers.

When an HDI container service instance is created by the SAP HANA Service Broker, for example, service instance “foo” with schema name “FOO”, the broker creates an HDI container named “FOO” (consisting of the run-time schema “FOO” , the HDI metadata and API schema “FOO#DI” , and the object owner “FOO#OO”) and, in addition, the following roles, which are assigned to the application user:

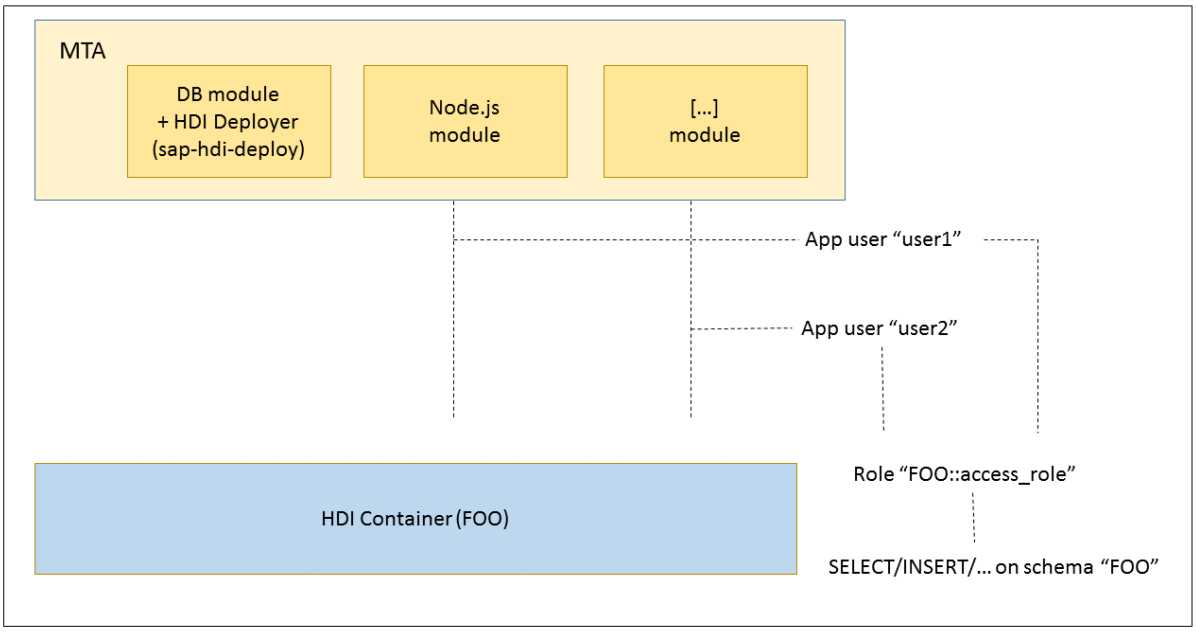
- `FOO::access_role`
  A global access role for the run-time schema. This access role is assigned a set of default permissions for the run-time schema: `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `EXECUTE`, `CREATE TEMPORARY TABLE`, and `SELECT CDS METADATA` on the run-time schema “FOO”.
- `FOO::external_privileges_role`
  A role that grants the application user the privileges required to enable access to schemas and objects outside the HDI container, for example, the run-time container “BAR”.

> **Note**
>
> The roles exist as long as the HDI container exists; they are not lost when the application binding user changes. New binding users are automatically assigned these roles by the broker.

Every time the service instance is bound to an application, the service broker creates two new users that are specific to this binding. The first user is the application user who is named user in the instance's credentials. This user is used by the application to access the HDI container's run-time schema “FOO”. This user is assigned the service instance's global access role “`FOO::access_role`” and the role “`FOO::external_privileges_role`”. The second user is the HDI API user - named “`hdi_user`” in the credentials. This user is equipped with privileges for the container's APIs in the “FOO#DI” schema.

The following diagram illustrates the binding-specific application users and the role of the global access role:

> **Note**
>
> For the sake of simplicity, the HDI API users and the bindings for the HDI Deployer are not shown in this diagram.

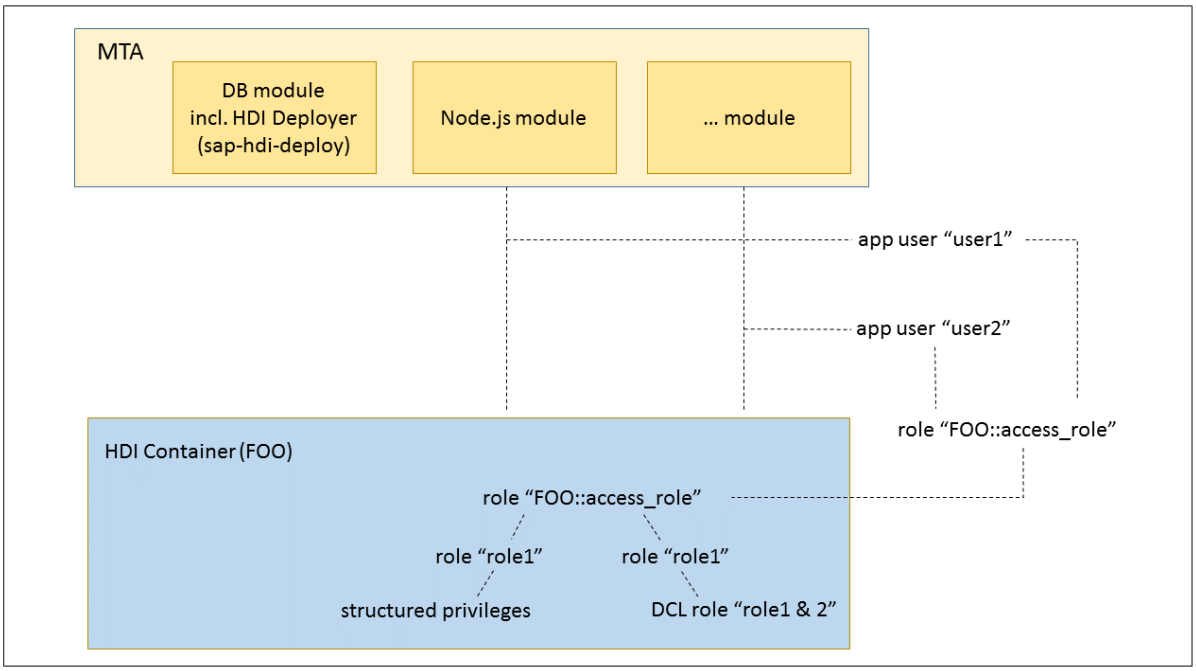


To assign roles from the HDI content to the application-binding-specific users (the users specified in `“user”:`), the HDI Deployer implements an automatic assignment of the “`default_access_role`” role if it is present in the deployed content.

If a role definition file exists at the path `src/defaults/default_access_role.hdbrole`, and this file defines a role named “`default_access_role`”, and this file is included in the deployment (for example, it is **not** excluded by means of an include filter), then the HDI Deployer grants the deployed “`default_access_role`” role to the service instance's global access role (for example, “`FOO::access_role`”). In addition, the HDI Deployer revokes all default permissions `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `EXECUTE`, `CREATE TEMPORARY TABLE`, and `SELECT CDS METADATA` on the run-time schema “FOO”) from the global access role.

> **i** Note
>
> If you use a `.hdinamespace` file in `src/` which defines a real name-space prefix for sub folders, then you need to put a `.hdinamespace` file with the empty name space “name”: '' at `src/defaults/` to ensure that the role can be named `default_access_role`.

The following diagram illustrates the binding-specific application users, the role of the global access role, and the container-specific default access role:



> **Note**
>
> The “`default_access_role`” is assumed to be an “umbrella” role which aggregates other roles.

The Development Debug Role

Similarly to the `default_access_role`, a `development_debug_role` can be used to grant additional privileges to the default access role.

> **Note**
>
> This development debug role is only intended for HDI development and debugging; it is not intended (or recommended) for use in a productive environment.

If a role-definition file exists at the path `src/defaults/development_debug_role.hdbrole`, and this file defines a role named `development_debug_role`, and this file is explicitly included in the deployment by means of the `--deploy` option, then the HDI Deployer grants the deployed `development_debug_role` role to the service instance's global access role (for example, `FOO::access_role`). In order to remove the privileges granted this way, the design-time, role-definition file has to be undeployed.

> **Note**
>
> For more information about defining design-time roles, for example, in `.hdbrole` artifacts, see *HDI Artifact Types and Build Plug-ins* in *Related Information* below.

Related Information

Configuring the HDI Deployer [page 253]
Integrate the HDI Deployer into an XS Advanced MTA Database Module [page 254]
The XS Advanced Database Module's File Structure [page 255]
HDI Delta Deployment and Undeploy Allowlist [page 267]
HDI Artifact Types and Build Plug-ins [page 1052]

5.2.1.8 Reusable Database Modules in XS Advanced Applications

Setup reusable components inside an application's own data persistence model.

To enable an application to use parts of the database persistence of a reusable component inside its own persistence model, the HDI Deployer allows you to automate the linking (include) of design-time files of reusable components in a consuming application. This automated mechanism is based on the Node.js package management mechanism for defining, publishing, and consuming reusable database modules, which also supports versioning based on the semantic versioning concepts.

A reusable database module is considered to have the same `src/` and `cfg/` folder structure as a normal database module. The `src/.hdiconfig` file is mandatory and used by the module mechanism as an indicator that the `src/` and `cfg/` folders belong to a consumable, reusable database module. In addition, the reusable database module needs to have a `package.json` file which defines the module's name, the module's version, and so on.

The following example illustrates the structure of a complete reusable database module:

```
/
+-- src/
|   +-- .hdiconfig
|   +-- <source files ...>
+-- cfg/
|   +-- <optional configuration files ...>
+-- package.json
```

The `package.json` file contains the module's name, description, version, repository URL, and the set of files which belong to the module, as illustrated in the following example:

Code Syntax

`package.json`

```
{
  "name": "module1",
  "description": "A set of reusable database objects",
  "version": "1.3.1",
  "repository": {
    "url": "git@github.com:modules/module1.git"
   },
  "files": [
    "src",
    "cfg",
    "package.json"
```

```
    ]
}
```

Consumption of a reusable database module is done by adding a dependency in the consuming module's `package.json` file, for example, alongside the dependency to the HDI Deployer "`@sap/hdi-deploy`" as illustrated below, where the consuming module requires "`module1`" in version "`1.3.1`" and "`module2`" in version "`1.7.0`":

Code Syntax

```
{
 "name": "deploy",
 "dependencies": {
   "@sap/hdi-deploy": "3.7.0",
   "module1": "1.3.1",
   "module2": "1.7.0"
 },
 "scripts": {
   "start": "node node_modules/@sap/hdi-deploy/"
 }
}
```

When running `npm install` to download and install the dependencies which are listed in the dependencies section of the `package.json` file, `npm` will also download the reusable database modules and places them into the `node_modules/` folder of the consuming module. For each module a separate subfolder is created with the name of the module.

When the HDI Deployer is triggered to do the actual deployment of the (consuming) database module, it scans the `node_modules/` folder and virtually integrates the `src/` and `cfg/` folders of reusable database modules it finds into the (consuming) database module's `lib/` folder. Reusable database modules are identified by the mandatory `src/.hdiconfig` file. On successful deployment, the HDI container will contain the consumed modules below the root-level `lib/` folder, as shown in the following example:

```
/
+-- src/
+-- cfg/
+-- lib/
|   +-- module1/
|   |   +-- src/
|   |   +-- cfg/
|   +-- module2/
|       +-- src/
|       +-- cfg/
```

Note

It is not permitted to recursively include reusable database modules.

Related Information

Configuring the HDI Deployer [page 253]
Integrate the HDI Deployer into an XS Advanced MTA Database Module [page 254]

5.2.1.9 Templates for HDI Configuration Files

The HDI Deployer implements a template mechanism for HDI configuration files.

The HDI Deployer enables you to set up a template mechanism for HDI configuration files, for example, configuration files for synonyms or projection views based on services which are bound to the `db/` module application. With this template mechanism, it is possible to configure synonyms, projection views, and similar artifacts. to point to the right database schema without knowing the schema name at development time.

On startup, the HDI Deployer recursively scans the local `cfg/` folder and picks up all files with a `.*config` suffix, for example, all `.hdbsynonymconfig`, `.hdbvirtualtableconfig` files. For all collected files which contain `.configure` markers in their content, the HDI Deployer applies the configuration template mechanism and creates transient configuration files which are then deployed to the HDI container as described in the following examples.

For a synonym configuration file `cfg/LOCAL_TARGET.hdbsynonymconfig`:

Code Syntax

```
{
 "LOCAL_TARGET" : {
   "target" : {
      "schema.configure" : "logical-external-service/schema",
      "database.configure" : "logical-external-service/database",
      "object" : "TARGET"
    }
  }
}
```

The following section:

Code Syntax

```
"schema.configure" : "logical-external-service/schema",
"database.configure" : "logical-external-service/database",
"object" : "TARGET"
```

is transformed by the template mechanism into:

Code Syntax

```
"schema" : "THE_SCHEMA",
"database" : "THE_DATABASE",
"object" : "TARGET"
```

where “`THE_SCHEMA`” and “`THE_DATABASE`” are the values for the schema and database fields of the bound service “`logical-external-service`” , which are denoted by the path expressions “`logical-external-service/schema`” and “`logical-external-service/database`”.

If a field in the service is missing, it will not be configured and will be removed instead, for example, “`database`” in the code examples above might be optional. The names of the services are subject to the service replacements mechanism, which can be used to map a real service, for example, “`real-external-service`”, to a logical service name which is used in the configuration files, for example, “`logical-external-service`”. It is not always applicable to use “`schema.configure`”, “`database.configure`”, in the configuration template files. For this reason, the HDI Deployer provides a generic way of copying

a set of properties from the bound service, for example, schema, database, remote source, etc. if they are present, although the template file doesn't mention them. For the configuration file `cfg/LOCAL_TARGET.hdbsynonymconfig`, this could looks as follows:

Code Syntax

`cfg/LOCAL_TARGET.hdbsynonymconfig`

```
{ "LOCAL_TARGET" : { "target" : { "*.configure" : "logical-external-service",
"object" : "TARGET" } } }
```

When the HDI Deployer encounters a `*.configure` entry, it copies all well-known fields which are present in the bound service into the configuration file. The well-known fields are currently "`remote`", "`database`", and "`schema`".

The HDI Deployer also supports old-style `.hdbsynonymtemplate` template files. If a `.hdbsynonymtemplate` file is found in the `cfg/` or `src/` folder, it is processed as a configuration template file and a transient file with the suffix `.hdbsynonymconfig` is created. A field grantor is replaced with the schema value from the referenced service; in a configuration template file, a `"grantor"` field is equivalent to a `"schema.configure" : "the-service/schema"` entry.

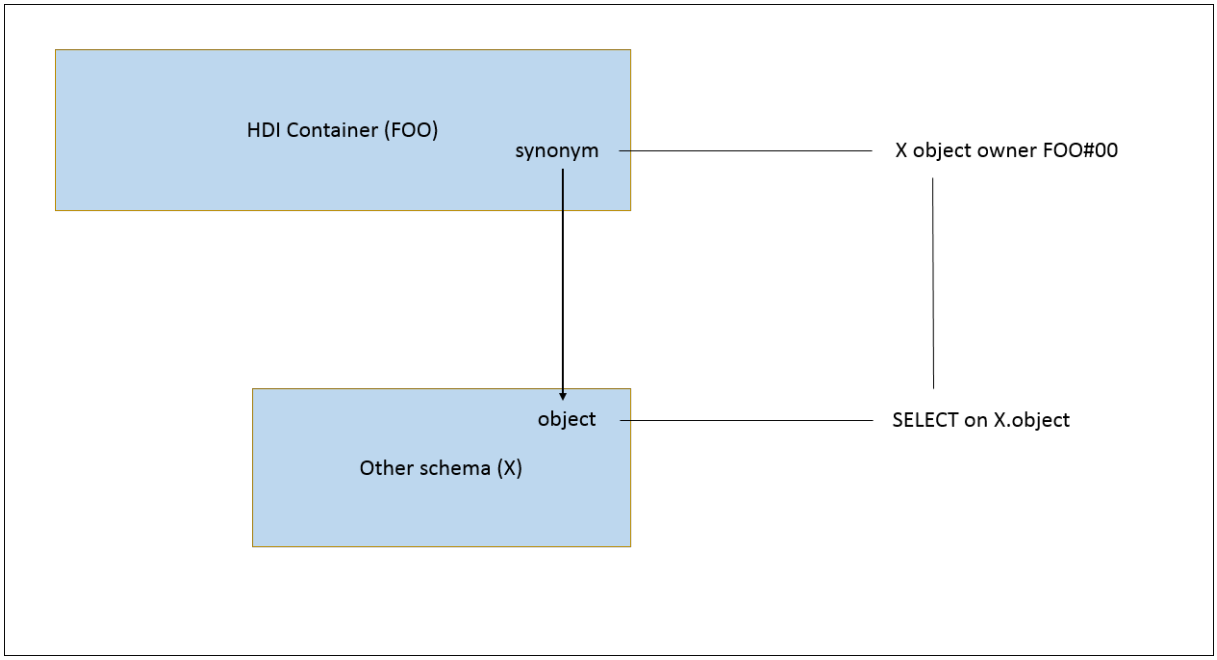
Related Information

Configuring the HDI Deployer [page 253]
Integrate the HDI Deployer into an XS Advanced MTA Database Module [page 254]
Logical Schemas (SAP HANA HDI Reference)
Projection Views (SAP HANA HDI Reference)
Synonyms (SAP HANA HDI Reference)

5.2.1.10 Permissions for Objects in HDI-Container

The owner of a container object needs additional privileges to the ones assigned by default.

By default, an HDI container is assigned very few database privileges. For example, the object owner ("#OO" user) is only assigned the `CREATE ANY` privilege on the container's run-time schema (schema "FOO" for an HDI container "FOO"). To access database objects inside other database schemata or other HDI containers, and to be able to deploy synonyms into the HDI container which point to objects outside the container, the object owner needs additional privileges. For example, for an object "`object`" in schema "X", the `SELECT` privilege on "`X.object`" :



The hdbgrants file

To assign privileges automatically to the object owner and (or) the application binding users, the HDI Deployer provides `.hdbgrants` files, which use a syntax that is similar to the `.hdbrole` artifact. The following example shows the structure of an `.hdbgrants` file:

Code Syntax

`granting-service.hdbgrants`

```
{
  "granting-service": {
    "object_owner": {
      <privileges>
    },
    "application_user": {
      <privileges>
    }
  }
}
```

The top-level keys define the grantors; the names of the bound services which **grant** the privileges, `“granting-service”` in the code example above. The next level defines the users to whom the privileges are granted, these are the grantees: `“object_owner”` is used to specify the HDI container's object owner, and `“application_user”` defines the application users who are bound to the application modules, for example, the Node.js module in an MTA. The third level defines the set of privileges to grant using a structure that is similar to the format used in a `.hdbrole` role-definition file.

Note

For backwards compatibility, the suffix `.hdbsynonymgrantor` is also supported.

On startup, the HDI Deployer looks for files with the `.hdbgrants` suffix and processes them in the following way: For each grantor specified in the `.hdbgrants` file, the HDI Deployer looks up a bound service with the specified name (subject to service replacements), connects to the database with the service's credentials, and grants the specified privileges to the grantees.

Code Syntax

Example `cfg/SYS-access.hdbgrants` File with Some Privileges

```
{
  "SYS-access": {
    "object_owner": {
      "object_privileges":[
        {
          "schema": "SYS",
          "name": "VIEWS",
          "privileges": ["SELECT"]
        },
        {
          "schema": "SYS",
          "name": "TABLES",
          "privileges": ["SELECT"]
        }
      ]
    },
    "application_user": {
      "object_privileges":[
        {
          "schema": "SYS",
          "name": "VIEWS",
          "privileges": ["SELECT"]
        },
        {
          "schema": "SYS",
          "name": "TABLES",
          "privileges": ["SELECT"]
        }
      ]
    }
  }
}
```

The hdbrevokes File

Starting with version 3.8, the HDI deployer enables the assignment of revoking rights. Any privileges that can be granted by means of the `.hdbgrants` file can be revoked with the `.hdbrevokes` file. The `.hdbgrants` and the `.hdbrevokes` file share the same structure.

Note

If both files exist, the `.hdbrevokes` file is processed before the `.hdbgrants` file.

Code Syntax

`revoking-service.hdbrevokes`

```
{
```

```
  "revoking-service": {
    "object_owner": {
      <privileges>
    },
    "application_user": {
      <privileges>
    }
  }
}
```

Related Information

Configuring the HDI Deployer [page 253]
Integrate the HDI Deployer into an XS Advanced MTA Database Module [page 254]

5.3 Creating the Data Persistence Artifacts in XS Advanced

The data persistence model comprises the underlying database objects you want to use to store and provide data for your application, for example: tables, views, data types.

As part of the process of defining the data model for your XS advanced application, you create database design-time artifacts such as tables and views, or constraints and indexes. In XS advanced you can choose to use either of the following languages to define the artifacts in your data model:

- SQL Data Definition Language (DDL)
  Tables, views, constraints, indexes, etc.
- Core Data Services (CDS)
  Tables (entities), views, associations, user-defined types, etc.

Related Information

Creating Data Persistence Artifacts with SQL DDL in XS Advanced [page 284]
Creating Data Persistence Artifacts with CDS in XS Advanced [page 310]
HDI Design-Time Resources and Build Plug-ins [page 279]

5.3.1 Design-Time Database Resources in XS Advanced

Design-time database resources reside in the database module of your multi-target application.

The design-time representations of you database resources must be collected in the database module of your multi-target application; the database module is a folder structure that you create, for example `/db/` that

includes sub folder called `src/` in you place all the source files. For database applications, the structure of the design-time resources should look something like the following example:

Sample Code

Database module in an XS Advanced Application

```
<MyAppName>
|- db/                                  # Database deployment artifacts
|  |- package.json                      # Database details/dependencies
|  \- src/                              # Database artifacts...
|     |- .hdiconfig                     # HDI build plug-in configuration
|     |- .hdinamespace                  # HDI run-time name-space config...
|     |- myTable.hdbtable               # SDL DDL design-time view def.
|     |- myView.hdbview                 # SQL DDL design-time table def.
|     |- myIndex.hdbindex               # SDL DDL design-time index def.
|     |- myEntity.hdbcds                # HDB CDS design-time table def.
|     |- myDataType.hdbcds              # HDB CDS design-time type def.
|     |- myDoc.hdbcds                   # HDB CDS design-time data-model
|     |- data/                          # Data source files
|        \- myData.csv                  # Table-data import sources (CSV)
|     |- roles                          #
|        |- myPriv.hdbstructuredprivilege # SQL DDL privilege definition
|        \- myRole.hdbrole              # Role definition (JSON)
|- web/
|  |- xs-app.json
|  \- resources/
|- js/
|  |- start.js
|  |- package.json
|  \- src/
|- security/
|  \- xs-security.json
\- mtad.yaml
```

As illustrated in the example above, the `/db/` folder contains all your design-time database artifacts, for example: tables, views, procedures, sequences, calculation views. In addition to the design-time database artifacts, the database content must also include the following components:

- `package.json`
  A file containing details of dependencies
- `.hdiconfig`
  The HDI container configuration
- `.hdinamespace`
  The run-time name-space configuration for the deployed objects

The HDI deployment tools deploy all database artifacts located in the `db/` folder to an HDI design-time container. At the same time, the HDI tools create the corresponding run-time container and populate the container with the specified catalog objects. The deployed objects can be referenced and consumed by your business applications.

⚠ Caution

If the deployment tools establish that a database run-time artifact has no corresponding design-time definition, the run-time artifact (for example, a table) is dropped from the catalog.

Deployment Configuration Artifacts

The following files are mandatory for the database component of your deployment. One instance of each configuration artifact is required in the `db/src/` folder, and this instance applies to all sub-folders unless another instance exists in a sub folder:

- `/db/src/.hdiconfig`
  Used to bind the database artifacts you want to deploy (determined by a file suffix, for example, `.hdbcds`) with the appropriate build plug-in. For example, to deploy a CDS data model to an HDI container, you must configure the CDS build plug-in and, in addition, any other plug-ins required.

```
"hdbcds" : {
    "plugin_name" : "com.sap.hana.di.cds",
    "plugin_version" : "2.0.19.0" },
"hdbprocedure" : {
  "plugin_name"   : "com.sap.hana.di.procedure",
  "plugin_version": "2.0.20.0" },
"hdbtable" : {
  "plugin_name"   : "com.sap.hana.di.table",
  "plugin_version": "2.0.20.0" }
```

- `/db/src/.hdinamespace`
  Defines naming rules for the run-time objects which are created in the catalog when the application is deployed

```
{
  "name"      : "com.sap.hana.db.cds",
  "subfolder" : "append"
}
```

> **i** Note
>
> For a name-space and build-plugin configuration to take effect, the name-space and build-plugin configuration file must be deployed - in the same way as any other design-time file.

Related Information

Creating Data Persistence Artifacts with SQL DDL in XS Advanced [page 284]
Creating Data Persistence Artifacts with CDS in XS Advanced [page 310]
Defining the Data Model in XS Advanced [page 233]
SAP HDI Artifact Types and Build Plug-ins Reference

5.3.2 HDI Design-Time Resources and Build Plug-ins

In XS advanced, database design-time resources must be mapped to a build plug-in for deployment purposes.

In XS advanced, design-time artifacts are distinguished by means of a unique file suffix that must be mapped to an HDI build plug-in. The following example of an abbreviated HDI configuration file (`.hdiconfig`) file

illustrates how the design-time artifact types `.hdbcalculationview` and `.hdbcds` are mapped to their corresponding HDI build plug-in:

Code Syntax

`.hdiconfig`

```
"hdbtable" : {
   "plugin_name" : "com.sap.hana.di.table",
   "plugin_version": "2.0.0.0"
},
"hdbview" : {
   "plugin_name" : "com.sap.hana.di.view",
   "plugin_version": "2.0.0.0"
},
"hdbcalculationview" : {
   "plugin_name" : "com.sap.hana.di.calculationview",
   "plugin_version": "2.0.0.0"
},
"hdbcds" : {
   "plugin_name" : "com.sap.hana.di.cds",
   "plugin_version": "2.0.0.0"
}
```

The following table lists in alphabetical order the design-time artifacts you can develop and deploy with the SAP HANA Deployment Infrastructure (HDI) and describes the artifact's scope. The table also indicates which build plug-in is required to ensure successful deployment of the artifact. For more information about the individual artifact types and the configuration of the corresponding build plug-in, see *Related Links* below.

Default File-Suffix to HDI Build Plug-in Mappings

Artifact Suffix	Description/Content	Build Plug-in
`csv`	Source data for a table-import operation (also `hdbtabledata`)	`com.sap.hana.di.tabledata.source`
`.? (txt, copy only)`	An arbitrary design-time resource	`com.sap.hana.di.copyonly`
`hdbafllangprocedure`	Definition of a language procedure for an application function library (AFL)	`com.sap.hana.di.afllangprocedure`
`hdbanalyticprivilege`	Definition of an XML-based analytic privilege	`com.sap.hana.di.analyticprivilege`
`hdbcalculationview`	Definition of a calculation view	`com.sap.hana.di.calculationview`
`hdbcds`	A document that contains the specification of one or more database objects written with the SAP HANA Core Data Services syntax	`com.sap.hana.di.cds`
`hdbconstraint`	Transforms a design-time constraint into a constraint on database tables	`com.sap.hana.di.constraint`

Artifact Suffix	Description/Content	Build Plug-in
`hdbdropcreatetable`	Transforms a design-time table resource into a table database object	`com.sap.hana.di.dropcreatetable`
`hdbflowgraph`	Transforms a design-time flow-graph description into a corresponding set of database procedure or task objects	`com.sap.hana.di.flowgraph`
`hdbfulltextindex`	Full text index definition	`com.sap.hana.di.fulltextindex`
`hdbfunction`	Database function definition	`com.sap.hana.di.function`
`hdbgraphworkspace`	Definition of a graph work space resource	`com.sap.hana.di.graphworkspace`
`hdbindex`	Table index definition	`com.sap.hana.di.index`
`hdbmrjob`	Transform a design-time Hadoop map-reduce job resource into a virtual-function-package database object **i Note** The plug-in for the file types (`.hdbmrjob` or `.jar`) has been superseded by the Virtual Package plug-in (for `.hdbvirtualpackage*` artifacts). Use the `hdbvirtualpackage*` plug-ins for design-time Hadoop and SparkSQL JAR files, respectively.	`com.sap.hana.di.virtualfunctionpackage.hadoop`
`jar`	Optional mapping, if you want direct access to Hadoop files	`com.sap.hana.di.virtualfunctionpackage.hadoop`
`hdblibrary`	A design-time library resource	`com.sap.hana.di.library`
`hdblogicalschema`	Transforms a design-time logical-schema definition into database objects that can be consumed by synonyms etc.	`com.sap.hana.di.logicalschema`
`hdbprocedure`	Definition of a database procedure	`com.sap.hana.di.procedure`
`hdbprojectionview`	Definition of a projection view	`com.sap.hana.di.projectionview`

Artifact Suffix	Description/Content	Build Plug-in
`hdbprojectionviewconfig`	Configuration file for a projection view	`com.sap.hana.di.projectionview.config`
`hdbpublicsynonym`	Definition of a public database synonym	`com.sap.hana.di.publicsynonym`
`hdbreptask`	Transform a design-time replication task description into a corresponding set of database procedure or task operations	`com.sap.hana.di.reptask`
`hdbresultcache`	Definition of a result cache	`com.sap.hana.di.resultcache`
`hdbrole`	Definition of a database roles	`com.sap.hana.di.role`
`hdbroleconfig`	Configuration of database privileges (and other roles) to be included in a database role	`com.sap.hana.di.roleconfig`
`hdbsearchruleset`	Definition of search configurations for built-in search procedure	`com.sap.hana.di.searchruleset`
`hdbsequence`	Definition of a database sequence	`com.sap.hana.di.sequence`
`hdbsynonym`	Database synonym definition	`com.sap.hana.di.synonym`
`hdbsynonymconfig`	Configuration file for a database synonym	`com.sap.hana.di.synonym.config`
`hdbstatistics`	Statistics definition file	`com.sap.hana.di.statistics`
`hdbstructuredprivilege`	Definition of analytic or structured privileges	`com.sap.hana.di.structuredprivilege`
`hdbsysbicsynonym`	Create synonyms in the _SYS_BIC schema that refer to database objects located in the target schema of the current container	`com.sap.hana.di.sysbicsynonym`
`hdbsystemversioning`	Transforms a design-time, system-versioned table that refers to a current and history table into a system-versioned table database object	`com.sap.hana.di.systemversioning`
`hdbtable`	Table operations	`com.sap.hana.di.table`

Artifact Suffix	Description/Content	Build Plug-in
`hdbtabledata`	Definition of a data-import operation for a database table (also `csv`)	`com.sap.hana.di.tabledata`
`hdbtabletype`	Definition of a table type	`com.sap.hana.di.tabletype`
`hdbtextconfig`	Customization of the options used for text analysis	`com.sap.hana.di.tabletype`
`hdbtextdict`	Specification of the custom entity types and entity names to be used with text analysis	`com.sap.hana.di.textdictionary`
`hdbtextrule`	Specification of the rules (patterns) for extracting complex entities and relationships using text analysis	`com.sap.hana.di.textrule`
`hdbtextinclude`	Definition of the rules to be used in one or more extraction rule sets for top-level, text analysis	`com.sap.hana.di.textrule.include`
`hdbtextlexicon`	Definition of the lists of words used in one or more top-level text analysis rule sets	`com.sap.hana.di.textrule.lexicon`
`hdbtextminingconfig`	Customization of the features and options used for text mining	`com.sap.hana.di.textminingconfig`
`hdbtrigger`	Database trigger definition	`com.sap.hana.di.trigger`
`hdbview`	View definition file	`com.sap.hana.di.view`
`hdbvirtualfunction`	Definition of a virtual database function	`com.sap.hana.di.virtualfunction`
`hdbvirtualfunctionconfig`	Configuration file for a virtual function	`com.sap.hana.di.virtualfunction.config`
`hdbvirtualpackage`	Transform a design-time Hadoop Map Reduce Job or SparkSQL resource into a virtual-package database object	`com.sap.hana.di.virtualpackage.hadoop`
`hdbvirtualprocedure`	Definition of a virtual database procedure	`com.sap.hana.di.virtualprocedure`
`hdbvirtualprocedureconfig`	Configuration file for the virtual database procedure definition	`com.sap.hana.di.virtualprocedure.config`

Artifact Suffix	Description/Content	Build Plug-in
`hdbvirtualtable`	Definition of a virtual table	`com.sap.hana.di.virtualtable`
`hdbvirtualtableconfig`	Virtual table configuration file	`com.sap.hana.di.virtualtable.config`
`properties` `tags`	Properties file for a table-import operation	`com.sap.hana.di.tabledata.properties`

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference
The SAP HDI Container Configuration File [page 241]
SAP HDI Container Configuration File Syntax [page 243]
Creating Data Persistence Artifacts with SQL DDL in XS Advanced [page 284]

5.3.3 Creating Data Persistence Artifacts with SQL DDL in XS Advanced

You can use SQL DDL to define the underlying database objects that store and provide data for your application, for example, tables and views.

As part of the process of defining the data model for your application, you create database design-time artifacts such as tables and views, for example using the SQL Data Definition Language (DDL). Each artifact must be defined in one or more DDL documents. You can find some simple examples of the syntax required for specific data artifacts in the following sections:

- SQL DDL tables (`.hdbtable`)
- SQL DDL views (`.hdbview`)
- SQL DDL constraints (`.hdbconstraint`)
- SQL DDL indexes (`.hdbindex`)
- SAP HANA Database Roles (`.hdbrole`)
  Unlike the other design-time artifacts listed here, the SAP HANA database role is defined using the JavaScript Object Notation (JSON) syntax.
- SQL DDL Structured Privileges (`.hdbstructuredprivilege`)

> → Tip
>
> The tasks in this section create artifacts that, in some cases, are interdependent. For example, a table-data artifact is used to load initial data into two tables from which a view selects information that is visible only to users with the appropriate structured privilege. A role is also used to broaden access scope to other database users.

Related Information

Data Definition Statements (SQL Reference)
Create a Database Table with SQL Data Definition Language [page 285]
Create a Database View with SQL Data Definition Language [page 289]
Load Data into a Database Table with the "hdbtabledata" Artifact [page 293]
Create a Structured Privilege Using SQL Data Definition Language [page 297]
Create a Database Role [page 300]
Create a Database Constraint with SQL Data Definition Language [page 305]
Create an Index with SQL Data Definition Language [page 307]

5.3.3.1 Create a Database Table with SQL Data Definition Language

Define a design-time database **table** using the SQL Data Definition Language (DDL) syntax.

Prerequisites

To complete this task successfully, note the following prerequisites:

- You must have access to an SAP HANA system.
- For the purposes of this tutorial, you must have access to the SAP Web IDE for SAP HANA. Generally though, you can use any text editor to create a design-time artifact.
- You must have already created a development workspace and a multi-target application (MTA) project.
- You must already have created a database module for your MTA application project.
- You must already have set up an HDI container for the database artifacts

  > **i** Note
  >
  > A container setup file (`.hdiconfig`) is required to define which plug-ins to use to create the corresponding catalog objects from your design-time artifacts when the multi-target application (or just the database module) is deployed.

- To view run-time objects, you must have access to the SAP HANA Database Explorer tools that enable you to view the contents of the catalog.

Context

In the SAP HANA database, as in other relational databases, a table contains a set of data elements that are organized using columns and rows. SAP HANA Deployment Infrastructure (HDI) enables you to use the SQL DDL syntax to create a database table as a design-time file. Building or deploying the database module that contains the SQL DDL table creates the corresponding table in the corresponding schema.

During the build process, the `.hdbtable` plug-in is called to transform a design-time table resource into an SQL table database object. The file formats use a DDL-style syntax which is equivalent to the syntax required in the corresponding SAP HANA SQL statement `CREATE TABLE`, but without the leading "`CREATE`", as illustrated in the examples below. Specifying the table type is mandatory, and the type of table you can define is limited to ROW and COLUMN.

A table must be self-contained and cannot reference another table. As result, it is not possible to define a table by using another table (for example, `TABLE A LIKE B`), and you cannot define a table based on a query, for example, `TABLE A AS SELECT [...] FROM B`.

> **i** Note
>
> The `.hdbtable` plug-in used to create the catalog table object does not provide any data migration. An already deployed version of the table will be dropped on deployment.

To create a design-time SQL DDL table-definition file, perform the following steps:

Procedure

1. Start SAP HANA Web IDE for SAP HANA.

   The SAP Web IDE for SAP HANA is typically available at the following URL:

   `https://<HANA_HOST>:53075/`

   > → Tip
   >
   > To display the URL for the SAP Web IDE for SAP HANA, open a command shell, log on to the XS advanced run time using the `xs login` command, and run the following command:
   >
   > `xs app webide --urls`

2. Open the application project to which you want to add your SQL DDL table.
3. Create the table-definition file in SQL DDL.

   Browse to the folder in the database module in your application's project workspace, for example, `<MyApp1>/HDB/src` where you want to create the new SQL DDL table-definition file and perform the following steps:

   a. Right-click the folder where you want to save the SQL DDL table-definition file and choose *New* > *Database Artifact* in the context-sensitive pop-up menu.
   b. Enter the name of the table-definition file in the *File Name* box, for example, `PurchaseOrderHeader`.

      You can specify the corresponding file extension from the drop-down list in the *File Type* text box.

      > → Tip
      >
      > The file extension is mandatory; it is used to determine which plug-in to call when creating the corresponding run-time object during application build or deployment. SQL DDL role-definition artifacts have the file extension `.hdbtable`, for example, `PurchaseOrderItem.hdbtable`.

   c. Choose *Finish* to save the new SQL DDL table-definition file in the database module of your local project workspace.

4. Define the structure of the SQL DDL table (`PurchaseOrderHeader`).

   If the new table-definition file is not automatically displayed by the file-creation wizard, double-click the table-definition file you created in the previous step, for example, `MyTable.hdbtable`, and add the SQL DDL code that defines the table to the file:

   > **i Note**
   >
   > The following code example is provided for illustration purposes only.

   ```
   COLUMN TABLE "PurchaseOrder.Header" (
         "PURCHASEORDERID" INTEGER CS_INT GENERATED BY DEFAULT AS IDENTITY (NO
   CYCLE NO CACHE NO MINVALUE START WITH 200000000 INCREMENT BY 1 MAXVALUE
   2999999999) NOT NULL COMMENT 'Purchase Order ID',
         "HISTORY.CREATEDBY" NVARCHAR(10) COMMENT 'Created By',
         "HISTORY.CREATEDAT" DATE CS_DAYDATE COMMENT 'Created Date',
         "HISTORY.CHANGEDBY" NVARCHAR(10) COMMENT 'Changed By',
         "HISTORY.CHANGEDAT" DATE CS_DAYDATE COMMENT 'Change Date',
         "NOTEID" NVARCHAR(10) COMMENT 'Notes',
         "PARTNER" NVARCHAR(10) COMMENT 'Supplier',
         "CURRENCY" NVARCHAR(5) COMMENT 'Currency',
         "GROSSAMOUNT" DECIMAL(15,2) CS_FIXED COMMENT 'Gross Amount',
         "NETAMOUNT" DECIMAL(15,2) CS_FIXED COMMENT 'Net Amount',
         "TAXAMOUNT" DECIMAL(15,2) CS_FIXED COMMENT 'Tax Amount',
         "LIFECYCLESTATUS" NVARCHAR(1) COMMENT 'Lifecycle Status',
         "APPROVALSTATUS" NVARCHAR(1) COMMENT 'Approval Status',
         "CONFIRMSTATUS" NVARCHAR(1) COMMENT 'Confirmation Status',
         "ORDERINGSTATUS" NVARCHAR(1) COMMENT 'Ordering Status',
         "INVOICINGSTATUS" NVARCHAR(1) COMMENT 'Invoicing Status',
         PRIMARY KEY ("PURCHASEORDERID"))
         COMMENT 'Purchase Order Header'
         WITH ASSOCIATIONS( JOIN "PurchaseOrder.Item" AS "ITEMS" ON
   "PURCHASEORDERID" = "PURCHASEORDERID")
         UNLOAD PRIORITY 5 AUTO MERGE
   ```

   > **i Note**
   >
   > This code example defines an association with a second table, `"PurchaseOrder.Item"` defined in `PurchaseOrderItem.hdbtable`, which we define in the next step.

5. Save the SQL DDL table-definition file.

   Saving the definition persists the file in your local workspace; it does not create any objects in the database catalog.

6. Create and define a second design-time table `PurchaseOrderItem.hdbtable` using SQL DDL.

   The purchase-order **items** table is associated with the purchase-order **header** table to provide a basis for an SQL view that you can define in subsequent task.

   a. Right-click the folder where you want to save the SQL DDL table-definition file and choose *New* > *Database Artifact* in the context-sensitive pop-up menu.
   b. Enter the name of the table-definition file in the *File Name* box, for example, `PurchaseOrderItem`. The `.hdbtable` file suffix is added automatically.
   c. Choose *Finish* to save the new SQL DDL table-definition file in the database module of your local project workspace.
   d. Define the structure of the SQL DDL table `PurchaseOrderItem`.

If the new table-definition file is not automatically displayed by the file-creation wizard, double-click the table-definition file you created in the previous step and add the SQL DDL code that defines the table to the file:

> **i Note**
>
> The following code example is provided for illustration purposes only.

```
COLUMN TABLE "PurchaseOrder.Item" (
      "POHeader.PURCHASEORDERID" INTEGER CS_INT NOT NULL COMMENT 'Purchase
Order ID',
      "PRODUCT" NVARCHAR(10) NOT NULL COMMENT 'Product ID',
      "NOTEID" NVARCHAR(10) COMMENT 'Notes',
      "CURRENCY" NVARCHAR(5) COMMENT 'Currency',
      "GROSSAMOUNT" DECIMAL(15,2) CS_FIXED COMMENT 'Gross Amount',
      "NETAMOUNT" DECIMAL(15,2) CS_FIXED COMMENT 'Net Amount',
      "TAXAMOUNT" DECIMAL(15,2) CS_FIXED COMMENT 'Tax Amount',
      "QUANTITY" DECIMAL(13,3) CS_FIXED COMMENT 'Quantity',
      "QUANTITYUNIT" NVARCHAR(3) COMMENT 'Quantity Unit',
      "DELIVERYDATE" DATE CS_DAYDATE COMMENT 'Delivery Date',
      PRIMARY KEY ("POHeader.PURCHASEORDERID","PRODUCT"))
      COMMENT 'Purchase Order Item'
      WITH ASSOCIATIONS( JOIN "PurchaseOrder.Header" AS "POHeader" ON
"POHeader"."PURCHASEORDERID" = "POHeader.PURCHASEORDERID")
      UNLOAD PRIORITY 5 AUTO MERGE
```

> **i Note**
>
> This code example defines an association with a second table, `"PurchaseOrder.Header"` defined in `PurchaseOrderHeader.hdbtable`, which we defined in the previous step. These two tables are the basis for an SQL view that you can create in a subsequent task.

7. Activate the changes in the catalog.

   To activate the new entity definition and generate a corresponding table in the catalog, use the *Build* feature.

   a. Right-click the new database module in your application project.
   b. In the context-sensitive pop-up menu, choose *Build*.

   > **→ Tip**
   >
   > You can follow progress of the build in the console at the bottom of the code editor.

8. Check that the new tables have been successfully created in the catalog.

   > **→ Tip**
   >
   > A selection of run-time tools is available in the *Database Explorer* perspective of SAP web IDE for SAP HANA at the following location:
   >
   > *Tools* > *Database Explorer*

   In XS advanced, your database run-time objects are located in the HDI container created for your multi-target application's database module; you need to locate and bind to this application-specific container to view its contents. The container name contains the name of the user logged into the SAP Web IDE

for SAP HANA, the name of the database module containing the CDS design-time entities, and the string *-hdi-container*, for example:

`<XS_UserName>`*-ctetig24[...]-*`<DB_Module>`*-hdi-container*

To view the contents of the HDI container that is bound to your development project, open the SAP HANA run-time *Development* perspective, open the project containing the new tables, right-click the database module where the design-time tables are stored, and in the context menu select *Open HDI Container*.

> → Tip
>
> The new tables are listed in the *Tables* node. You can display details of the table definition, however, the tables do not yet contain any data, so the *Open Data* tool will not work.

9. Delete a table created by SAP HDI (**optional**).

   If you need to delete a table created by HDI in the context of the SAP Web IDE, perform the following steps:

   a. Remove the design-time table definition from the database module of your application project in SAP Web IDE.
   b. Right-click the database module in your application project.
   c. In the context-sensitive pop-up menu, choose *Build*.

   > → Tip
   >
   > In the context of HDI, you can use the HDI container API's `DELETE` command to remove files from the HDI container's virtual file system and then use the `MAKE` command with the path to the file slated for removal specified in the `undeploy_paths` parameter. For more information about the HDI APIs, see **The HDI Container API** in *Related Information* below.

Related Information

Creating Data Persistence Artifacts with SQL DDL in XS Advanced [page 284]
Load Data into a Database Table with the "hdbtabledata" Artifact [page 293]
Data Definition Statements (SQL Reference)
The HDI Container API

5.3.3.2 Create a Database View with SQL Data Definition Language

Define a design-time database **view** using the SQL Data Definition Language (DDL) syntax.

Prerequisites

To complete this task successfully, note the following prerequisites:

- You must have access to an SAP HANA system.
- For the purposes of this tutorial, you must have access to the SAP Web IDE for SAP HANA. Generally though, you can use any text editor to create a design-time artifact.
- You must have already created a development workspace and a multi-target application (MTA) project.
- You must already have created a database module for your MTA application project.
- You must already have set up an HDI container for the database artifacts

  > ℹ Note
  >
  > A container setup file (`.hdiconfig`) is required to define which plug-ins to use to create the corresponding catalog objects from your design-time artifacts when the multi-target application (or just the database module) is deployed.

- To view run-time objects, you must have access to the SAP HANA Database Explorer tools that enable you to view the contents of the catalog.

Context

In the SAP HANA database, as in other relational databases, a view contains a set of data elements that are organized using columns and rows. SAP HANA Deployment Infrastructure (HDI) enables you to use the SQL DDL syntax to create a database view as a design-time file. Building or deploying the database module that contains the SQL DDL view creates the corresponding view in the corresponding schema.

During the build process, the `.hdbview` plug-in is called to transform a design-time view resource into an SQL view database object. The file format uses a DDL-style syntax which is equivalent to the syntax required in the corresponding SQL statement `CREATE VIEW` , but without the leading "`CREATE`". If the view selects from a synonym which points to an object inside another schema or a different container or to an object inside the same container which is owned by a different user, then the container's object owner ("`<container>#OO`") needs to have the required privileges on this target object, for example, `SELECT`, `UPDATE`, or `INSERT`. If the views are exposed to other users, the object owner usually needs privileges `WITH GRANT OPTION`. If access to the view needs to be filtered by means of analytic (or structured) privileges, then the view defined must contain the `WITH STRUCTURED PRIVILEGE CHECK` clause, as illustrated in the examples below.

To create a design-time SQL DDL view-definition file, perform the following steps:

Procedure

1. Start SAP HANA Web IDE for SAP HANA.

   The SAP Web IDE for SAP HANA is typically available at the following URL:

   `https://<HANA_HOST>:53075/`

   > → Tip
   >
   > To display the URL for the SAP Web IDE for SAP HANA, open a command shell, log on to the XS advanced run time using the `xs login` command, and run the following command:

```
xs app webide --urls
```

2. Open the application project to which you want to add your SQL DDL view.
3. Create the view-definition file in SQL DDL.

   Browse to the folder in the database module in your application's project workspace, for example, `<MyApp1>/HDB/src` where you want to create the new SQL DDL view-definition file and perform the following steps:

   a. Right-click the folder where you want to save the SQL DDL view-definition file and choose *New* > *Database Artifact* in the context-sensitive pop-up menu.
   b. Enter the name of the view-definition file in the *File Name* box, for example, `PurchaseOrderItemView`.

      You can specify the corresponding file extension from the drop-down list in the *File Type* text box.

      > → Tip
      >
      > The file extension is mandatory; it is used to determine which plug-in to call when creating the corresponding run-time object during application build or deployment. SQL DDL role-definition artifacts have the file extension `.hdbview`, for example, `PurchaseOrderItem.hdbview`.

   c. Choose *Finish* to save the new SQL DDL view-definition file in the database module of your local project workspace.
4. Define the structure of the SQL DDL view.

   Locate and double-click the new view-definition file that you created in the previous step, for example, `PurchaseOrderItem.hdbview`, and add the following SQL DDL code that defines the view:

   > ℹ Note
   >
   > The following code example is provided for illustration purposes only.

   `PurchaseOrderItem.hdbview`

```
VIEW "PurchaseOrder.ItemView" COMMENT 'Purchase Order Item View' (
     "PurchaseOrderItemId",
     "PartnerId",
     "ProductID",
     "CurrencyCode",
     "Amount",
     "NetAmount",
     "TaxAmount",
     "Quantity",
     "QuantityUnit",
     "DeliveryDate1"
     ) AS SELECT
     "Item_$0"."POHeader"."PURCHASEORDERID" AS "PurchaseOrderItemId" ,
     "Item_$0"."POHeader"."PARTNER" AS "PartnerId" ,
     "Item_$0"."PRODUCT" AS "ProductID" ,
     "Item_$0"."CURRENCY" AS "CurrencyCode" ,
     "Item_$0"."GROSSAMOUNT" AS "Amount" ,
     "Item_$0"."NETAMOUNT" AS "NetAmount" ,
     "Item_$0"."TAXAMOUNT" AS "TaxAmount" ,
     "Item_$0"."QUANTITY" AS "Quantity" ,
     "Item_$0"."QUANTITYUNIT" AS "QuantityUnit" ,
     "Item_$0"."DELIVERYDATE" AS "DeliveryDate1"
FROM "PurchaseOrder.Item" AS "Item_$0"
WITH READ ONLY STRUCTURED PRIVILEGE CHECK
```

> ### Note
>
> The view `PurchaseOrder.ItemView` specifies an association with the table `"PurchaseOrder.Item"` defined in `PurchaseOrderItem.hdbtable`, which we describe in another task. Access to the view is also restricted by a `STRUCTURED PRIVILEGE CHECK`, which is described in another task. For more information about either task, see *Create a Database Table with SQL Data Definition Language* and *Create a Structured Privilege with SQL Data Definition Language*, respectively, in *Related Information* below.

5. Save the SQL DDL view-definition file.

   Saving the view definition persists the file in your local workspace; it does not create any objects in the database catalog.

6. Activate the changes in the catalog.

   To activate the new view definition and generate a corresponding view object in the catalog, use the *Build* feature.

   a. Right-click the new database module in your application project.

   b. In the context-sensitive pop-up menu, choose *Build* > *Build*.

   > ### → Tip
   >
   > You can follow progress of the build in the console at the bottom of the code editor.

7. Check that the new view has been successfully created in the catalog.

> ### → Tip
>
> A selection of run-time tools is available in the *Database Explorer* perspective of SAP web IDE for SAP HANA at the following location:
>
> *Tools* > *Database Explorer*

In XS advanced, your database run-time objects are located in the HDI container created for your multi-target application's database module; you need to locate and bind to this application-specific container to view its contents. The container name contains the name of the user logged into the SAP Web IDE for SAP HANA, the name of the database module containing the CDS design-time entities, and the string *-hdi-container*, for example:

`<XS_UserName>`*-ctetig24[...]-*`<DB_Module>`*-hdi-container*

To view the contents of the HDI container that is bound to your development project, open the SAP HANA run-time *Development* perspective, open the project containing the new tables, right-click the database module where the design-time tables are stored, and in the context menu select *Open HDI Container*.

> ### → Tip
>
> The new views are listed in the *Views* node. However, the view we created in this task is protected by a "read-only" structured privilege which still needs to be defined. In addition, until the underlying tables used by the view contain some data, the *Open Data* tool will not display the view contents.

Related Information

Create a Database Table with SQL Data Definition Language [page 285]
Data Definition Statements (SQL Reference)

5.3.3.3 Load Data into a Database Table with the "hdbtabledata" Artifact

Use an `hdbtabledata` artifact to load data external data into an existing database table.

Prerequisites

To complete this task successfully, note the following prerequisites:

- You must have access to an SAP HANA system.
- For the purposes of this tutorial, you must have access to the SAP Web IDE for SAP HANA. Generally though, you can use any text editor to create a design-time artifact.
- You must have already created a development workspace and a multi-target application (MTA) project.
- You must already have created a database module for your MTA application project.
- You must already have set up an HDI container for the database artifacts

  > **i** Note
  >
  > A container setup file (`.hdiconfig`) is required to define which plug-ins to use to create the corresponding catalog objects from your design-time artifacts when the multi-target application (or just the database module) is deployed.

- To view run-time objects, you must have access to the SAP HANA Database Explorer tools that enable you to view the contents of the catalog.

Context

You can use the `hdbtabledata` development object to enable the loading of data from one or more comma-separated-values (CSV) files into the specified target tables during their creation. The CSV files used as the data source are also created as part of this exercise.

Procedure

1. Start SAP HANA Web IDE for SAP HANA.

The SAP Web IDE for SAP HANA is typically available at the following URL:

`https://<HANA_HOST>:53075/`

> → Tip
>
> To display the URL for the SAP Web IDE for SAP HANA, open a command shell, log on to the XS advanced run time using the `xs login` command, and run the following command:
>
> `xs app webide --urls`

2. Open the application project to which you want to add your SQL DDL table-data artifact.
3. Create the table-data definition file.

   Create a new folder for the data and browse to the new folder in the database module in your application's project workspace, for example, `<MyApp1>/HDB/src/data` where you want to create the new SQL DDL table-data definition file and perform the following steps:

   a. Right-click the folder where you want to create the SQL DDL table-data definition file and choose *New* > *Folder* in the context-sensitive pop-up menu and enter *data* as the folder name.
   b. Right-click the `data/` folder where you want to create the SQL DDL table-data definition file and choose *New* > *Database Artifact* in the context-sensitive pop-up menu.
   c. Enter the name of the table-definition file in the *File Name* box, for example, `PurchaseDemo.hdbtabledata`.

      > → Tip
      >
      > Use the drop-down menu in the *File Type* box to manually define the file extension, for example, `.hdbtabledata`. The file extension is used to determine which plug-in to use to create the corresponding run-time object during deployment. SQL DDL table-data definition artifacts have the file extension `.hdbtabledata`, for example, `PurchaseDemo.hdbtabledata`.

   d. Choose *Finish* to save the new SQL DDL table-data definition file in the`data/` folder of the database module in your local project workspace.
4. Define the structure of the table data you want to import.

   If the new table-data definition file is not automatically displayed by the file-creation wizard, double-click the table-data definition file you created in the previous step, for example, `PurchaseDemo.hdbtabledata`, and add the code that defines the data, for example: the data source (a CSV file), the target table where the data is inserted, any column mappings, etc., as illustrated in the following example:

   > ℹ Note
   >
   > The following code example is provided for illustration purposes only.

```
{
    "format_version": 1,
    "imports": [{
        "target_table": "PurchaseOrder.Header",
        "source_data": {
            "data_type": "CSV",
            "file_name": "header.csv",
            "has_header": false,
            "dialect": "HANA",
            "type_config": {
                "delimiter": ","
```

```
            }
        },
        "import_settings": {
             "include_filter" : [ ],
            "import_columns": ["PURCHASEORDERID",
            "NOTEID",
            "PARTNER",
            "CURRENCY",
            "GROSSAMOUNT",
            "NETAMOUNT",
            "TAXAMOUNT",
            "LIFECYCLESTATUS",
            "APPROVALSTATUS",
            "CONFIRMSTATUS",
            "ORDERINGSTATUS",
            "INVOICINGSTATUS"]
        },
        "column_mappings": {
            "PURCHASEORDERID": 1,
            "NOTEID": 6,
            "PARTNER": 7,
            "CURRENCY": 8,
            "GROSSAMOUNT": 9,
            "NETAMOUNT": 10,
            "TAXAMOUNT": 11,
            "LIFECYCLESTATUS": 12,
            "APPROVALSTATUS": 13,
            "CONFIRMSTATUS": 14,
            "ORDERINGSTATUS": 15,
            "INVOICINGSTATUS": 16
        }
    },
    {
        "target_table": "PurchaseOrder.Item",
        "source_data": {
            "data_type": "CSV",
            "file_name": "item.csv",
            "has_header": false,
            "dialect": "HANA",
            "type_config": {
                "delimiter": ","
            }
        },
        "import_settings": {
               "include_filter" : [ ],
            "import_columns": ["POHeader.PURCHASEORDERID",
            "PRODUCT",
            "NOTEID",
            "CURRENCY",
            "GROSSAMOUNT",
            "NETAMOUNT",
            "TAXAMOUNT",
            "QUANTITY",
            "QUANTITYUNIT" ]
        },
        "column_mappings": {
            "POHeader.PURCHASEORDERID": 1,
            "PRODUCT": 3,
            "NOTEID": 4,
            "CURRENCY": 5,
            "GROSSAMOUNT": 6,
            "NETAMOUNT": 7,
            "TAXAMOUNT": 8,
            "QUANTITY": 9,
            "QUANTITYUNIT": 10
        }
    }]
}
```

5. Create the comma-separated-values (CSV) files to use as the source for the table data you want to import into your database table.

   You should also create a new folder called "`loads`" to store the data-source files in your proje, for example, `<MyApp1>/HDB/src/data/loads` where you want to create the new CSV data-source files and perform the following steps:

   a. Right-click the folder where you want to create the SQL DDL table-data definition file and choose *New* > *Folder* in the context-sensitive pop-up menu and enter *data* as the folder name.
   b. Right-click the `data/` folder where you want to create the SQL DDL table-data definition file and choose *New* > *Database Artifact* in the context-sensitive pop-up menu.
   c. Enter the name of the table-data definition file in the *File Name* box, for example, `header.csv`.

      > → Tip
      >
      > Use the drop-down menu in the *File Type* box to manually define the file extension, for example, `.hdbtabledata`. The file extension is used to determine which plug-in to use to create the corresponding run-time object during deployment. SQL DDL table-data definition artifacts have the file extension `.csv`, for example, `header.csv`.

   d. Choose *Finish* to save the new CSV file in the `data/loads` folder of the database module in your local project workspace.
   e. Add data to the new CSV file `header.csv` in the `data/loads` folder of the database module in your local project workspace.
      `myHDBmodule/src/data/loads/header.csv`

      ```
      0500000000,0000000033,20120101,0000000033,20120101,9000000001,0100000000,EU
      R,13224.47,11113,2111.47,N,I,I,I,I
      0500000001,0000000033,20120102,0000000033,20120102,9000000001,0100000002,EU
      R,12493.73,10498.94,1994.79,N,I,I,I,I
      ```

   f. Create a new CSV file called `item.csv`.

      Repeat these steps described above to create a new CSV file that contains the data for the purchase order items. The CSV file should be called `item.csv` and be placed in the folder `loads` folder of the application's database module alongside the `header.csv` CSV file.

   g. Add data to the new CSV file `item.csv` in the `data/loads` folder of the database module in your local project workspace.
      `myHDBmodule/src/data/loads/item.csv`

      ```
      0500000000,0000000010,HT-1000,,EUR,1137.64,956,181.64,1,EA,20121204
      0500000000,0000000020,HT-1091,,EUR,61.88,52,9.88,2,EA,20121204
      0500000000,0000000030,HT-6100,,EUR,1116.22,938,178.22,2,EA,20121204
      0500000000,0000000040,HT-1001,,EUR,2275.28,1912,363.28,2,EA,20121204
      0500000000,0000000050,HT-1092,,EUR,92.82,78,14.82,3,EA,20121204
      0500000000,0000000060,HT-6101,,EUR,1116.22,938,178.22,2,EA,20121204
      0500000000,0000000070,HT-1002,,EUR,2275.28,1912,363.28,2,EA,20121204
      0500000000,0000000080,HT-1090,,EUR,61.88,52,9.88,2,EA,20121204
      0500000000,0000000090,HT-6102,,EUR,1674.33,1407,267.33,3,EA,20121204
      0500000000,0000000100,HT-1007,,EUR,3412.92,2868,544.92,3,EA,20121204
      0500000001,0000000010,HT-1100,,USD,213.96,179.8,34.16,2,EA,20121204
      0500000001,0000000020,HT-2026,,USD,35.69,29.99,5.7,1,EA,20121204
      0500000001,0000000030,HT-1002,,USD,3736.6,3140,596.6,2,EA,20121204
      0500000001,0000000040,HT-1101,,USD,213.96,179.8,34.16,2,EA,20121204
      0500000001,0000000050,HT-2027,,USD,71.38,59.98,11.4,2,EA,20121204
      0500000001,0000000060,HT-1003,,USD,3736.6,3140,596.6,2,EA,20121204
      0500000001,0000000070,HT-1102,,USD,320.94,269.7,51.24,3,EA,20121204
      0500000001,0000000080,HT-2028,,USD,107.06,89.97,17.09,3,EA,20121204
      0500000001,0000000090,HT-1004,,USD,3736.6,3140,596.6,2,EA,20121204
      ```

```
0500000001,0000000100,HT-1103,,USD,320.94,269.7,51.24,3,EA,20121204
```

h. Save all the table-data definition and CSV files.

i. Build the database module again.

6. Load the data into the target tables.

   To activate the new table-data definition artifacts and load the source data into the target tables, rebuild the database module of your application, as follows:

   a. Right-click the new database module in your application project.

   b. In the context-sensitive pop-up menu, choose *Build* > *Build*.

   > → Tip
   >
   > You can follow progress of the build in the console at the bottom of the code editor.

7. View the contents of the activated tables in the catalog.

   You can use the *Database Explorer* to view the activated tables in the run-time catalog and verify that the tables contain the imported data.

   a. Right-click the database module in your application project and in the context menu choose *Open HDI Container*.

   b. Select *Tables* from the list; a list of activated tables is displayed in the *Catalog* pane.

   c. Select and right-click a table (for example, `PurchaseOrder.Header` or `PurchaseOrder.Item`) and in the context menu choose *Open Data*.

Related Information

Create a Database Table with SQL Data Definition Language [page 285]
Data Definition Statements (SQL Reference)

5.3.3.4 Create a Structured Privilege Using SQL Data Definition Language

Define a structured privilege using the SQL Data Definition Language (DDL) syntax.

Prerequisites

To complete this task successfully, note the following prerequisites:

- You must have access to an SAP HANA system.
- For the purposes of this tutorial, you must have access to the SAP Web IDE for SAP HANA. Generally though, you can use any text editor to create a design-time artifact.
- You must have already created a development workspace and a multi-target application (MTA) project.

- You must already have created a database module for your MTA application project.
- You must already have set up an HDI container for the database artifacts

  > **i** Note
  >
  > A container setup file (`.hdiconfig`) is required to define which plug-ins to use to create the corresponding catalog objects from your design-time artifacts when the multi-target application (or just the database module) is deployed.

- To view run-time objects, you must have access to the SAP HANA Database Explorer tools that enable you to view the contents of the catalog.

Context

Access to the HDI database objects from XS advanced is performed automatically by the HDI container's technical user. If you want to grant additional privileges to the technical user (for example, in structured privileges that grant access to an SQL view), you first need to create this structured privilege.

SAP HANA Deployment Infrastructure (HDI) enables you to use the SQL DDL syntax to define a structured privilege in a design-time file. Building or deploying the database module that contains the SQL DDL structured privilege creates the corresponding object in the corresponding schema.

During the build process, the `.hdbstructuredprivilege` plug-in is called to transform a design-time structured-privilege resource into an SQL database object. The format or the design-time file requires a DDL-style syntax which is equivalent to the syntax required in the corresponding SQL statement `CREATE STRUCTURED PRIVILEGE`, but without the leading "`CREATE`", as illustrated in the code examples below.

> **i** Note
>
> The views to which the structured privileges apply must include the `WITH STRUCTURED PRIVILEGE CHECK` clause.

A dedicated database role, created in another tutorial, can also be used to provide broader access for other database users. For more information about creating a database role in SQL DDL, see *Create a Database Role Using SQL Data Definition Language* in *Related Information* below.

To create a design-time SQL DDL structured-privilege definition file, perform the following steps:

Procedure

1. Start SAP HANA Web IDE for SAP HANA.

   The SAP Web IDE for SAP HANA is typically available at the following URL:

   `https://<HANA_HOST>:53075/`

   > → Tip
   >
   > To display the URL for the SAP Web IDE for SAP HANA, open a command shell, log on to the XS advanced run time using the `xs login` command, and run the following command:

```
xs app webide --urls
```

2. Open the application project to which you want to add your SQL DDL structured privilege.
3. Create the structured-privilege definition file.

   Browse to the folder in the database module in your application's project workspace, for example, `<MyApp1>/HDB/src/roles` where you want to create the new SQL DDL structured-privilege definition file and perform the following steps:

   a. Right-click the folder where you want to add a new folder to store the SQL DDL structured-privilege definition file and choose *New* > *Folder* in the context-sensitive pop-up menu. Add a folder called `roles`.

   b. Right-click the new `roles` folder and choose *New* > *Database Artifact* in the context-sensitive pop-up menu.

   c. Enter the name of the structured-privilege definition file in the *File Name* box, for example, `PurchaseOrder`.

      You can specify the corresponding file extension from the drop-down list in the *File Type* text box.

      > → Tip
      >
      > The file extension is mandatory; it is used to determine which plug-in to call when creating the corresponding run-time object during application build or deployment. SQL DDL structured-privilege definition artifacts have the file extension `.hdbstructuredprivilege`, for example, `PurchaseOrder.hdbstructuredprivilege`.

   d. Choose *Finish* to save the new SQL DDL structured-privilege definition file in the `roles` folder of the database module of your local project workspace.

4. Define the structure of the SQL DDL structured privilege.

   Locate and double-click the structured-privilege definition file that you created in the previous step, for example, `PurchaseOrder.hdbstructuredprivilege`, and add the following SQL DDL code that defines the structured-privilege:

   > ℹ Note
   >
   > The following code example is provided for illustration purposes only. For this reason, the filter value `'EUR'` is hard-coded; in a real-life example, the attribute value used in the privilege would typically be resolved from user attributes defined elsewhere.

   `PurchaseOrder.hdbstructuredprivilege`

   ```
   STRUCTURED PRIVILEGE
       "PO_VIEW_PRIVILEGE"
       FOR SELECT ON
       "PurchaseOrder.ItemView"
       WHERE "CurrencyCode" = 'EUR'
   ```

   > ℹ Note
   >
   > This example of a structured privilege is used in the SQL view `"PurchaseOrder.ItemView"`, which is created and defined in another task. For more information, see *Create a Database View with SQL Data Definition Language* in *Related Information* below.

5. Save all files and rebuild the database module.

a. Right-click the new database module in your application project.
b. In the context-sensitive pop-up menu, choose *Build* > *Build*.

> → Tip
>
> You can follow progress of the build in the console at the bottom of the code editor.

6. Verify that the structured privilege is working and filtering access to the view data.

   You can use the *Database Explorer* to display activated views in the run-time catalog and verify that the view contains the selected data.

   a. Right-click the database module in your application project and in the context menu choose *Open HDI Container*.
   b. Select *Views* from the list; a list of activated views is displayed in the *Catalog* pane.
   c. Select and right-click view `PurchaseOrder.ItemView` and in the context menu choose *Open Data*.

   Thanks to the new structured privilege required by the view, you should now be able to see data in the view, but only those records with currency of EUR, as specified in the structured-privilege.

> i Note
>
> To enable a database user other than the technical user to access the structured-privilege-based view, a suitable database role must be granted to the additional database user requiring access. This rule also applies for non-technical-user access to any other catalog objects. For more information about creating roles, see *Create a Database Role with SQL Data Definition Language* in *Related Information* below.

Related Information

Create a Database View with SQL Data Definition Language [page 289]
Creating Data Persistence Artifacts with SQL DDL in XS Advanced [page 284]
Data Definition Statements (SQL Reference)

5.3.3.5 Create a Database Role

Define a design-time database role using the JavaScript Object Notation (JSON) syntax.

Prerequisites

To complete this task successfully, note the following prerequisites:

- You must have access to an SAP HANA system.
- For the purposes of this tutorial, you must have access to the SAP Web IDE for SAP HANA. Generally though, you can use any text editor to create a design-time artifact.

- You must have already created a development workspace and a multi-target application (MTA) project.
- You must already have created a database module for your MTA application project.
- You must already have set up an HDI container for the database artifacts.

  > **i** Note
  >
  > A container setup file (`.hdiconfig`) is required to define which plug-ins to use to create the corresponding catalog objects from your design-time artifacts when the multi-target application (or just the database module) is built or deployed.

- To view run-time objects, you must have access to the SAP HANA Database Explorer tools that enable you to view the contents of the catalog.

Context

Access to the HDI database objects from XS advanced is performed automatically by the HDI container's technical user. If you want to provide additional users (other than the default technical user assigned to the HDI container) with access to catalog objects, you need to create a dedicated database role and grant this role to the users requiring access to the database objects.

The role plug-in transforms a design-time role resource, a file with the `.hdbrole` extension, into a run-time role, which can be assigned to database users. The format of the design-time role file is based on JSON, and enables you to specify a list of privileges (for example, system, schema, object, and analytic privileges) and, as an additional option, include other roles, too.

> → Tip
>
> The additional roles can be either global or local. `"global_roles"` are created without a schema; `"schema_roles"` are created with a schema. For more information about the design-time role artifact, for example, file-name restrictions and the use of "`_with_grant_option`" privileges, see *HDI Design-Time Resources* in *Related Information* below.

To create and activate a database role, perform the following steps:

Procedure

1. Start SAP HANA Web IDE for SAP HANA.

   The SAP Web IDE for SAP HANA is typically available at the following URL:

   `https://<HANA_HOST>:53075/`

   > → Tip
   >
   > To display the URL for the SAP Web IDE for SAP HANA, open a command shell, log on to the XS advanced run time using the `xs login` command, and run the following command:
   >
   > ```
   > xs app webide --urls
   > ```

2. Open the application project to which you want to add your SQL DDL role.
3. Create the role-definition file `admin.hdbrole`.

   Browse to the folder in the database module in your application's project workspace, for example, `<MyApp1>/HDB/src/roles` where you want to create the new SQL DDL role-definition file and perform the following steps:

   a. If it does not already exist, add a folder called `roles` to the database module of your application.

      Right-click the folder `src/` in the database module of your application project and choose *New* > *Folder* in the context-sensitive pop-up menu.

   b. Right-click the `roles` folder and choose *New* > *Database Artifact* in the context-sensitive pop-up menu.

   c. Enter the name of the role-definition file in the *File Name* box, for example, `admin`.

      You can specify the corresponding file extension from the drop-down list in the *File Type* text box.

      > → Tip
      >
      > The file extension is mandatory; it is used to determine which plug-in to call when creating the corresponding run-time object during application build or deployment. SQL DDL role-definition artifacts have the file extension `.hdbrole`, for example, `admin.hdbrole`.

   d. Choose *Finish* to save the new SQL DDL role-definition file in the `roles` folder of the database module of your local project workspace.

4. Define the contents of the SQL DDL role `admin.hdbrole`.

   Locate and double-click the role-definition file that you created in the previous step, for example, `admin.hdbrole`, and add the following SQL DDL code that defines the role scope:

   > ℹ Note
   >
   > The following code example is provided for illustration purposes only.

```
{
  "role":{
     "name": "admin",
     "schema_privileges": [{
        "privileges": ["SELECT METADATA",
                       "SELECT CDS METADATA",
                       "SELECT",
                       "INSERT",
                       "EXECUTE",
                       "DELETE",
                       "UPDATE",
                       "CREATE TEMPORARY TABLE"
         ]
     }],
     "schema_analytic_privileges": [
        {
          "privileges":[ "PO_VIEW_PRIVILEGE" ]
        }
        ]
      }
}
```

5. Create the role-definition file `adminOwner.hdbrole`.

Browse to the folder in the database module in your application's project workspace, for example, `<MyApp1>/HDB/src/roles` where you want to create the new SQL DDL role-definition file and perform the following steps:

a. Right-click the `roles` folder in the database module and choose *New* > *Database Artifact* in the context-sensitive pop-up menu.
b. Enter the name of the role-definition file in the *File Name* box, for example, `adminOwner`.

   You can specify the corresponding file extension from the drop-down list in the *File Type* text box.

   > → Tip
   >
   > The file extension is mandatory; it is used to determine which plug-in to call when creating the corresponding run-time object during application build or deployment. SQL DDL role-definition artifacts have the file extension `.hdbrole`, for example, `adminOwner.hdbrole`.

c. Choose *Finish* to save the new SQL DDL role-definition file in the `roles` folder of the database module of your local project workspace.

6. Define the contents of the SQL DDL role `adminOwner.hdbrole`.

   Locate and double-click the role-definition file that you created in the previous step, for example, `admin.hdbrole`, and add the following SQL DDL code that defines the role scope:

   > i Note
   >
   > The following code example is provided for illustration purposes only.

   ```
   {
     "role":{
       "name": "admin#",
       "schema_roles": [{
       "names": ["admin"]
       }]
     }
   }
   ```

   > → Tip
   >
   > A role name that ends with the hash character (#), for example, `"admin#"` can include "`with-grant-option`" privileges as well as references to other roles whose names also end with #. However, a role whose name ends with # can only be granted to an HDI container's object owner (`<container>#OO`).

7. Assign the `admin` role to the technical user of the application container.

   To ensure the automatic assignment of the `admin` role to the technical user of the application container, you need to create a role with a special name `default_access_role` as described in the following steps:

   a. If it does not already exist, add a folder called `defaults` to the database module of your application.

      Right-click the folder `src/` in the database module of your application project and choose *New* > *Folder* in the context-sensitive pop-up menu. Enter the name `defaults`.

   b. Right-click the `defaults` folder and choose *New* > *Database Artifact* in the context-sensitive pop-up menu.
   c. Enter the name of the new role-definition file in the *File Name* box, for example, `default_access_role`.

      You can specify the corresponding file extension from the drop-down list in the *File Type* text box.

> → Tip
>
> The file extension is mandatory; it is used to determine which plug-in to call when creating the corresponding run-time object during application build or deployment. SQL DDL role-definition artifacts have the file extension `.hdbrole`, for example, `default_access_role.hdbrole`.

d. Define the contents of the new SQL DDL role `default_access_role.hdbrole`.

Locate and double-click the role-definition file that you created in the previous step, for example, `default_access_role.hdbrole`, and add the following SQL DDL code that defines the role scope:

> i Note
>
> The following code example is provided for illustration purposes only.

```
{
  "role":{
    "name": "default_access_role",
    "schema_roles": [{
      "names": ["admin"]
    }]
  }
}
```

e. Choose *Finish* to save the new SQL DDL role-definition file in the `defaults` folder of the database module of your local project workspace.

8. Save all files and rebuild the database module.
   a. Right-click the new database module in your application project.
   b. In the context-sensitive pop-up menu, choose *Build* > *Build*.

> → Tip
>
> You can follow progress of the build in the console at the bottom of the code editor.

9. Verify that the new roles are working and enabling access to the view data.

You can use the *Database Explorer* to display activated views in the run-time catalog and verify that the view contains the selected data. The SQL view `PurchaseOrder.ItemView` should now be accessible to database users other than the HDI container's technical database user.

a. Right-click the database module in your application project and in the context menu choose *Open HDI Container*.
b. Select *Views* from the list; a list of activated views is displayed in the *Catalog* pane.
c. Select and right-click view `PurchaseOrder.ItemView` and in the context menu choose *Open Data*.

Related Information

Create a Database View with SQL Data Definition Language [page 289]
Create a Structured Privilege Using SQL Data Definition Language [page 297]
Data Definition Statements (SQL Reference)
HDI Design-Time Resources and Build Plug-ins [page 279]

5.3.3.6 Create a Database Constraint with SQL Data Definition Language

Define a design-time database **constraint** using the SQL Data Definition Language (DDL) syntax.

Prerequisites

To complete this task successfully, note the following prerequisites:

- You must have access to an SAP HANA system.
- For the purposes of this tutorial, you must have access to the SAP Web IDE for SAP HANA. Generally though, you can use any text editor to create a design-time artifact.
- You must have already created a development workspace and a multi-target application (MTA) project.
- You must already have created a database module for your MTA application project.
- You must already have set up an HDI container for the database artifacts

  > **i** Note
  >
  > A container setup file (`.hdiconfig`) is required to define which plug-ins to use to create the corresponding catalog objects from your design-time artifacts when the multi-target application (or just the database module) is deployed.

- To view run-time objects, you must have access to the SAP HANA Database Explorer tools that enable you to view the contents of the catalog.

Context

An SQL constraint specifies rules for the data stored in a database table; the rules can apply to a column, a table, multiple tables, or an entire schema. Constraints limit the type of data that can be loaded into a table based on the specified rules. A constraint is typically used to define the properties that data in a database must comply with in order to ensure the consistency, accuracy, and reliability of the data in the database.

You can use the HDI `.hdbconstraint` artifact to define constraints on foreign keys. The file format used in the `.hdbconstraint` artifact is similar to the syntax used in the corresponding SQL statement "`ALTER TABLE ADD CONSTRAINT` ", but phrased as "`CONSTRAINT constraint ON table` ..." as illustrated in the example below.

> **i** Note
>
> The database constraint plug-in (`.hdbconstraint`) only supports `FOREIGN KEY` constraints; unique constraints are handled by the table index plug-in (`.hdbindex`).

Procedure

1. Start SAP HANA Web IDE for SAP HANA.

   The SAP Web IDE for SAP HANA is typically available at the following URL:

   `https://<HANA_HOST>:53075/`

   > → Tip
   >
   > To display the URL for the SAP Web IDE for SAP HANA, open a command shell, log on to the XS advanced run time using the `xs login` command, and run the following command:
   >
   > `xs app webide --urls`

2. Open the application project to which you want to add your SQL DDL database constraint artifact.
3. Create the database-constraint definition file in SQL DDL.

   Browse to the folder in the database module in your application's project workspace, for example, `<MyApp1>/HDB/src` where you want to create the new SQL DDL constraint-definition file and perform the following steps:

   a. Right-click the folder where you want to save the SQL DDL constraint-definition file and choose *New* > *Database Artifact* in the context-sensitive pop-up menu.
   b. Enter the name of the constraint-definition file in the *File Name* box, for example, `PurchaseOrderConstraint`.

      You can specify the corresponding file extension from the drop-down list in the *File Type* text box.

      > → Tip
      >
      > The file extension is mandatory; it is used to determine which plug-in to call when creating the corresponding run-time object during application build or deployment. SQL DDL constraint-definition artifacts have the file extension `.hdbconstraint`, for example, `PurchaseOrderConstraint.hdbconstraint`.

   c. Choose *Finish* to save the new SQL DDL constraint-definition file in the database module of your local project workspace.
4. Define the structure of the SQL DDL database constraint.

   Locate and double-click the new constraint-definition file that you created in the previous step, for example, `PurchaseOrderConstraint.hdbconstraint`, and add the following SQL DDL code that defines the rules of the constraint:

   > ℹ Note
   >
   > The following code example is provided for illustration purposes only.

   `PurchaseOrderConstraint.hdbconstraint`

   ```
   CONSTRAINT PO_CONST
   ON "PurchaseOrder.Item" FOREIGN KEY ("POHeader.PURCHASEORDERID")
   REFERENCES "PurchaseOrder.Header" ("PURCHASEORDERID") ON DELETE CASCADE
   ```

> **i Note**
>
> The database constraint `PurchaseOrderConstraint.hdbconstraint` references the table `"PurchaseOrder.Header""` defined in `PurchaseOrderHeader.hdbtable`, which is described in another task. For more information, see *Create a Database Table with SQL Data Definition Language* in *Related Information* below.

5. Save all files and rebuild the database module.
   a. Right-click the new database module in your application project.
   b. In the context-sensitive pop-up menu, choose *Build* > *Build*.

      > **→ Tip**
      >
      > You can follow progress of the build in the console at the bottom of the code editor.

6. Verify that the new constraint is activated in the catalog.

   You can use the *Database Explorer* to display activated objects in the run-time catalog and verify that the constraint is present.

   a. Right-click the database module in your application project and in the context menu choose *Open HDI Container*.
   b. Select *Indexes* from the list; a list of activated indexes is displayed in the *Catalog* pane.

Related Information

Create an Index with SQL Data Definition Language [page 307]
Create a Database Table with SQL Data Definition Language [page 285]
Data Definition Statements (SQL Reference)

5.3.3.7 Create an Index with SQL Data Definition Language

Define a design-time database **index** using the SQL Data Definition Language (DDL) syntax.

Prerequisites

To complete this task successfully, note the following prerequisites:

- You must have access to an SAP HANA system.
- For the purposes of this tutorial, you must have access to the SAP Web IDE for SAP HANA. Generally though, you can use any text editor to create a design-time artifact.
- You must have already created a development workspace and a multi-target application (MTA) project.
- You must already have created a database module for your MTA application project.

- You must already have set up an HDI container for the database artifacts

  > **i** Note
  >
  > A container setup file (`.hdiconfig`) is required to define which plug-ins to use to create the corresponding catalog objects from your design-time artifacts when the multi-target application (or just the database module) is deployed.

- To view run-time objects, you must have access to the SAP HANA Database Explorer tools that enable you to view the contents of the catalog.

Context

An index is a data structure that helps to make searching data for a specific column in a database faster and more efficient by reducing the number of records, columns, or rows to consider in the search operation. The data structure stores values for indexed columns from one table in another table. Although the indexed data is usually structured in a so-called b-tree, which ensures that the structure is sorted, indexes can also be structured using the r-tree or hash table method, each of which has advantages and disadvantages. For more information, see the *SAP HANA SQL Reference* in *Related Information*.

You can use the HDI `.hdbindex` artifact to define constraints on unique keys. The file format used in the `.hdbindex` artifact is similar to the syntax used in the corresponding SQL statement "`CREATE INDEX` ", but without the leading "`CREATE ...`" as illustrated in the example below.

> **i** Note
>
> The database index plug-in (`.hdbindex`) only supports constraints on unique keys; foreign-key constraints are handled by the database constraint plug-in (`.hdbconstraint`).

Procedure

1. Start SAP HANA Web IDE for SAP HANA.

   The SAP Web IDE for SAP HANA is typically available at the following URL:

   `https://<HANA_HOST>:53075/`

   > **→** Tip
   >
   > To display the URL for the SAP Web IDE for SAP HANA, open a command shell, log on to the XS advanced run time using the `xs login` command, and run the following command:
   >
   > ```
   > xs app webide --urls
   > ```

2. Open the application project to which you want to add your SQL DDL database index artifact.
3. Create the index-definition file in SQL DDL.

Browse to the folder in the database module in your application's project workspace, for example, `<MyApp1>/HDB/src` where you want to create the new SQL DDL index-definition file and perform the following steps:

a. Right-click the folder where you want to save the SQL DDL index-definition file and choose *New* > *Database Artifact* in the context-sensitive pop-up menu.
b. Enter the name of the index-definition file in the *File Name* box, for example, `partnerPO`.

   You can specify the corresponding file extension from the drop-down list in the *File Type* text box.

   > → Tip
   >
   > The file extension is mandatory; it is used to determine which plug-in to call when creating the corresponding run-time object during application build or deployment. SQL DDL index-definition artifacts have the file extension `.hdbindex`, for example, `partnerPO.hdbindex`.

c. Choose *Finish* to save the new SQL DDL index-definition file in the database module of your local project workspace.

4. Define the SQL DDL database index.

   Locate and double-click the new index-definition file that you created in the previous step, for example, `partnerPO.hdbindex`, and add the following SQL DDL code that defines the scope and target of the index:

   > ℹ Note
   >
   > The following code example is provided for illustration purposes only.

   `partnerPO.hdbindex`

   ```
   INDEX "PARTNER_IDX" ON "PurchaseOrder.Header" (PARTNER)
   ```

   > ℹ Note
   >
   > The database index `partnerPO.hdbindex` references the `"PARTNER"` column in the table `"PurchaseOrder.Header"` defined in `PurchaseOrderHeader.hdbtable`, which is described in another task. For more information, see *Create a Database Table with SQL Data Definition Language* in *Related Information* below.

5. Save all files and rebuild the database module.
   a. Right-click the new database module in your application project.
   b. In the context-sensitive pop-up menu, choose *Build* > *Build*.

      > → Tip
      >
      > You can follow progress of the build in the console at the bottom of the code editor.

6. Verify that the new index is activated in the catalog.

   You can use the *Database Explorer* to display activated objects in the run-time catalog and verify that the index is present.

   a. Right-click the database module in your application project and in the context menu choose *Open HDI Container*.
   b. Select *Indexes* from the list; a list of activated indexes is displayed in the *Catalog* pane.

Related Information

Create a Database Constraint with SQL Data Definition Language [page 305]
Create a Database Table with SQL Data Definition Language [page 285]
Data Definition Statements (SQL Reference)

5.3.4 Creating Data Persistence Artifacts with CDS in XS Advanced

You can use Core Data Services to define the underlying database objects that store and provide data for your application, for example, tables, views, and data types.

As part of the process of defining the database persistence model for your application, you create database design-time artifacts such as tables and views, for example using Core Data Services (CDS). You can define your data persistence model in one or more CDS documents. The syntax for the specific data artifacts is described in the following sections:

- CDS documents
- CDS entities (tables)
- CDS user-defined types
- CDS associations (between entities)
- CDS views
- CDS extensions

> **i** Note
>
> SAP HANA CDS and SAP CAP CDS are not interchangeable. Although the Core Data Services (CDS) used by the SAP Cloud Application Programming Model (CAP) is syntactically similar to the SAP HANA CDS used in XS classic and XS advanced, they use different artifact types (`.hdbcds` and `.cds`, respectively) and are intended for use in different development scenarios. For more information about SAP CAP CDS, see *About SAP CAP* in *Related Information* below.

Related Information

Create the Data Persistence Artifacts with CDS in XS Advanced [page 311]
Create a CDS Document (XS Advanced) [page 313]
Create a CDS Entity in XS Advanced [page 348]
Create a CDS User-Defined Structure in XS Advanced [page 372]
Create a CDS Association in XS Advanced [page 386]
Create a CDS View in XS Advanced [page 401]
Create a CDS Extension [page 423]
About CAP (SAP Cloud Application Programming Model)

5.3.4.1 Create the Data Persistence Artifacts with CDS in XS Advanced

Use Core Data Services (CDS) to define the artifacts that make up the data-persistence model.

Context

CDS artifacts are design-time definitions that are used to generate the corresponding run-time objects, when the CDS document that contains the artifact is deployed to the SAP HANA XS advanced model run time. In XS advanced, the CDS document containing the design-time definitions that you create using the CDS-compliant syntax must have the file extension `.hdbcds`, for example, `MyCDSTable.hdbcds`.

The HDI deployment tools deploy database artifacts to an HDI design-time container. Design-time database objects are typically located in the `db/` folder of the application design-time hierarchy, as illustrated in the following example:

Procedure

1. Create the infrastructure in your design-time file-system.

   Design-time database objects are typically located in the `db/src/` folder of the application design-time hierarchy, as illustrated in the following example:

   Sample Code

   ```
   AppName
   |- db/                          # Database deployment artifacts
      |- src/                      # Database artifacts: tables, views, etc.
   ```

2. Define the CDS artifacts.

   The data-persistence model can be defined in one central CDS document (for example, `myCDSmodel.hdbcds`) or split into separate CDS documents (`myCDStype.hdbcds` and `myCDSentity.hdbcds`) which describe individual artifacts, as illustrated in the following example:

   Sample Code

   ```
   AppName
   |- db/                             # Database deployment artifacts
   |  |- src/                         # Database artifacts: tables, views, etc.
   |     |- myCDSentity.hdbcds        # Database design-time table definition
   |     |- myCDSDataType.hdbcds      # Database design-time table definition
   |     \- myCDSmodel.hdbcds         # Database design-time model definition
   ```

3. Add a package descriptor for the XS advanced application.

   The package descriptor (`package.json`) describes the prerequisites and dependencies that apply to the database module of an application in SAP HANA XS advanced. Add a `package.json` file to the root folder of your application's database module (`db/`).

Sample Code

```
AppName
|- db/                              # Database deployment artifacts
|  |- package.json                  # Database details/dependencies
|  |- src/                          # Database artifacts: tables, views, etc.
|     |- myCDSentity.hdbcds         # Database design-time table definition
|     |- myCDSDataType.hdbcds       # Database design-time table definition
|     \- myCDSmodel.hdbcds          # Database design-time definition
overview
```

The basic `package.json` file for your database module (`db/`) should look similar to the following example:

Sample Code

```
{
  "name": "deploy",
  "dependencies": {
     "@sap/hdi-deploy": "3.7.0"
  },
  "scripts": {
     "start": "node node_modules/@sap/hdi-deploy/deploy.js"
  }
}
```

4. Add the container-configuration files required by the SAP HANA Deployment Infrastructure (HDI).

   The following design-time artifacts are used to configure the HDI containers:

   - Container deployment configuration (`.hdiconfig`)
     Mandatory: A JSON file containing a list of the bindings between database artifact types (for example, sequence, procedure, table) and the corresponding deployment plug-in (and version).
   - Run-time container name space rules (`.hdinamespace`)
     Optional: A JSON file containing a list of design-time file suffixes and the naming rules for the corresponding runtime locations.

   Sample Code

   ```
   AppName
   |- db/                              # Database deployment artifacts
   |  |- package.json                  # Database details/dependencies
   |  |- src/                          # Database artifacts: tables, views, etc.
   |     |- .hdiconfig                 # HDI build plug-in configuration
   |     |- .hdinamespace              # HDI run-time name-space configuration
   |     |- myCDSentity.hdbcds         # Database design-time table definition
   |     |- myCDSDataType.hdbcds       # Database design-time table definition
   |     \- myCDSmodel.hdbcds          # Database design-time definition
   overview
   ```

Related Information

Design-Time Database Resources in XS Advanced [page 277]
Creating the Persistence Model in Core Data Services
Managing SAP HDI Containers and Artifacts [page 234]

Create the XS Advanced Application Package Descriptor [page 818]

5.3.4.2 Create a CDS Document (XS Advanced)

A CDS document is a design-time source file that contains definitions of the database objects you want to create in the SAP HANA catalog.

Context

CDS documents are design-time source files that contain DDL code that describes a persistence model according to rules defined in Core Data Services. CDS documents have the file suffix `.hdbcds`. Deploying the application with the database module that contains the CDS document creates the corresponding catalog objects in the corresponding schema.

> **i** Note
>
> The examples of CDS document elements includled in this topic are incomplete `[...]`; it is intended for illustration purposes only.

Procedure

1. Start the SAP HANA Web IDE for SAP HANA.

   The SAP Web IDE for SAP HANA is available at the following URL:

   ```
   https://<HANA_HOST>:53075/
   ```

   > → Tip
   >
   > To display the URL for the SAP Web IDE for SAP HANA, open a command shell, log on to the XS advanced run time, and run the following command:
   >
   > ```
   > xs app webide --urls
   > ```

2. Display the application project to which you want to add a CDS document.

   In XS advanced, SAP Web IDE for SAP HANA creates an application within a context of a project. If you do not already have a project, there are a number of ways to create one, for example: by importing it, cloning it, or creating a new one from scratch.

   a. In the SAP Web IDE for SAP HANA, choose *File* > *New* > *Project from Template*.
   b. Choose the project template type.

      Currently, there is only one type of project template available, namely: *Multi-Target Application Project*. Select *Multi-Target Application Project* and choose *Next*.
   c. Type a name for the new MTA project (for example, **myApp** and choose *Next* to confirm.

d. Specify details of the new MTA project and choose *Next* to confirm.
e. Create the new MTA project; choose *Finish*.

3. Create the CDS document that defines the entity you want to create.

   Database artifacts such as the ones defined in a CDS document belong in the MTA's database module.

   > → Tip
   >
   > If you do not already have a database module, right-click the root folder of your new MTA project and, in the context menu, choose *New* > *HDB Module*. Name the new database model **db**.

   a. Navigate to the `src/` folder in your application's database module `db/`.
   b. Right-click the folder `myworld/db/src/` and choose *New* > *CDS Artifact* in the context menu.
   c. Name the new CDS artifact **`myCDSModel`**.

      The setup Wizard adds the mandatory suffix for CDS artifacts (`.hdbcds`) to the new file name automatically.

4. Define the details of the CDS artifacts.

   In the CDS document you just created , for example, `MyCDSModel.hdbcds`, add the CDS-definition code to the file. The CDS code describes the CDS artifacts you want to add, for example, entity definitions, type definitions, view definitions. Note that the following code examples are provided for illustration purposes only.

   a. Add structured types, if required.

      Use the `type` keyword to define a type artifact in a CDS document. In this example, you add the user-defined types and structured types to the top-level entry in the CDS document, the context `MyCDSModel`.

      ```
      context MyCDSModel {
       type BusinessKey : String(10);
       type SString  : String(40);
       type MyStruct
       {
         aNumber     : Integer;
         aText       : String(80);
         anotherText : MyString80;  // defined in a separate type
       };
      <[...]>
      };
      ```

   b. Add a new context, if required.

      Contexts enable you to group together related artifacts. A CDS document can only contain one top-level context, for example, `MyModel {};`. Any new context must be **nested** within the top-level entry in the CDS document, as shown in the following example.

      ```
      context MyCDSModel {
       type BusinessKey : String(10);
       type SString  : String(40);
       type <[...]>
       context MasterData {
       <[...]>
       };
       context Sales {
       <[...]>
       };
       context Purchases {
       <[...]>
       };
      ```

```
};
```

c. Add new entities.

You can add the entities either to the top-level entry in the CDS document (in this example, the context `MyCDSModel`) or to any other context, for example, `MasterData`, `Sales`, or `Purchases`. In this example, the new entities are column-based tables in the `MasterData` context.

```
context MyCDSModel {
 type BusinessKey : String(10);
 type SString  : String(40);
 type <[...]>
 context MasterData {
    Entity Addresses {
        key  AddressId: BusinessKey;
        City: SString;
        PostalCode: BusinessKey;
        <[...]>
    };
    Entity BusinessPartner {
        key  PartnerId: BusinessKey;
        PartnerRole: String(3);
        <[...]>
    };
 };
 context Sales {
 <[...]>
 };
 context Purchases {
 <[...]>
 };
};
```

5. Save the CDS document.

Related Information

Creating the Persistence Model in Core Data Services
CDS Documents

5.3.4.2.1 CDS Editors

The SAP Web IDE for SAP HANA provides editing tools specially designed to help you create and modify CDS documents.

SAP Web IDE for SAP HANA includes dedicated editors that you can use to define data-persistence objects in CDS documents using the DDL-compliant Core Data Services syntax. SAP HANA XS advanced model recognizes the `.hdbcds` file extension required for CDS object definitions and, at deployment time, calls the appropriate plug-in to parse the content defined in the CDS document and create the corresponding run-time object in the catalog. If you right-click a file with the `.hdbcds` extension in the *Project Explorer* view of your application project, SAP Web IDE for SAP HANA provides the following choice of editors in the context-sensitive menu.

- CDS Text Editor [page 316]

View and edit DDL source code in a CDS document as text with the syntax elements highlighted for easier visual scanning.

Right-click a CDS document: *Open With* > *Text Editor*

- CDS Graphical Editor [page 317]
  View a graphical representation of the contents of a CDS source file, with the option to edit the source code as text with the syntax elements highlighted for easier visual scanning.

  Right-click a CDS document: *Open With* > *Graphical Editor*

CDS Text Editor

SAP Web IDE for SAP HANA includes a dedicated editor that you can use to define data-persistence objects using the CDS syntax. SAP HANA recognizes the `.hdbcds` file extension required for CDS object definitions and calls the appropriate repository plug-in. If you double-click a file with the `.hdbcds` extension in the *Project Explorer* view, SAP Web IDE for SAP HANA automatically displays the selected file in the CDS text editor.

The CDS editor provides the following features:

- Syntax highlights
  The CDS DDL editor supports syntax highlighting, for example, for keywords and any assigned values. To customize the colors and fonts used in the CDS text editor, choose *Tools* > *Preferences* > *Code Editor* > *Editor Appearance* and select a theme and font size.

  > **i Note**
  >
  > The CDS DDL editor automatically inserts the keyword *namespace* into any new DDL source file that you create using the *New* > *CDS Artifact* dialog.
  >
  > The following values are assumed:
  >
  > - *namespace* = `<ProjectName>.<ApplDBModuleName>`
  > - *context* = `<NewCDSFileName>`

- Keyword completion
  The editor displays a list of DDL suggestions that could be used to complete the keyword you start to enter.

  To change the settings, choose *Tools* > *Code Completion* in the toolbar menu.
- Code validity
  The CDS text editor provides syntax validation, which checks for parser errors as you type. Semantic errors are only shown when you build the XS advanced application module to which the CDS artifacts belong; the errors are shown in the console tab.
- Comments
  Text that appears after a double forward slash (`//`) or between a forward slash and an asterisk (`/*...*/`) is interpreted as a comment and highlighted in the CDS editor (for example, `//this is a comment`).

CDS Graphical Editor

The CDS graphical editor provides graphical modeling tools that help you to design and create database models using standard CDS artifacts with minimal or no coding at all. You can use the CDS graphical editor to create CDS artifacts such as entities, contexts, associations, structured types, and so on.

The built-in tools provided with the CDS Graphical Editor enable you to perform the following operations:

- Create CDS files (with the extension `.hdbcds`) using a file-creation wizard.
- Create standard CDS artifacts, for example: entities, contexts, associations (to internal and external entities), structured types, scalar types, ...
- Define technical configuration properties for entities, for example: indexes, partitions, and table groupings.
- Generate the relevant CDS source code in the text editor for the corresponding database model.
- Open in the CDS graphical editor data models that were created using the CDS text editor.

> → Tip
>
> The built-in tools included with the CDS Graphical Editor are context-sensitive; right-click an element displayed in the CDS Graphical editor to display the tool options that are available.

Related Information

Getting Started with the CDS Graphical Editor [page 1095]

5.3.4.2.2 CDS Documents in XS Advanced

CDS documents are design-time source files that contain DDL code that describes a persistence model according to rules defined in Core Data Services.

In general, CDS works in XS advanced (HDI) in the same way that it does in the SAP HANA XS classic Repository, which is described in the *SAP HANA Developer Guide for SAP HANA Studio* or the *SAP HANA Developer Guide for SAP HANA Web Workbench* listed in the related links below. For XS advanced, however, there are some incompatible changes and additions, which are described in the following sections:

- General overview [page 318]
- Name Space [page 319]
- `@<annotations>` [page 319]
- Entity definitions [page 321]
- Structured Types [page 324]

> ℹ Note
>
> The following example of a CDS document for XS advanced is incomplete `[...]`; it is intended for illustration purposes only.

Sample Code

```
context MyModel {
 type BusinessKey : String(10);
 type SString  : String(40);
 type MyStruct
 {
   aNumber     : Integer;
   aText       : String(80);
   anotherText : MyString80;  // defined in a separate type
 };
 table type Structure2 { ... };
 context MasterData {
    Entity Addresses {
        key  AddressId: BusinessKey;
        City: SString;
        PostalCode: BusinessKey;
        <[...]>
    } technical configuration {
      column store;
      index MyIndex1 on (a, b) asc;
      unique index MyIndex2 on (c, s) desc;
      partition by hash (id) partitions 2,
          range (a) (partition 1 <= values < 10, partition values = 10,
partition others);
      group type Foo group subtype Bar group name Wheeeeezz;
      unload priority <integer_literal>;
    };
 };
 context Purchases {
 <[...]>
 };
};
```

General Overview

In XS advanced, CDS documents must have the file suffix `.hdbcds`, for example, `MyCDSDocument.hdbcds`. Each CDS document must contain the following basic elements:

- CDS artifact definitions
  The objects that make up your persistence model, for example: contexts, entities, structured types, and views

For XS advanced applications, the CDS document does not require a `namespace` declaration, and some of the `@<Annotations>` (for example, `@Schema` or `@Catalog`) are either not required or are no longer supported. Instead, most of the features covered by the `@<Annotations>` in XS classic can now be defined in the `technical configuration` section of the entity definition or in the view definition.

→ Tip

From SAP HANA 2.0 SPS 01, it is possible to define **multiple** top-level artifacts (for example, contexts, entities, etc.) in a single CDS document. For this reason, you can choose any name for the CDS source file; there is no longer any requirement for the name of the CDS source file to be the same as the name of the top-level artifact.

Multiple top-level artifacts are now allowed in a single CDS document, as illustrated in the following example:

Sample Code

Multiple Top-Level Artifacts in a CDS Document

```
type T : Integer;
entity E {
  key id : Integer;
  a : String(20);
};
context ctx1 {
  ...
};
context ctx2 {
  ...
};
```

Name Spaces

From SPS12, the declaration of a name space in a CDS document for XS advanced usage is optional.

Note

You can only omit the name-space declaration in a CDS document if no name space is defined in the corresponding HDI container-configuration file (`.hdinamespace`).

@annotations in CDS Documents

When you are defining CDS models for XS advanced application, bear in mind that there are restrictions on the annotations that you can use. In XS advanced, the `technical configuration` section of the corresponding entity definition CDS document is used to define much of what previously was defined in annotations. This section indicates which annotations are (or are not) allowed in CDS documents in XS advanced.

Supported CDS Annotations in XS Advanced

The following table lists the annotations that **are** supported in CDS models for XS advanced:

Supported CDS Annotations in SAP HANA XS Advanced

CDS Annotation	XS Advanced (HDI)
@OData.publish	Expose a CDS context or nested context (and any artifacts the context includes) as an OData version 4 service.

Note

For more information about publishing CDS contexts as OData services, see *Related Links*.

Annotations in Parameter Definitions in CDS Views

In a CDS view, parameter definitions can be annotated in the same way as any other artifact in CDS; the annotations must be prepended to the parameter name. Multiple annotations are separated either by whitespace or new-line characters.

> → Tip
>
> To improve readability and comprehension, it is recommended to include only one annotation assignment per line.

In the following example, the view `TargetWindowView` selects from the entity `TargetEntity`; the annotation `@positiveValuesOnly` is not checked; and the `targetId` is required for the `ON` condition in the entity `SourceEntity`.

> Sample Code
>
> Annotation Assignments to Parameter Definitions in CDS Views

```
annotation remark: String(100);

view TargetWindowView with parameters
   @remark: 'This is an arbitrary annotation'
   @positiveValuesOnly: true
   LOWER_LIMIT: Integer
as select from TargetEntity
{
  targetId,
  ....
} where targetId > :LOWER_LIMIT and targetId <= :LOWER_LIMIT + 10;
```

Unsupported CDS Annotations in XS Advanced

The following table lists the annotations that are **not** supported in CDS models for XS advanced:

Unsupported CDS Annotations in SAP HANA XS Advanced

CDS Annotation	XS Advanced (HDI)
@Catalog	To specify an index or table type in XS advanced, use the `technical configuration` section of the corresponding entity definition.
@nokey	In XS advanced, it is possible to define an entity without key elements without using an annotation.
@Schema	In XS advanced, schema handling is performed automatically by the HDI container.
@GenerateTableType	In SAP HANA XS advanced, no SAP HANA table type is generated for a structured type by default. To enforce the generation of an SAP HANA table type in SAP HANA XS advanced, use the keyword `table type` in a type definition instead of the keyword `type`, for example, `table type Structure2 { ... };`

CDS Annotation	XS Advanced (HDI)
@SearchIndex	To define a search index in XS Advanced, use the `technical configuration` section of the corresponding entity definition.
@WithStructuredPrivilegeCheck	In XS advanced, you define the privilege check in the view. `view MyView as select from Foo { <select_list> } <where_groupBy_Having_OrderBy> with structured privilege check;`

Entity Definitions

The definition of an entity can contain a section called `technical configuration`, which you use to define the elements listed in the following table:

Technical Configuration Features for Entities in SAP HANA XS Versions

Configuration Element	XS Advanced
Storage type [page 321]	Yes
Indexes [page 322]	Yes
Full text indexes [page 322]	Yes
Partitioning [page 323]	Yes
Grouping [page 323]	Yes
Unload priority [page 324]	Yes

> ### Note
>
> The syntax in the technical configuration section is as close as possible to the corresponding clauses in the SAP HANA SQL `Create Table` statement. Each clause in the technical configuration must end with a semicolon.

Storage Type

In XS advanced, the `@Catalog.tableType` annotation is not supported; you must use the technical configuration. In the technical configuration for an entity, you can use the `store` keyword to specify the storage type ("`row`" or "`column`") for the generated table, as illustrated in the following example. If no store type is specified, a "column" store table is generated by default.

> ### Sample Code

```
entity MyEntity {
  key id : Integer;
```

```
  a : Integer;
  }
  technical configuration {
    row store;
  };
```

The specification of table **type** is split into separate components. The storage type (`row store` or `column store`) is specified in the technical configuration section of the CDS entity definition; to generate a **temporary** table, use the keyword "`temporary entity`", as illustrated in the following example:

Sample Code

```
temporary entity MyEntity2 {
  ...
} technical configuration {
  row store;
};
```

Indexes

In XS advanced, the `@Catalog.index` annotation is not supported; you must use the technical configuration. In the technical configuration for an entity, you can use the `index` and `unique index` keywords to specify the index type for the generated table. For example: "`asc`" (ascending) or "`desc`" (descending) describes the index order, and `unique` specifies that the index is unique, where no two rows of data in the indexed entity can have identical key values.

Sample Code

```
} technical configuration {
  index MyIndex1 on (a, b) asc;
  unique index MyIndex2 on (c, s) desc;
};
```

Full Text Indexes

In the technical configuration for an entity, you can use the `fulltext index` keyword to specify the full-text index type for the generated table, as illustrated in the following example.

Sample Code

```
entity MyEntity {
  key id : Integer;
  t : String(100);
  s {
    u : String(100);
  };
} technical configuration {
  fulltext index MyFTI1 on (t) <fulltext_parameter_list>;
  fuzzy search index on (s.u);
};
```

The `<fulltext_parameter_list>` is identical to the standard SAP HANA SQL syntax for `CREATE FULLTEXT INDEX`. A `fuzzy search index` in the technical configuration section of an entity definition corresponds to the `@SearchIndex` annotation in XS classic and the statement "`FUZZY SEARCH INDEX ON`" for a table column in SAP HANA SQL. It is not possible to specify both a full-text index and a fuzzy search index for the same element.

! Restriction

In XS advanced, the `@SearchIndex` annotation is not supported; you must use the technical configuration to define which of the columns should be indexed for search capabilities. In XS classic, it is not possible to use both the `@SearchIndex` annotation **and** the technical configuration (for example, `fulltext index`) at the same time. In addition, the full-text parameters `CONFIGURATION` and `TEXT MINING CONFIGURATION` are not supported.

Partitioning

In the technical configuration for an entity, you can use the `partition by` clause to specify the partitioning information for the generated table, as illustrated in the following example.

! Restriction

The `partition by` clause is only supported in XS advanced.

Sample Code

```
entity MyEntity {
  key id : Integer;
  a : Integer;
} technical configuration {
  partition by hash (id) partitions 2,
               range (a) (partition 1 <= values < 10, partition values = 10,
partition others);
};
```

The syntax in the `partition by` clause is identical to the standard SAP HANA SQL syntax for the `PARTITION BY` expression in the `HANA SQL CREATE TABLE` statement.

Grouping

In the technical configuration for an entity, you can use the `group` clause to specify the partitioning information for the generated table, as illustrated in the following example.

! Restriction

The `group` clause is only supported in XS advanced.

Sample Code

```
entity MyEntity {
  key id : Integer;
  a : Integer;
} technical configuration {
  group type Foo group subtype Bar group name Wheeeeezz;
};
```

The syntax in the `group` clause is identical to the standard SAP HANA SQL syntax for the `GROUP OPTION` expression in the `HANA SQL CREATE TABLE` statement.

Unload Priority

In the technical configuration for an entity, you can use the `Unload Priority` clause to specify the priority for unloading the generated table from memory, as illustrated in the following example:

> **! Restriction**
>
> The `unload priority` clause is only supported in XS advanced.

> **Sample Code**

```
entity MyEntity {
  <element_list>
} technical configuration {
  unload priority <integer_literal>;
};
```

The syntax in the `unload priority` clause is identical to the standard SAP HANA SQL syntax for the `UNLOAD PRIORITY` expression in the `HANA SQL CREATE TABLE` statement.

Structured Types

In the SAP HANA XS classic model, for each structured CDS type, an SAP HANA table type is generated by default in the repository. For this reason, in XS classic, the generation of table types could be controlled explicitly by means of the `@GenerateTableType` annotation. In SAP HANA XS advanced, however, no SAP HANA table type is generated for a structured type by default.

In SAP HANA XS advanced, to enforce the generation of an SAP HANA table type you must use the keyword `table type` instead of the keyword `type`, as illustrated in the following example:

> **! Restriction**
>
> The `table type` keyword is only supported in SAP HANA XS advanced.

```
type Struc1 { ... };       // no table type is generated
table type Struc2 { ... }; // creates a corresponding table type
```

You can define structured types that do not contain any elements, for example, using the keywords `type EmptyStruct { };`. In the example, below the generated table for entity “E” contains only one column: “a”.

> **→ Tip**
>
> It is not possible to generate an SAP HANA table type for an empty structured type.

```
type EmptyStruct { };
entity E {
  a : Integer;
  s : EmptyStruct;
};
```

Related Information

Create a CDS Document (XS Advanced) [page 313]
Create the Data Persistence Artifacts with CDS in XS Advanced [page 311]
Defining OData v4 Services for XS Advanced Java Applications [page 580]

5.3.4.2.3 External CDS Artifacts in XS Advanced

You can define an artifact in one CDS document by referring to an artifact that is defined in another CDS document.

The CDS syntax enables you to define a CDS artifact in one document by basing it on an "external" artifact - an artifact that is defined in a separate CDS document. Each external artifact must be explicitly declared in the source CDS document with the `using` keyword, which specifies the location of the external artifact, its name, and where appropriate its CDS context.

> → Tip
>
> The `using` declarations must be located in the header of the CDS document between the `namespace` declaration and the beginning of the top-level artifact, for example, the `context`.

The external artifact can be either a single object (for example, a type, an entity, or a view) or a context. You can also include an optional alias in the `using` declaration, for example, `ContextA.ContextA1 as ic`. The alias (`ic`) can then be used in subsequent type definitions in the source CDS document.

> ! Restriction
>
> For SAP HANA XS advanced deployments, the file name of a design-time CDS artifact must use the extension `.hdbcds`, for example, `ContextA.hdbcds` and `ContextB.hdbcds`.

```
//Filename = Pack1/Distributed/ContextB.hdbcds
namespace Pack1.Distributed;
using Pack1.Distributed::ContextA.T1;
using Pack1.Distributed::ContextA.ContextAI as ic;
using Pack1.Distributed::ContextA.ContextAI.T3 as ict3;
using Pack1.Distributed::ContextA.ContextAI.T3.a as a;  // error, is not an
artifact
context ContextB {
  type T10 {
    a : T1;              // Integer
    b : ic.T2;           // String(20)
    c : ic.T3;           // structured
    d : type of ic.T3.b; // String(88)
    e : ict3;            // structured
    x : Pack1.Distributed::ContextA.T1;  // error, direct reference not allowed
  };
  context ContextBI {
    type T1 : String(7);  // hides the T1 coming from the first using declaration
    type T2 : T1;         // String(7)
  };
};
```

The CDS document `ContextB.hdbcds` shown above uses external artifacts (data types `T1` and `T3`) that are defined in the "target" CDS document `ContextA.hdbcds` shown below. Two `using` declarations are present

in the CDS document `ContextB.hdbcds`; one with no alias and one with an explictly specified alias (`ic`). The first `using` declaration introduces the scalar type `Pack1.Distributed::ContextA.T1`. The second `using` declaration introduces the context `Pack1.Distributed::ContextA.ContextAI` and makes it accessible by means of the explicitly specified alias `ic`.

> **i Note**
>
> If no explicit alias is specified, the last part of the fully qualified name is assumed as the alias, for example `T1`.

The `using` keyword is the only way to refer to an externally defined artifact in CDS. In the example above, the type x would cause an activation error; you cannot refer to an externally defined CDS artifact directly by using its fully qualilfed name in an artifact definition.

You can use the "`using`" keyword to reference the following SAP HANA artifacts in XS advanced:

- CDS DDL documents (specify which document is used)
- CDS DCL documents (CDS access policy documents)
- Calculation views (`.hdbcalculationview`)
- SAP HANA entities (`.hdbcds`)
- SAP HANA user role definitions (`.hdbrole`)

```
//Filename = Pack1/Distributed/ContextA.hdbcds
namespace Pack1.Distributed;
context ContextA {
  type T1 : Integer;
  context ContextAI {
    type T2 : String(20);
    type T3 {
      a : Integer;
      b : String(88);
    };
  };
};
```

> **i Note**
>
> Whether you use a single or multiple CDS documents to define your data-persistence model, each CDS document must contain only **one** top-level artifact, and the name of the top-level artifact must correspond to the name of the CDS document. For example, if the top-level artifact in a CDS document is `ContextA`, then the CDS document itself must be named `ContextA.hdbcds`.

Related Information

Defining the Data Model in XS Advanced [page 233]
Create a CDS Document (XS Advanced) [page 313]

5.3.4.2.4 CDS Naming Conventions in XS Advanced

Rules and restrictions apply to the names of CDS documents and the package in which the CDS document resides.

The rules that apply for naming CDS documents are the same as the rules for naming the packages in which the CDS document is located. When specifying the name of a package or a CDS document (or referencing the name of an existing CDS object, for example, within a CDS document), bear in mind the following rules:

- CDS source-file name
  From SAP HANA 2.0 SPS 01, it is possible to define **multiple** top-level artifacts (for example, contexts, entities, etc.) in a single CDS document. For this reason, you can choose any name for the CDS source file; there is no longer any requirement that the name of the CDS source file must be the same as the name of a top-level artifact.
- File suffix
  The file suffix differs according to SAP HANA XS version:
  - XS classic
    `.hdbdd`, for example, `MyModel.hdbdd`
  - XS advanced
    `.hdbcds`, for example, `MyModel.hdbcds`
- Permitted characters
  CDS object and package names can include only a subset of ASCII, namely, the following characters:
  - Lower or upper case letters (aA-zZ) and the underscore character (_)
  - Digits (0-9)
- Forbidden characters
  The following restrictions apply to the characters that you can use (and their position) in the name of a CDS artifact or a package that contains CDS artifacts:
  - The hyphen (-) or the dot (.) are **not** permitted in the name of a CDS document, for example:
    `art-if.act.hdbcds`
  - A digit (0-9) is **not** permitted as the first character in the name of either a CDS document or a package that contains CDS documents, for example:
    - XS classic
      `2CDSobjectname.hdbdd`
    - XS advanced
      `acme.com.1package.hdbcds`
  - The CDS parser does not recognize either CDS document names or package names that consist **exclusively** of digits, for example:
    - XS classic
      `1234.hdbdd`
    - XS advanced
      `acme.com.999.hdbcds`
- Special considerations for artifact names in HDI/XS advanced
  Additional restrictions apply to the names of files and folders in the HDI virtual file system. For more details, see *Related Information*.

> ⚠ Caution
>
> Although it is possible to use quotation marks (“”) to wrap a name that includes forbidden characters, as a general rule, it is recommended to follow the naming conventions for CDS documents specified here in order to avoid problems during activation in the catalog.

Related Information

Create a CDS Document (XS Advanced) [page 313]
CDS Documents in XS Advanced [page 317]
Naming Conventions in SAP HDI [page 236]

5.3.4.2.5 Accessing CDS Metadata in HDI

CDS metadata is available as standard SQL monitoring views and table functions.

As of SAP HANA 1.0 SP11, any CDS catalog metadata, which was created by the CDS plug-in of the HANA Deployment Infrastructure (HDI), `com.sap.hana.di.cds`, is available as standard SQL monitoring views (CDS_*) and one table function (CDS_ARTIFACT_DEFINITION). In the same was as any SAP HANA SQL metadata, the views listing CDS metadata are available in schema SYS. The exposure of CDS metadata by means of SQL views enables consumers to combine CDS and database metadata with the SQL JOIN command.

The CDS catalog metadata enables you to reconstruct a CDS source; if the reconstructed source is processed again by the CDS compiler, it produces the same metadata even if the recreated CDS source is not identical to the original source. The result set structure of the main CDS metadata access point, for example, the table function “CDS_ARTIFACT_DEFINITION” is be minimal. If necessary, it can be joined with database catalog system views and views containing other CDS related metadata in order to enrich the result set with additional, more specific metadata.

> ℹ Note
>
> It is very important to control the run time access to the CDS metadata. Although the user has access to the CDS metadata SQL objects (in schema SYS), instance-based access control ensures that the metadata content exposed to a user is only that which the requesting user is authorized to see.

The instance based CDS metadata access control is managed at the HDI run time container schema level by means of a newly introduced schema level object privilege “SELECT CDS METADATA”. Since the HDI run time container schema is created and owned by an HDI internal technical user, the CDS metadata privilege cannot be assigned using the usual GRANT command. Instead, the granting must be performed using an built-in administrative stored procedure “GRANT_CONTAINER_SCHEMA_PRIVILEGES”; the stored procedure is provided by SAP HANA HDI in a schema which is related to the actual HDI run time container schema (for example, named `<HDI_RT_Container_Name>#DI`).

→ Tip

You can also use HDI container tools to granting the privilege “SELECT CDS METADATA” to an HDI run time container schema.

Related Information

CDS Catalog Reader API for HDI [page 329]
Managing SAP HDI Containers and Artifacts [page 234]

5.3.4.2.6 CDS Catalog Reader API for HDI

The new SQL-based CDS metadata access API in schema SYS is only available CDS content created for (and deployed in) the SAP HANA HDI.

From SAP HANA 1.0 SPS 11, any CDS catalog metadata that is created by the CDS plugin of the HANA Deployment Infrastructure (HDI), `com.sap.hana.di.cds`, is available as standard SQL monitoring views (`CDS_*`) and one table function (`CDS_ARTIFACT_DEFINITION`). In the same way as HANA SQL metadata, these views are available in schema SYS. The exposure of CDS catalog metadata in SQL views enables consumers to combine CDS and database metadata by means of the SQL JOIN command.

CDS Catalog Metadata Views

View Name	Description
CDS_ARTIFACT_DEFINITION	Retrieves CDS metadata for a given CDS artifact name
CDS_ARTIFACT_NAMES	Exposes all CDS artifact names and their kind for all schemas
CDS_ANNOTATION_VALUES	Exposes multiple rows, if a single CDS artifact is annotated with multiple annotation values
CDS_ANNOTATION_ASSIGNMENTS	Returns a flat list of annotation assignments
CDS_ASSOCIATIONS	Expose association metadata and definitions
CDS_ENTITIES	Expose entity (table) metadata and definitions
CDS_VIEWS	Expose view metadata and definitions

CDS_ARTIFACT_DEFINITION

`CDS_ARTIFACT_DEFINITION` is intended to be the main entry point to retrieve CDS metadata for a given CDS artifact name. If no CDS artifact name is known, the `CDS_ARTIFACT_NAMES` view can be used to retrieve all

artifact names for a given schema. The other CDS_* monitoring views can be accessed individually, too, but are actually intended for use by joining with the result of CDS_ARTIFACT_DEFINITION, to enrich the result with additional, more detailed CDS metadata.

The `CDS_ARTIFACT_DEFINITION TableFunction` expects exactly two mandatory **input** parameters; the parameters specify the requested CDS artifact name in a certain schema; that is, the corresponding HDI runtime container. The input parameters are as follows:

- `SCHEMA_NAME (NVARCHAR(256))`
- `ARTIFACT_NAME (NVARCHAR(127))`

Result-Set Structure

Column Name	Type	Description
SCHEMA_NAME	NVARCHAR(256)	The HDI runtime container schema name. The returned schema name always matches the schema name requested when calling the `TableFunction`. Since HDI allows you to deploy the same CDS artifact name into the same namespace in two different runtime container schemas, it is necessary to specify the schema name as part of the ON-condition of SQL JOINS, if additional metadata is selected from additional monitoring views.
ARTIFACT_NAME	NVARCHAR(127)	The fully qualified CDS artifact name, including the namespace, separated by a double colon "`::`", for example, `namespaceComponent1.namespaceComponent1::ContextName.EntityName`
ELEMENT_NAME	NVARCHAR(127)	CDS element name is populated with the element name, if `ARTIFACT_KIND` is "`ELEMENT`". The name can also be a path with the name components separated by a double colon "`::`", where flattened elements of (nested) structures are included.
ARTIFACT_KIND	VARCHAR(32)	The CDS artifact kind, for example, one of the following enumeration values: • ANNOTATION • ARRAY_TYPE • ASSOCIATION • ASSOCIATION_ELEMENT • CONTEXT • DERIVED_TYPE • ELEMENT • ENTITY • ENUM • STRUCTURED_TYPE • VIEW

Column Name	Type	Description
PARENT_ARTIFACT_NAME	NVARCHAR(127)	The name of the parent CDS artifact. NULL for top-level root artifacts. → Tip The parent relationship is meant in terms of structural definition; not the usage dimension.
PARENT_ARTIFACT_ELEMENT	NVARCHAR(127)	The name of the parent CDS element. The PARENT_ARTIFACT_ELEMENT column has a value in the following cases: 1. Anonymous, non-primitive type definitions "behind" elements have the element as parent artifact, for example: "`elem: array of String(3);`". The array type definition has the "`elem`" artifact as parent. The row representing the element definition refers to the anonymous array type definition in the USED_ARTIFACT_NAME column. 2. For flattened elements, non-top-level elements refer to their parent elements, which were also created as part of the flattening process.
IS_ANONYMOUS	VARCHAR(5)	Specifies whether the artifact definition is anonymous: TRUE or FALSE (Default). Artifact definitions in CDS can be anonymous in the sense that there is no identifier specified for them in the source. i Note "anonymous" artifact definitions are internally still supplied with a unique artifact name, which is constructed by the convention: Parent artifact name, concatenated with "." and the relative local component name. Elements are always syntactically named in the CDS source and are therefore always represented as non-anonymous, even if an enclosing structured type definition itself might be anonymous. The same is true for the foreign key elements of managed associations, which are represented with artifact kind ASSOCIATION_ELEMENT.
USED_ARTIFACT_SCHEMA	NVARCHAR(256)	The schema name of the used CDS artifact. (See also column SCHEMA_NAME). The triplet USED_ARTIFACT_SCHEMA, NAME, and KIND is similar to OBJECT_SCHEMA, NAME, and TYPE in SYS.SYNONYMS; it defines a reference to another CDS artifact definition, which is used. These three columns are only filled for usage relationships, not for nested definitions.

Column Name	Type	Description
USED_ARTIFACT_NAME	NVARCHAR(127)	The name of the used CDS artifact. If CDS or SAP HANA database (built-in) primitive types are used, the USED_ARTIFACT_NAME is filled and the kind is PRIMITIVE_TYPE. No value is given for the SCHEMA because CDS built-in types are virtual and do not exist in a actual database schema. **i Note** The list of available CDS primitive types is described in the *SAP HANA Core Data Services (CDS) Reference* available on the SAP Help Portal. CDS built-in primitive types are prefixed with namespace "`sap.cds::`". SAP HANA types that are usable but deprecated within CDS are located in the top-level context "hana" (prefixed with "hana"). If explicitly defined custom CDS artifacts are used, the USED_ARTIFACT_KIND values will be one of the following: DERIVED_TYPE, ANNOTATION, ARRAY, CONSTANT, or ENUM.
USED_ARTIFACT_KIND	VARCHAR(32)	The kind of CDS artifact used
ORDINAL_NUMBER	INTEGER	The ordinal position of the CDS artifact definition within the returned artifact tree for the requested artifact name. The ordinal number starts with 0 and is increased in pre-order manner during depth-first traversal of the requested artifact tree. **i Note** The CDS ordinal number must not be confused with the column order of tables at the database level. The user of the API can JOIN the POSITION information of the SYS.COLUMNS view to prefix the column position in the generated database object with the CDS ORDINAL_NUMBER.
SQL_DATA_TYPE_NAME	VARCHAR(16)	The SQL data type name, which is only filled for elements with built-in CDS primitive types. See also column DATA_TYPE_NAME in SYS.COLUMNS view.
TYPE_PARAM_1	INTEGER	The number of characters for character types; maximum number of digits for numeric types; number of bytes for LOB types; SRID value for GIS (geospatial) types. **i Note** Internal values for time and timestamp data type are not exposed. If such information is required, the user of the CDS metadata SQL API has to JOIN the CDS element row with the SYS.COLUMNS view.
TYPE_PARAM_2	INTEGER	The numeric types: the maximum number of digits to the right of the decimal point

Column Name	Type	Description
IS_NULLABLE	VARCHAR(5)	Specifies whether the database column generated for this CDS element is allowed to accept null values. Set to TRUE (default) or FALSE. See also column IS_NULLABLE in SYS.COLUMNS view.
IS_KEY	VARCHAR(5)	Specifies whether the column is part of the primary key, in order of the definition in the source. Set to TRUE or FALSE (default). The order of the columns in the key in given by the order of the elements in the Entity. The key information is also available for the flattened elements, for structured-type usage for elements that are part of the key.
VALUE	NVARCHAR(5000)	Default value in case of `ARTIFACT_KIND = 'ELEMENT'` and constant value in case of `ARTIFACT_KIND = 'CONSTANT'`.
AUX_ELEMENT_INFO	NVARCHAR(5000)	This column exposes the following auxiliary element related properties: • Original name path if used with "as" for foreign key element definitions of managed associations (ARTIFACT_KIND = ASSOCIATION_ELEMENT) • The period element kind in case of series entities: PERIOD or ALTERNATE_PERIOD

CDS_ARTIFACT_NAMES

This view exposes all CDS artifact names and their kind for all schemas the user has the `SELECT CDS METADATA` privilege for. It is intended for use with a "`WHERE`" condition, for example, to restrict the result set to **all** artifacts defined within one single schema (HDI runtime container) or CDS namespace.

Result-Set Structure

Column Name	Type	Description
SCHEMA_NAME	NVARCHAR(256)	The name of the SAP HANA HDI run time container schema
ARTIFACT_NAME	NVARCHAR(127)	The fully qualified CDS artifact name, including the namespace, separated by a double colon "`::`"

Column Name	Type	Description
ARTIFACT_KIND	VARCHAR(32)	The artifact kind, for example, one of the following enumeration values: • ANNOTATION • ARRAY_TYPE • ASSOCIATION • CONTEXT • DERIVED_TYPE • ENTITY • ENUM • STRUCTURED_TYPE • VIEW

CDS_ANNOTATION_VALUES

The view `SYS_RT.CDS_ANNOTATION_VALUES` exposes multiple rows if a single CDS artifact is annotated with multiple annotation values. This allows you to select all artifact names that are annotated with a certain annotation. The values of CDS internal annotations that are prefixed with namespace "`sap.cds::`" are not exposed with this view.

Result-Set Structure

Column Name	Type	Description
SCHEMA_NAME	NVARCHAR(256)	The name of the SAP HANA HDI run time container schema
ARTIFACT_NAME	NVARCHAR(127)	The name of the annotated CDS artifact
ELEMENT_NAME	NVARCHAR(127)	Element name, if an element is annotated
ANNOTATION_SCHEMA_NAME	NVARCHAR(256)	Schema name of annotation definition
ANNOTATION_NAME	NVARCHAR(127)	Name of the annotation definition **i Note** For built-in CDS annotations, no entry can be found in SYS.CDS_ANNOTATIONS. This is because the definitions of built-in CDS annotations are not delivered as CDS catalog content; they are defined during compiler initialization.
VALUE	NCLOB	The annotation value tree, serialized as JSON, for example: `{ "value": "<SomeAnnotationValue>" }`

CDS_ANNOTATION_ASSIGNMENTS

This view in the CDS catalog API returns a flat list of annotation assignments:

Sample Code

```
select * from SYS.CDS_ANNOTATION_ASSIGNMENTS
```

The view `SYS_RT.CDS_ANNOTATION_VALUES` exposes multiple rows if a single CDS artifact is annotated with multiple annotation values. This allows you to select all artifact names that are annotated with a certain annotation. The values of CDS internal annotations that are prefixed with namespace "`sap.cds::`" are not exposed with this view.

Result-Set Structure

Column Name	Description
SCHEMA_NAME	The schema of the annotated object
ARTIFACT_NAME	The name of the annotated CDS object
COMPONENT_NAME	The name of the annotated element (if the annotation is assigned to an element of an object)
ANNOTATION_NAME	The name of the annotation
EXTENSION_PACKAGE_NAME	The name of the extension package, if the assignment happens in an extension
FORMAT	The format of the annotation value; possible values for FORMAT are: `Enum`, `Boolean`, `String`, `Number`, `Array`, and `Other`. **Note** Values for array-like annotations are always returned as "blob"; flattening stops at arrays.
VALUE	The annotation value
VALIDATION_RESULT	Possible values for validation results are: • CORRECT A definition exists for the annotation, and the assignment matches the definition. • MISMATCHED A definition exists for the annotation, and the assignment **does not** match the definition. • UNCHECKED No definition exists for the annotation.
DEFINITION_SCHEMA_NAME	The schema of the annotation definition (if a definition exists)
DEFINITION_NAME	The name of annotation definition (if a definition exists)

The following is an example of a CDS source:

```
annotation KnownAnno {
  a : Integer;
  b : String(100);
  c : Integer;
  t : UTCTimestamp;
};
@KnownAnno : { a:1 , b:'some value' , c:'no int' , t:'2017-03-02 10:33'}
type T : Integer;
@ArrayAnno : [1,2,3]
type S {
  x : Integer;
  @UnknownAnno : { c:3 , d:'hi there' , e:true}
  y : Integer;
}
```

The following table shows the resulting annotation assignments:

> **i Note**
>
> The contents of the columns SCHEMA_NAME and DEFINITION_SCHEMA_NAME are not shown. SCHEMA_NAME always shows the name of the schema of the HDI container where the model is deployed. This is also true for DEFINITION_SCHEMA_NAME, if the annotation has a definition.

Annotation Assignments 1

ARTIFACT_NAME	COMPONENT_NAME	ANNOTATION_NAME	FORMAT
S		ArrayAnno	Array
S	y	UnknownAnno.c	Number
S	y	UnknownAnno.d	String
S	y	UnknownAnno.e	Boolean
T		KnownAnno.a	Number
T		KnownAnno.b	String
T		KnownAnno.t	String
T		KnownAnno.c	String

Annotation Assignments 2

ARTIFACT_NAME	VALUE	VALIDATION_RESULT	DEFINITION_NAME
S	[1,2,3]	UNCHECKED	
S	3	UNCHECKED	
S	hi there	UNCHECKED	
S	true	UNCHECKED	

ARTIFACT_NAME	VALUE	VALIDATION_RESULT	DEFINITION_NAME
T	1	CORRECT	KnownAnno
T	some value	CORRECT	KnownAnno
T	2017-03-02 10:33	CORRECT	KnownAnno
T	no int	MISMATCHED	KnownAnno

CDS_ASSOCIATIONS

Association definitions are already exposed as part of the `CDS_ARTIFACT_DEFINITION` result set, represented by `ARTIFACT_KIND "ASSOCIATION"`. Foreign key elements are also available with kind `ASSOCIATION_ELEMENT`, which allows then to be distinguished from "normal" elements.

The CDS association metadata can be retrieved with the following SQL SELECT statement:

Sample Code

```
SELECT * FROM CDS_ARTIFACT_DEFINITION('<SomeSchemaName>',
'<SomeCdsArtifactName>') def
LEFT OUTER JOIN SYS.CDS_ASSOCIATIONS assoc
ON def.SCHEMA_NAME = assoc.SCHEMA_NAME AND def.ARTIFACT_NAME =
assoc.ASSOCIATION_NAME
WHERE def.ARTIFACT_KIND = 'ASSOCIATION' OR def.ARTIFACT_KIND =
'ASSOCIATION_ELEMENT';
```

Result-Set Structure

Column Name	Type	Description
SCHEMA_NAME	NVARCHAR(256)	The name of the SAP HANA HDI run time container schema
ASSOCIATION_NAME	NVARCHAR(127)	The name of the CDS entity
ASSOCIATION_KIND	NVARCHAR(32)	One of the following supported association-specific enumeration values: • UNMANAGED • FOREIGN_KEY_EXPLICIT • FOREIGN_KEY_IMPLICIT
TARGET_ARTIFACT_SCHEMA_NAME	NVARCHAR(127)	Name of the schema for the associated CDS artifact.

Column Name	Type	Description
TARGET_ARTIFACT_NAME	NVARCHAR(127)	Name of the associated CDS artifact **! Restriction** Only CDS entities are allowed as the target of an association.
TARGET_CARDINALITY_MIN	INTEGER	Minimum cardinality of association target. Default is 0.
TARGET_CARDINALITY_MAX	INTEGER	Maximum cardinality of association target. Default is 1. -1 represents unlimited.
JOIN_CONDITION	NCLOB	The join condition for unmanaged associations (ASSOCIATION_KIND = UNMANAGED)

CDS_ENTITIES

Result-Set Structure

Column Name	Type	Description
SCHEMA_NAME	NVARCHAR(256)	The name of the SAP HANA HDI run time container schema
ENTITY_NAME	NVARCHAR(127)	The name of the CDS entity
SERIES_KIND	NVARCHAR(32)	One of the following supported series-kind enumeration values: • NO_SERIES • NOT_EQUIDISTANT, EQUIDISTANT • EQUIDISTANT_PIECEWISE

CDS_VIEWS

Result-Set Structure

Column Name	Type	Description
SCHEMA_NAME	NVARCHAR(256)	The name of the SAP HANA HDI run time container schema
VIEW_NAME	NVARCHAR(127)	The name of the CDS view
DEFINITION	NCLOB	The view definition string. Relative names are resolved to absolute names already and constant expressions to concrete values.

Related Information

Accessing CDS Metadata in HDI [page 328]

5.3.4.2.7 CDS Contexts in XS Advanced

You can define multiple CDS-compliant elements (for example, entities or views) in a single file by assigning them to a *context*.

The following example illustrates how to assign two simple entities to a context using the CDS-compliant syntax; you store the context-definition file with a specific name and the file extension `.hdbcds`, for example, `MyContext.hdbcds`.

> → Tip
>
> From SAP HANA 2.0 SPS 01, it is possible to define **multiple** top-level artifacts (for example, contexts, entities, etc.) in a single CDS document. For this reason, you can choose any name for the CDS source file; there is no longer any requirement that the name of the CDS source file must be the same as the name of a top-level artifact.

In the example below, you must save the context definition "Books" in the CDS document `Books.hdbcds`. In XS advanced, the name space declaration is optional.

The following code example illustrates how to use the CDS syntax to define multiple design-time entities in a context named `Books`.

```
namespace com.acme.myapp1;
@OData.publish : true      //OData v4 only
context Books {
    entity Book {
        key AuthorID : String(10);
        key BookTitle : String(100);
        ISBN : Integer not null;
        Publisher : String(100);
    } technical configuration {
        column store;
        unique index MYINDEX1 on (ISBN) desc;
    };
    entity Author {
        key AuthorName : String(100);
        AuthorNationality : String(20);
        AuthorBirthday : String(100);
        AuthorAddress : String(100);
    } technical configuration {
        column store;
        unique index MYINDEX1 on (AuthorNationality) desc;
    };
};
```

Activation of the file `Books.hdbcds` containing the context and entity definitions creates the catalog objects "Book" and "Author".

> ℹ Note
>
> The namespace specified at the start of the file, for example, `com.acme.myapp1` corresponds to the location of the entity definition file (`Books.hdbcds`) in the application-package hierarchy. The namespace

is not required in XS advanced scenarios; it is optional and provided only for backwards compatibility with XS classic.

This section includes details of the following concepts and features:

- Nested Contexts [page 340]
- Name Resolution Rules [page 341]
- OData Services [page 342]

Nested Contexts

The following code example shows you how to define a nested context called `InnerCtx` in the parent context `MyContext`. The example also shows the syntax required when making a reference to a user-defined data type in the nested context, for example, (`field6 : type of InnerCtx.CtxType.b;`).

The `type of` keyword is only required if referencing an element in an entity or in a structured type; types in another context can be referenced directly, without the `type of` keyword. The nesting depth for CDS contexts is restricted by the limits imposed on the length of the database identifier for the name of the corresponding SAP HANA database artifact (for example, table, view, or type); this is currently limited to 126 characters (including delimiters).

> **i Note**
>
> The context itself does not have a corresponding artifact in the SAP HANA catalog; the context only influences the names of SAP HANA catalog artifacts that are generated from the artifacts defined in a given CDS context, for example, a table or a structured type.

```
namespace com.acme.myapp1;
context MyContext {
    // Nested contexts
    context InnerCtx {
        Entity MyEntity {
            key id : Integer;
            // <...>
        };
        Type CtxType {
            a : Integer;
            b : String(59);
        };
    };
    type MyType1 {
        field1 : Integer;
        field2 : String(40);
        field3 : Decimal(22,11);
        field4 : Binary(11);
    };
    type MyType2 {
        field1 : String(50);
        field2 : MyType1;
    };
    type MyType3 {
        field1 : UTCTimestamp;
        field2 : MyType2;
    };
    entity MyEntity1 {
        key id : Integer;
```

```
        field1 : MyType3 not null;
        field2 : String(24);
        field3 : LocalDate;
        field4 : type of field3;
        field5 : type of MyType1.field2;
        field6 : type of InnerCtx.CtxType.b; // refers to nested context
        field7 : InnerCtx.CtxType;           // more context references
    } technical configuration {
        unique index IndexA on (field1) asc;
    };
};
```

Name Resolution Rules

The sequence of definitions inside a block of CDS code (for example, `entity` or `context`) does not matter for the scope rules; a binding of an aritfact type and name is valid within the confines of the smallest block of code containing the definition, except in inner code blocks where a binding for the same identifier remains valid. This rules means that the definition of `nameX` in an inner block of code hides any definitions of `nameX` in outer code blocks.

> **i** Note
>
> An identifier may be used before its definition without the need for forward declarations.

```
context OuterCtx
{
  type MyType1 : Integer;
  type MyType2 : LocalDate;
  context InnerCtx
  {
    type Use1 : MyType1;           // is a String(20)
    type Use2 : MyType2;           // is a LocalDate
    type MyType1 : String(20);
  };
   type invalidUse : Use1;       // invalid: Use1 is not
                                 // visible outside of InnerCtx
   type validUse   : InnerCtx.Use1;   // ok
};
```

No two artifacts (including namespaces) can be defined whose absolute names are the same or are different only in case (for example, `MyArtifact` and `myartifact`), even if their artifact type is different (entity and view). When searching for artifacts, CDS makes no assumptions which artifact kinds can be expected at certain source positions; it simply searches for the artifact with the given name and performs a final check of the artifact type.

The following example demonstrates how name resolution works with multiple nested contexts, Inside context `NameB`, the local definition of `NameA` shadows the definition of the context `NameA` in the surrounding scope. This means that the definition of the identifier `NameA` is resolved to `Integer`, which does not have a sub-component `T1`. The result is an error, and the compiler does not continue the search for a “better” definition of `NameA` in the scope of an outer (parent) context.

```
context OuterCtx
{
  context NameA
  {
    type T1 : Integer;
```

```
    type T2 : String(20);
  };
  context NameB
  {
    type NameA : Integer;
    type Use   : NameA.T1;  // invalid: NameA is an Integer
    type Use2  : OuterCtx.NameA.T2;  // ok
  };
};
```

Publishing OData Services

You can use the Boolean `@OData.publish` annotation at the context level to indicate that the annotated CDS Context should be exposed (`true`) as an OData service.

> **! Restriction**
>
> The `@OData.publish` feature only works with OData version 4.

The `@OData.publish` annotation cannot be used to publish individual CDS entities, or CDS views. In addition, the annotation `@OData.publish` annotation cannot be used to publish a CDS context that includes a subcontext.

All contexts defined in a CDS document and annotated with `@OData.publish` are published as OData v4 services. There is no restriction on the number of CDS contexts that can be annotated as `OData.publish : true`. In the following example, the CDS artifacts `MyEntity1` and `MyEntity2` defined in the CDS context `ContextA` are exposed for consumption by OData.

> **Sample Code**
>
> Publish a CDS Context (and Contents) as an OData v4 Service

```
namespace acme.com;
@OData.publish : true
context ContextA {
  MyEntity1 {
    key ID : Integer;
  };
  MyEntity2 {
    key ID : Integer;
    type Address1
  };
};
```

> **i Note**
>
> Although an annotated context that includes a nested (sub) context cannot be published as an OData service, it is possible to publish the individual nested contexts. However, any nested context annotated for OData publication must not include any nested contexts of its own.

In the following example, the CDS artifacts defined in the subcontexts `SubContextA1` and `SubContextA2` are exposed as OData v4 services.

Sample Code

Publish CDS Subcontexts an OData v4 Services

```
namespace acme.com;
context ContextA {
  @OData.publish : true
  context ContextA1 {
    MyEntity3 {
      key ID : Integer;
      name : String(80);
      number : Integer;
    };
    MyEntity4 {
      key ID : Integer;
      name : String(80);
    };
  };
  @OData.publish : true
  context ContextA2 {
    MyEntity5 {
      key ID : Integer;
      field1 : Address1;
    };
    type Address1 {
      a : Integer;
      b : String(40);
    };
  };
};
```

Namespaces in OData v4 Service URLs

The name of the OData service created from a CDS context in a specified name space is: `<name.space>._.<context>`. In the OData query, the name space and the context name (which is also the OData service name) are separated by the character combination "._" (dot underscore dot). If no name space is declared in the CDS document, the name space and separator "._" are omitted from the service name; the service name is the same as the context name it is derived from. For example, assuming the context `ContextA` is published as an OData service, the OData service name used in the query is as follows:

- Name space specified ("acme.com") in the CDS document (no nested contexts)
  OData Service Name: `http::/<BaseURL>/java/odata/v4/<acme.com>._.<ContextA>/$metadata`
- Name space **not** specified in the CDS document (no nested contexts)
  OData Service Name: `http::/<BaseURL>/java/odata/v4/<ContextA>/$metadata`

If the CDS document includes a nested **subcontext** that is annotated for publication as an OData v4 service, you can access the exposed artifacts defined in the nested **subcontext** directly, by specifying the complete context path (separated by a ".") in the query URL, as illustrated in the following examples:

- Subcontext `ContextA1`
  `https://<BaseURL>/java/odata/v4/<acme.com>._.<ContextA>.<ContextA1>/$metadata`
- Subcontext `ContextA2`
  `https://<BaseURL>/java/odata/v4/<acme.com>._.<ContextA>.<ContextA2>/$metadata`

Related Information

Create a CDS Document (XS Advanced) [page 313]
Defining OData v2 Services for XS Advanced JavaScript Applications [page 524]
Tutorial: Create OData v4 Services for XS Advanced Java Applications [page 581]

5.3.4.2.8 CDS Comment Types

The Core Data Services (CDS) syntax enables you to insert comments into object definitions.

Example: Comment Formats in CDS Object Definitions

```
namespace com.acme.myapp1;

  /**
   *  multi-line comment,
   *  for doxygen-style,
   *  comments and annotations
   */
  type Type1 {
       element Fstr:       String( 5000 ); // end-of-line comment
               Flstr:      LargeString;
       /*inline comment*/  Fbin:       Binary( 4000 );
       element Flbin:      LargeBinary;
               Fint:       Integer;
       element Fint64:     Integer64;
               Ffixdec:    Decimal( 34, 34 /* another inline comment */);
       element Fdec:       DecimalFloat;
               Fflt:       BinaryFloat;
       //complete line comment element Flocdat:    LocalDate;    LocalDate
temporarily switched off
       //complete line comment        Floctim:    LocalTime;
       element Futcdatim:  UTCDateTime;
               Futctstmp:  UTCTimestamp;
  };
```

Overview

You can use the forward slash (`/`) and the asterisk (`*`) characters to add comments and general information to CDS object-definition files. The following types of comment are allowed:

- In-line comment
- End-of-line comment
- Complete-line comment
- Multi-line comment

In-line Comments

The in-line comment enables you to insert a comment into the middle of a line of code in a CDS document. To indicate the start of the in-line comment, insert a forward-slash (`/`) followed by an asterisk (`*`) before the comment text. To signal the end of the in-line comment, insert an asterisk followed by a forward-slash character (`*/`) after the comment text, as illustrated by the following example:.

```
element Flocdat:  /*comment text*/  LocalDate;
```

End-of-Line Comment

The end-of-line comment enables you to insert a comment at the end of a line of code in a CDS document. To indicate the start of the end-of-line comment, insert two forward slashes (`//`) before the comment text, as illustrated by the following example:.

```
element Flocdat:    LocalDate; // Comment text
```

Complete-Line Comment

The complete-line comment enables you to tell the parser to ignore the contents of an entire line of CDS code. The comment out a complete line, insert two forward slashes (`//`) at the start of the line, as illustrated in the following example:

```
//  element Flocdat:    LocalDate;      Additional comment text
```

Multi-Line Comments

The multi-line comment enables you to insert comment text that extends over multiple lines of a CDS document. To indicate the start of the multi-line comment, insert a forward-slash (`/`) followed by an asterisk (`*`) at the start of the group of lines you want to use for an extended comment (for example, `/*`). To signal the end of the multi-line comment, insert an asterisk followed by a forward-slash character (`*/`). Each line between the start and end of the multi-line comment must start with an asterisk (`*`), as illustrated in the following example:

```
/*
*  multiline,
*  doxygen-style
*  comments and annotations
*/
```

Related Information

Create a CDS Document

5.3.4.2.9 CDS Extensions Artifacts

Extend existing artifact definitions with properties stored in an additional, external artifact.

The CDS extension mechanism allows you to add properties to existing artifact definitions without modifying the original source files. In this way, you can split the definition of an artifact across multiple files each of which can have a different life cycle and code owner. For example, a customer can add a new element to an existing entity definition by the following statement:

> Sample Code
>
> CDS Artifact Extension Syntax

```
extend EntityE with {
  newElement: Integer;
}
```

In the example above, the code illustrated shows how to define a new **element** inside an existing entity (`EntityE`) artifact.

> Note
>
> The `extend` statement changes an existing artifact; it does not define any additional artifact.

It is essential to ensure that additional element definitions specified in custom extensions do not break the existing definitions of the base application. This is achieved by adapting the name-search rules and by additional checks for the `extend` statements. For the definition of these rules and checks, it is necessary to define the relationship between an `extend` statement and the artifact definitions, as well as the relationship between an `extend` statement and any additional `extend` statements.

Organization of Extensions

When you extend an SAP application, you typically add new elements to entities or views; these additional elements usually work together and can, themselves, require additional artifacts, for example, "types" used as element "types". To facilitate the process, we define an extension package (or package for short), which is a set of extend statements, normal artifact definitions (for example, "types" which are used in an `extend` declaration), and extension relationships (also known as "dependencies"). Each CDS source file belongs to exactly one package; all the definitions in this file contribute to that one (single) package. However, a "package" typically contains contributions from multiple CDS source files.

> → Tip
>
> It is also possible to use a package to define a clear structure for an application, even if no extensions are involved.

Package Hierarchies

The extension mechanism can be used by developers as well as SAP industry solutions, partners, and customers. A productive system is likely to have more than one package; some packages might be independent from each other; some packages might depend on other packages. With such a model, we get an acyclic directed graph, with the base application and the extension packages as nodes and the dependencies as edges. This induces a partial order on the packages with the base application as lowest package (for simplicity we also call the base application a package). There is not necessarily a single top package (here: the final customer extension).

It is essential to ensure that which package is semantically self-contained and self-explanatory; avoid defining "micro" packages which can be technically applied individually but have no independent business value.

> ! Restriction
>
> Circular dependencies between extension packages are not allowed.

Package Definition

It is necessary to specify which `extend` statements and normal artifact definitions belong to which package and, in addition, on which other packages a package depends. A package is considered to be a normal CDS artifact; it has a name, and a corresponding definition, and its use can be found in the CDS Catalog. An extension package is defined by a special CDS source file with the file suffix `.package.hdbcds`.

> i Note
>
> The full stop (.) **before** the extension-package file name is mandatory.

Related Information

Create a CDS Document (XS Advanced) [page 313]
Create a CDS Extension [page 423]
The CDS Extension Descriptor [page 429]
The CDS Extension Package Descriptor [page 436]

5.3.4.3 Create a CDS Entity in XS Advanced

Define a design-time **entity** (or table) using the Core Data Services (CDS) syntax.

Prerequisites

To complete this task successfully, note the following prerequisites:

- You must have access to an SAP HANA system.
- You must have access to the SAP Web IDE for SAP HANA

  > **i** Note
  >
  > The permissions defined in the XS advanced role collection `XS_AUTHORIZATION_USER` must be assigned to the user who wants to access the tools included in the SAP Web IDE for SAP HANA.

- You must have already created a development workspace and a multi-target application (MTA) project.
- You must already have created a database module for your MTA application project.
- You must already have set up an HDI container for the CDS artifacts

  > **i** Note
  >
  > A container setup file (`.hdiconfig`) is required to define which plug-ins to use to create the corresponding catalog objects from your design-time artifacts when the multi-target application (or just the database module) is deployed.

- To view run-time objects, you must have access to the SAP HANA XS advanced run-time tools that enable you to view the contents of the catalog.

  > **i** Note
  >
  > The permissions defined in the XS advanced role collection `XS_AUTHORIZATION_USER` must be assigned to the user who wants to access the SAP HANA run-time tools.

Context

In the SAP HANA database, as in other relational databases, a CDS entity is a table with a set of data elements that are organized using columns and rows. SAP HANA Extended Application Services for XS advanced (XS advanced) enables you to use the CDS syntax to create a database entity as a design-time file. Deploying the

database module that contains the CDS entity creates the corresponding table in the corresponding schema. To create a design-time CDS entity-definition file, perform the following steps:

Procedure

1. Start the SAP HANA Web IDE for SAP HANA.

   The SAP Web IDE for SAP HANA is available at the following URL:

   `https://<HANA_HOST>:53075/`

   > → Tip
   >
   > To display the URL for the SAP Web IDE for SAP HANA, open a command shell, log on to the XS advanced run time, and run the following command:
   >
   > `xs app webide --urls`

2. Open the application project to which you want to add your CDS entity.
3. Create the CDS entity-definition file.

   Browse to the folder in the database module in your application's project workspace, for example, `<MyApp1>/HDB/src` where you want to create the new CDS entity-definition file and perform the following steps:

   a. Right-click the folder where you want to save the CDS entity-definition file and choose *New* > *CDS Artifact* in the context-sensitive pop-up menu.
   b. Enter the name of the entity-definition file in the *File Name* box, for example, `MyEntity`.

      > → Tip
      >
      > If you use the available setup Wizards to create your design-time artifacts, the correct file extensions is added automatically. The file extension is used to determine which plug-in to use to create the corresponding run-time object during deployment. CDS artifacts have the file extension `.hdbcds`, for example, `MyEntity.hdbcds`.

   c. Choose *Finish* to save the new CDS entity-definition file in the database module of your local project workspace.
4. Define the structure of the CDS entity.

   If the new entity-definition file is not automatically displayed by the file-creation wizard, double-click the entity-definition file you created in the previous step, for example, `MyEntity.hdbcds`, and add the entity-definition code to the file:

   > ℹ Note
   >
   > The following code example is provided for illustration purposes only.

```
entity MyEntity {
    key Author    : String(100);
    key BookTitle : String(100);
        ISBN      : Integer not null;
        Publisher : String(100);
} technical configuration {
```

```
    column store;
    unique index MYINDEX1 on (ISBN) desc;
};
```

5. Save the CDS entity-definition file.

   Saving the definition persists the file in your local workspace; it does not create any objects in the database catalog.

6. Activate the changes in the catalog.

   To activate the new entity definition and generate a corresponding table in the catalog, use the *Build* feature.

   a. Right-click the new database module in your application project.

   b. In the context-sensitive pop-up menu, choose *Build*.

   > → Tip
   >
   > You can follow progress of the build in the console at the bottom of the CDS editor.

7. Check that the new entity has been successfully created in the catalog.

   > → Tip
   >
   > A selection of run-time tools is available in the *Database Explorer* perspective of SAP web IDE for SAP HANA at the following location:
   >
   > *Tools* > *Database Explorer*

   In XS advanced, your database run-time objects are located in the HDI container created for your multi-target application's database module; you need to locate and bind to this application-specific container to view its contents. The container name contains the name of the user logged into the SAP Web IDE for SAP HANA, the name of the database module containing the CDS design-time entities, and the string *-hdi-container*, for example:

   `<XS_UserName>`*-ctetig24[...]-*`<DB_Module>`*-hdi-container*

   To bind to the HDI container, in the SAP HANA run-time *Catalog* tool, right-click *Catalog* in the catalog list, and in the *Search HDI Containers* dialog, locate the container to which you want to bind, and choose *Bind*.

8. Delete a table (entity) created by SAP HDI (**optional**).

   If you need to delete a table created by HDI in the context of the SAP Web IDE, perform the following steps:

   a. Remove the design-time table definition from the database module of your application project in SAP Web IDE.

   b. Right-click the database module in your application project.

   c. In the context-sensitive pop-up menu, choose *Build*.

   > → Tip
   >
   > In the context of HDI, you can use the HDI container API's `DELETE` command to remove files from the HDI container's virtual file system and then use the `MAKE` command with the path to the file slated for removal specified in the `undeploy_paths` parameter. For more information about the HDI APIs, see **The HDI Container API** in *Related Information* below.

Related Information

Set up an SAP HDI Container for XS Advanced Applications [page 237]
Create a CDS Document (XS Advanced) [page 313]
The HDI Container API

5.3.4.3.1 CDS Entities in XS Advanced

In the SAP HANA database, as in other relational databases, a CDS entity is a table with a set of data elements that are organized using columns and rows.

A CDS entity has a specified number of columns, defined at the time of entity creation, but can have any number of rows. Database entities also typically have meta-data associated with them; the meta-data might include constraints on the entity or on the values within particular columns. SAP HANA Extended Application Services (SAP HANA XS) enables you to create a database entity as a design-time file. All design-time files, including your CDS entity definition, can be transported to other SAP HANA systems and, when deployed, used to generate the same catalog objects. You can define the entity using CDS-compliant DDL.

> **i** Note
>
> In XS classic, the delivery unit is the medium SAP HANA provides to enable you to assemble all your application-related repository artifacts together into an archive that can be easily exported to other systems. In XS advanced, you add your artifacts to application modules (for example, a database module); the modules are used to define and deploy a multi-target application (MTA).

The following code illustrates an example of a single design-time entity definition using CDS-compliant DDL. The name of the top-level artifact, in the following example, the entity `MyTable` must match the name of the CDS artifact. In the example below, the CDS document must be named `MyTable.hdbcds`. In XS advanced, an optional name space can be declared; it indicates the repository package in which the object the document defines is located.

> **→** Tip
>
> From SAP HANA 2.0 SPS 01, it is possible to define **multiple** top-level artifacts (for example, contexts, entities, etc.) in a single CDS document. For this reason, you can choose any name for the CDS source file; there is no longer any requirement for the name of the CDS source file to be the same as the name of the top-level artifact.

> **‹≡›** Code Syntax
>
> CDS Entity definition in XS advanced (MyTable.hdbcds)

```
namespace com.acme.myapp1;
entity MyTable {
    key Author    : String(100);
    key BookTitle : String(100);
        ISBN      : Integer not null;
        Publisher : String(100);
} technical configuration {
    column store;
    unique index MYINDEX1 on (ISBN) desc;
```

```
};
```

If you want to create a CDS-compliant database entity definition as a design-time file in SAP HANA XS advanced model, you must create the entity as a flat file and save the file containing the DDL entity dimensions with the suffix `.hdbcds`, for example, `MyTable.hdbcds`. The

> **i Note**
>
> On deployment of the CDS design-time artifact, the file suffix, for example, `.hdbcds`, is used to determine which runtime plug-in to call during the activation process. The plug-in reads the repository file selected for activation, in this case a CDS-compliant entity, parses the object descriptions in the file, and creates the appropriate runtime objects.

When a CDS document is deployed (or activated), the deployment process generates a corresponding catalog object for each of the artifacts defined in the document; the location in the catalog is determined by the type of object generated. For example, the corresponding database table for a CDS entity definition is generated in the following catalog location:

`<SID>` > *Catalog* > `<SCHEMA_NAME>` > *Tables*

Entity Element Definition

You can expand the definition of an entity element beyond the element's name and type by using element **modifiers**. For example, you can specify if an entity element is the primary key or **part** of the primary key. The following entity element modifiers are available:

- `key`
  Defines if the specified element is the **primary** key or **part** of the primary key for the specified entity.

  > **i Note**
  >
  > Structured elements can be part of the key, too. In this case, all table fields resulting from the flattening of this structured field are part of the primary key.

- `null`
  Defines if an entity element can (`null`) or cannot (`not null`) have the value NULL. If neither `null` nor `not null` is specified for the element, the default value `null` applies (except for the `key` element).
- `default <literal_value>`
  Defines the default value for an entity element in the event that no value is provided during an INSERT operation. The syntax for the literals is defined in the primitive data-type specification.
- `generated always as <expression>`
  Defines a value that is computed as specified in `<expression>`, for example, `a=b`.
  You can also use the SAP HANA SQL clause `generated always as identity` in CDS entity definitions to defined a field in the database table that is present in the persistence and has a value that is computed as specified in the expression.

```
entity MyEntity {
    key   MyKey  : Integer;
    key   MyKey2 : Integer null;      // illegal combination
    key   MyKey3 : Integer default 2;
          elem2  : String(20) default 'John Doe';
```

```
          elem3  : String(20) default 'John Doe' null;
          elem4  : String default 'Jane Doe' not null;
          elem5  : Integer
          elem6  : Integer
          elem7  : Integer generated always as elem5+elem6;
};
```

Spatial Data

CDS entities support the use of spatial data types such as `hana.ST_POINT` or `hana.ST_GEOMETRY` to store geo-spatial coordinates. Spatial data is data that describes the position, shape, and orientation of objects in a defined space; the data is represented as two-dimensional geometries in the form of points, line strings, and polygons.

Multi-Store Tables, Partitions, and Extended Storage

Provided that SAP HANA Dynamic Tiering is correct installed running and extended storage is configured and available, CDS allows you to partition column entities into extended storage.

The sequence of multiple storage declarations, for example, "`using default storage`" and "`using extended storage`", is maintained against the database. Where multiple "`partition others`" clauses are defined, CDS replaces them with one "`partition others`" that is added as the last partition specification.

Sample Code

Storage Partition in CDS

```
entity Person
  {
    key id : Integer;
    ...
  }
  technical configuration {
    partitition by range(id) (
        using default storage (partitions ...)
        using extended storage (partitions...)
    ...);
  };
```

Where multi-level partitioning is defined, the first partition must not contain any "`using [ default | extended ] storage`" specification, and you can use the multi-level partition schemes “Hash-Range” and “Range-Range”.

Restriction

Some CDS data types are not supported in selected elements of a multi-level storage scenario. For example, `sap.cds::LargeString` cannot be used in a multi-storage range expression. For more information, see *CDS Entity Syntax Options in XS Advanced* in *Related Information* below.

Related Information

Create a CDS Entity in XS Advanced [page 348]
Entity Element Modifiers [page 354]
CDS Entity Syntax Options in XS Advanced [page 359]

5.3.4.3.2 Entity Element Modifiers

Element **modifiers** enable you to expand the definition of an entity element beyond the element's name and type. For example, you can specify if an entity element is the primary key or **part** of the primary key.

Example

```
entity MyEntity {
    key   MyKey  : Integer;
          elem2  : String(20) default 'John Doe';
          elem3  : String(20) default 'John Doe' null;
          elem4  : String default 'Jane Doe' not null;
};
entity MyEntity1 {
    key id : Integer;
    a  : integer;
    b  : integer;
    c  : integer generated always as a+b;
};
entity MyEntity2 {
    autoId : Integer generated [always|by default] as identity ( start with 10
increment by 2 );
    name : String(100);
};
```

key

```
key   MyKey  : Integer;
key   MyKey2 : Integer null;                // illegal combination
key   MyKey3 : Integer default 2;
```

You can expand the definition of an entity element beyond the element's name and type by using element **modifiers**. For example, you can specify if an entity element is the primary key or **part** of the primary key. The following entity element modifiers are available:

- `key`
  Defines if the element is the **primary** key or **part** of the primary key for the specified entity. You **cannot** use the `key` modifier in the following cases:
  - In combination with a `null` modifier. The `key` element is `non null` by default because NULL cannot be used in the `key` element.

> **Note**
>
> Structured elements can be part of the key, too. In this case, all table fields resulting from the flattening of this structured field are part of the primary key.

null

```
elem3  : String(20) default 'John Doe' null;
elem4  : String default 'Jane Doe' not null;
```

`null` defines if the entity element can (`null`) or cannot (`not null`) have the value NULL. If neither `null` nor `not null` is specified for the element, the default value `null` applies (except for the `key` element), which means the element **can** have the value NULL. If you use the `null` modifier, note the following points:

> **Caution**
>
> The keywords `nullable` and `not nullable` are no longer valid; they have been replaced for SPS07 with the keywords `null` and `not null`, respectively. The keywords `null` and `not null` must appear at the end of the entity element definition, for example, `field2 : Integer null;`.

- The `not null` modifier can only be added if the following is true:
  - A `default` it also defined
  - no `null` data is already in the table
- Unless the table is empty, bear in mind that when adding a new `not null` element to an existing entity, you must declare a default value because there might already be existing rows that do not accept NULL as a value for the new element.
- `null` elements with default values are permitted
- You cannot combine the element `key` with the element modifier `null`.
- The elements used for a unique index must have the `not null` property.

```
entity WithNullAndNotNull
{
  key id :  Integer;
  field1 :  Integer;
  field2 :  Integer null;  // same as field1, null is default
  field3 :  Integer not null;
};
```

default

```
default <literal_value>
```

For each scalar element of an entity, a default value can be specified. The `default` element identifier defines the default value for the element in the event that no value is provided during an INSERT operation.

> **i** Note
>
> The syntax for the literals is defined in the primitive data-type specification.

```
entity WithDefaults
{
  key id :  Integer;
  field1 :  Integer       default -42;
  field2 :  Integer64     default 9223372036854775807;
  field3 :  Decimal(5, 3) default 12.345;
  field4 :  BinaryFloat   default 123.456e-1;
  field5 :  LocalDate     default date'2013-04-29';
  field6 :  LocalTime     default time'17:04:03';
  field7 :  UTCDateTime   default timestamp'2013-05-01 01:02:03';
  field8 :  UTCTimestamp  default timestamp'2013-05-01 01:02:03';
  field9 :  Binary(32)    default x'0102030405060708090a0b0c0d0e0[...]';
  field10 : String(10)    default 'foo';
};
```

generated always as <expression>

```
entity MyEntity1 {
    key id : Integer;
    a  : integer;
    b  : integer;
    c  : integer generated always as a+b;
};
```

The SAP HANA SQL clause `generated always as <expression>` is available for use in CDS entity definitions; it specifies the expression to use to generate the column value at run time. An element that is defined with `generated always as <expression>` corresponds to a field in the database table that is present in the persistence and has a value that is computed as specified in the expression, for example, "`a+b`".

> **!** Restriction
>
> For use in XS advanced only; it is not possible to use generated calculated elements in XS classic. Please also note that the `generated always as <expression>` clause is only for use with column-based tables.

"Generated" fields and "calculated" field differ in the following way. **Generated** fields are physically present in the database table; values are computed on `INSERT` and need not be computed on `SELECT`. **Calculated** fields are not actually stored in the database table; they are computed when the element is "selected". Since the value of the **generated** field is computed on `INSERT`, the expression used to generate the value must not contain any non-deterministic functions, for example: `current_timestamp`, `current_user`, `current_schema`, and so on.

generated [always | by default] as identity

```
entity MyEntity2 {
```

```
    autoId : Integer generated always as identity ( start with 10 increment by
2 );
    name : String(100);
};
```

The SAP HANA SQL clause `generated as identity` is available for use in CDS entity definitions; it enables you to specify an identity column. An element that is defined with `generated as identity` corresponds to a field in the database table that is present in the persistence and has a value that is computed as specified in the sequence options defined in the `identity` expression, for example, `( start with 10 increment by 2 )`.

In the example illustrated here, the name of the generated column is `autoID`, the first value in the column is "10"; the `identity` expression `( start with 10 increment by 2 )` ensures that subsequent values in the column are incremented by 2, for example: 12, 14, and so on.

> **!** Restriction
>
> For use in XS advanced only; it is not possible to define an element with `IDENTITY` in XS classic. Please also note that the `generated always as identity` clause is only for use with column-based tables.

You can use either `always` or `by default` in the clause `generated as identity`, as illustrated in the examples in this section. If `always` is specified, then values are **always** generated; if `by default` is specified, then values are generated **by default**.

```
entity MyEntity2 {
    autoId : Integer generated by default as identity ( start with 10 increment
by 2 );
    name : String(100);
};
```

> **!** Restriction
>
> CDS does not support the use of reset queries, for example, `RESET BY <subquery>`.

Column Migration Behavior

The following table shows the migration strategy that is used for modifications to any given column; the information shows which actions are performed and what strategy is used to preserve content. During the migration, a comparison is performed on the column **type**, the generation **kind**, and the expression, if available. From an end-user perspective, the result of a column modification is either a preserved or new value. The aim of any modification to an entity (table) is to cause as little loss as possible.

- Change to the column **type**
  In case of a column type change, the content is converted into the new type. HANA conversion rules apply.
- Change to the expression clause
  The expression is re-evaluated in the following way:
  - "early"
    As part of the column change
  - "late"
    As part of a query

- Change to a calculated column
  A calculated column is transformed into a plain column; the new column is initialized with NULL.

Technically, columns are either dropped and added or a completely new "shadow" table is created into which the existing content is copied. The shadow table will then replace the original table.

Before column/ After row	**Plain**	**As** `<expr>`	**generated always as** `<expr>`	**generated always as identity** `<expr>`	**generated by default as identity** `<expr>`
Plain	Migrate Keep content	Drop/add Evaluate on select	Drop/add Evaluate on add	Migrate Keep content	Migrate Keep content
generated by default as identity `<expr>`	Migrate Keep content	Drop/add Evaluate on select	Drop/add Evaluate on add	Migrate Keep content	Migrate Keep content
generated always as identity `<expr>`	Migrate Keep content	Drop/add Evaluate on select	Drop/add Evaluate on add	Migrate Keep content	Migrate Keep content
generated always as `<expr>`	Drop/add NULL	Drop/add Evaluate on select	Drop/add Evaluate on add	Drop/add Keep content	Migrate Keep content
as `<expr>`	Drop/add NULL	Drop/add Evaluate on select	Drop/add Evaluate on add	Drop/add Keep content	Migrate Keep content

Related Information

Create an Entity in CDS
Create an Entity in CDS
CDS Entity Syntax Options
SAP HANA SQL and System Views Reference (CREATE TABLE)

5.3.4.3.3 CDS Entity Syntax Options in XS Advanced

The CDS syntax specifies a number of options you can use to define an **entity** (table) in a design-time artifact.

Example

> **i** Note
>
> This example is not a working example; it is intended for illustration purposes only.

```
namespace Pack1."pack-age2";
context MyContext {
  entity MyEntity1  {
    key id : Integer;
    name   : String(80);
  };
  entity MyEntity2 {
    key id : Integer;
    x      : Integer;
    y      : Integer;
    a      : Integer;
    field7 : Decimal(20,10) = power(ln(x)*sin(y), a);
  } technical configuration {
        column store;
        unique index Index1 on (x, y) desc;
        index Index2 on (x, a) desc;
        index Index3 on (y asc, a desc);
        partition by <partition_clause>;
        group <grouping_clause>;
        unload priority 0;
        no auto merge;
  };
  entity MyPartitionEntity  {
    key id : Integer;
    name   : String(80);
  } technical configuration {
        partition by range(id) (
           using default storage (partitions...)
           using extended storage (partitions...) );
  };
  entity MyEntity {
    key id : Integer;
    a : Integer;
    b : Integer;
    c : Integer;
    s {
      m : Integer;
      n : Integer;
    };
    } technical configuration {
      row store;
      index MyIndex1 on (a, b) asc;
      unique index MyIndex2 on (c, s) desc;
      fulltext index MYFTI1 on (t)
      FUZZY SEARCH INDEX off;
      };
    temporary entity MyTempEntity {
        a : Integer;
        b : String(20);
    } technical configuration {
```

```
        column store;
      };
  };
};
context MySpatialContext {
    entity Address {
      key id        : Integer;
      street_number : Integer;
      street_name   : String(100);
      zip           : String(10);
      city          : String(100);
      loc           : hana.ST_POINT(4326);
    };
};
context MySeriesContext {
    entity MySeriesEntity {
      key setId : Integer;
      t : UTCTimestamp;
      value : Decimal(10,4);
      series (
        series key (setId)
        period for series (t)
        equidistant increment by interval 0.1 second
        equidistant piecewise //increment or piecewise; not both
      )
    };
};
```

> ! Restriction
>
> For series data, you can use either `equidistant` or `equidistant piecewise`, but not both at the same time. The example above is for illustration purposes only.

Overview

Entity definitions resemble the definition of structured types, but with the following additional features:

- Key definition [page 361]
- Calculated Fields [page 361]
- Technical Configuration [page 362]
  Includes: index, unique index, full-text index, table type (row/column), partitioning, grouping, unload priority, and auto-merge options.
- Spatial data [page 371] *
- Series Data [page 371] *

On deployment in the SAP HANA XS advanced, each entity definition in CDS is used to generate a database table.

> → Tip
>
> From SAP HANA 2.0 SPS 01, it is possible to define **multiple** top-level artifacts (for example, contexts, entities, etc.) in a single CDS document. For this reason, you can choose any name for the CDS source file; there is no longer any requirement for the name of the CDS source file to be the same as the name of the top-level artifact.

The name of the table generated for each entity definition is built according to the same rules as for table types, for example, `Pack1.Pack2::MyModel.MyContext.MyTable`. In addition, the CDS name is restricted by the limits imposed on the length of the database identifier for the name of the corresponding SAP HANA database artifact (for example, table, view, or type); this is currently limited to 126 characters (including delimiters).

> **i** Note
>
> As of HANA 1.0 SPS 12, the name-space definition is optional.

Key Definition

```
type MyStruc2
{
  field1 : Integer;
  field2 : String(20);
};
entity MyEntity2
{
  key id  : Integer;
  name    : String(80);
  key str : MyStruc2;
};
```

Usually an entity must have a key; you use the keyword `key` to mark the respective elements. The key elements become the primary key of the generated SAP HANA table and are automatically flagged as `not null`. Key elements are also used for managed associations. Structured elements can be part of the key, too. In this case, all table fields resulting from the flattening of this structured element are part of the primary key.

Calculated Fields

The definition of an entity can contain calculated fields, as illustrated in type "`z`" the following example:

> Sample Code

```
entity MyCalcField {
  a : Integer;
  b : Integer;
  c : Integer = a + b;
  s : String(10);
  t : String(10) = upper(s);
  x : Decimal(20,10);
  y : Decimal(20,10);
  z : Decimal(20,10) = power(ln(x)*sin(y), a);
};
```

The calculation expression can contain arbitrary expressions and SQL functions. The following restrictions apply to the expression you include in a calculated field:

- The definition of a calculated field must not contain other calculated fields, associations, aggregations, or subqueries.

- A calculated field cannot be key.
- No index can be defined on a calculated field.
- A calculated field cannot be used as foreign key for a managed association.

In a query, calculated fields can be used like ordinary elements.

> **i Note**
>
> In SAP HANA tables, you can define columns with the additional configuration "`GENERATED ALWAYS AS`". These columns are physically present in the table, and all the values are stored. Although these columns behave for the most part like ordinary columns, their value is computed upon insertion rather than specified in the `INSERT` statement. This is in contrast to calculated field, for which no values are actually stored; the values are computed upon `SELECT`.

technical configuration

The definition of an entity can contain a section called `technical configuration`, which you use to define the elements listed in the following table:

- Storage type (row/column) [page 362]
- Indexes [page 363]
- Full text indexes [page 363]
- Partitioning [page 364]
- Grouping [page 369]
- Unload priority [page 370]
- Auto merge option [page 370]

> **i Note**
>
> The syntax in the technical configuration section is as close as possible to the corresponding clauses in the SAP HANA SQL `Create Table` statement. Each clause in the technical configuration must end with a semicolon (";").

Storage Type and Table Types

In the technical configuration for an entity, you can use the `store` keyword to specify the storage type ("`row`" or "`column`") for the generated table, as illustrated in the following example. If no store type is specified, a "column" store table is generated by default.

> **Sample Code**
>
> Storage Type

```
entity MyEntity {
   key id : Integer;
   a : Integer;
   b : Integer;
   t : String(100);
   s {
       u : String(100);
   };
```

```
} technical configuration {
   row store;
};
```

To specify a table of type “global temporary”, use the `temporary entity` keywords in the CDS entity definition, as illustrated in the following example. To specify a “global temporary table” with the type **column**, use the `temporary entity` keywords and, in addition, define the storage type as `column`, as illustrated in the following examples.

Sample Code

Temporary Table Types

```
context MyContext1 {
    temporary entity MyEntity3 {
        ID : Integer;
        name : String(30);
    };
    temporary entity MyTempEntity {
        a : Integer;
        b : String(20);
    } technical configuration {
        column store;
    };
};
```

Indexes

In the technical configuration for an entity, you can use the `index` and `unique index` keywords to specify the index type for the generated table. For example, `unique` generates an index where no two sets of data in the indexed entity can have identical key values. You can use the keywords “`asc`” (ascending) or “`desc`” (descending) to specify the order of the index.

Sample Code

```
entity MyEntity {
   key id : Integer;
   a : Integer;
   b : Integer;
   t : String(100);
   s {
       u : String(100);
   };
} technical configuration {
   index MyIndex1 on (a, b) asc;
   unique index MyIndex2 on (c, s) desc;
   index MyIndex3 on (b asc, t desc);
};
```

Note

You specify the index order (ascending or descending) for individual rows or columns, for example, `(b asc, t desc, s.u desc)`.

Full text indexes

In the technical configuration for an entity, you can use the `fulltext index` keyword to specify the full-text index type for the generated table, as illustrated in the following example.

Sample Code

```
entity MyEntity {
    key id : Integer;
    a : Integer;
    b : Integer;
    t : String(100);
    s {
        u : String(100);
    };
} technical configuration {
    row store;
    index MyIndex1 on (a, b) asc;
    unique index MyIndex2 on (a, b) asc;
    fulltext index MYFTI1 on (t)
        LANGUAGE COLUMN t
        LANGUAGE DETECTION ('de', 'en')
        MIME TYPE COLUMN s.u
        FUZZY SEARCH INDEX off
        PHRASE INDEX RATIO 0.721
        SEARCH ONLY off
        FAST PREPROCESS off
        TEXT ANALYSIS off;
    fuzzy search index on (s.u);
};
```

The `fulltext index` is identical to the standard SAP HANA SQL syntax for `CREATE FULLTEXT INDEX`. A `FUZZY SEARCH INDEX` in the technical configuration section of an entity definition corresponds to the `@SearchIndex` annotation in XS classic and the statement "`FUZZY SEARCH INDEX ON`" for a table column in SAP HANA SQL. It is not possible to specify both a full-text index and a fuzzy search index for the same element.

Partitioning

In the `technical configuration` of an entity definition, you can specify the partitioning information for the generated table, as illustrated in the following example:

Sample Code

Partition Specification

```
entity MyEntity {
   key id : Integer;
   a : Integer;
   b : Integer;
   t : String(100);
   s {
      u : String(100);
   };
} technical configuration {
   <partition_clause>;
};
```

The code required in the `<partition_clause>` is identical to the corresponding clause in the standard `HANA SQL CREATE TABLE` statement, as illustrated in the following example:

Note

SAP HANA CDS only supports those SAP HANA partitioning features introduced up to and including SAP HANA 2.0 SPS 02.

Sample Code

Partition Specification

```
entity MyEntity {
   key id : Integer;
   a : Integer;
   b : Integer;
   t : String(100);
} technical configuration {
   partition by hash (id) partitions 2,
                range (a) (partition 1 <= values < 10,
                           partition values = 10,
                           partition others);
};
```

The `partition by` clause defines the rules to use to partition the corresponding table, for example, using hash, range, or round-robin partition rules.

Note

You can use the `partition by` clause to ensure any partitions added after activation of a CDS entity (for example, with an SQL statement `ALTER TABLE <MyEntity> PARTITION BY RANGE("id")`) are retained on reactivation of the original CDS entity.

Normally, any external changes to a table that was originally created by activating a CDS entity definition are lost on reactivation; the partitions defined after the entity activation are lost. With the `keeping existing layout` option, you can use the `partition by` clause to preserve an existing partitioning, if possible. It is not possible to preserve the added partitions on reactivation of the CDS entity, if the fields used in the external partition specification have changed in the entity definition in a way that no longer fits the partition.

Sample Code

Retain existing partitions on reactivation

```
entity MyEntity {
  ...
} technical configuration {
  partition by keeping existing layout;
};
```

CDS allows you to create a table fully in extended storage, as illustrated in the following example:

Sample Code

Create a table in extended storage

```
entity MyStorageEntity {
  ...
} technical configuration {
    using extended storage;
};
```

When using CDS to create a table in extended storage, all existing restrictions relating to SAP HANA extended storage tables apply and, in addition, it is not permitted to use any of the following features:

- Series Tables

- Temporary Tables (global and local)
- Elements that are "`generated by (default)`", "`generated as (identity)`"
- “Full Text” and “Fuzzy Search” indexes
- Partitioning
- The clause "`PARTITION BY KEEPING EXISTING LAYOUT`"
- Unload Priority
- Auto Merge

CDS supports the use of multi-store column-based tables in extended storage provided that SAP HANA Dynamic Tiering is installed and, in addition, the target (extended) storage is configured and available. This support for multi-store tables means that you can partition entities of storage type `"column"` into extended storage.

Sample Code

Column Tables using partitions in extended storage

```
entity MyStorageEntity {
  ...
} technical configuration {
    partitition by range(id)
       (using default storage (partitions ...)
        using extended storage (partitions...) ...);
};
```

When declaring multiple partitions, bear in mind that the sequence of "`using [ default | extended ] storage`" declarations is maintained against the database. Note, too that multiple "`partition others`" clauses are removed and replaced with a single "`partition others`" clause and added as the last partition specification.

For configurations with multi-level partitions, all existing restrictions relating to SAP HANA multi-storage tables apply and, in addition, it is not permitted to use any of the following features:

- Series Tables
- Temporary Tables (both global and local)
- Elements that are "`generated by (default)`", or "`generated as (identity)`"
- “Full Text” and “Fuzzy Search” indexes
- The clause "`USING EXTENDED STORAGE`" to create the table fully in extended storage

Range partitions can now contain the time-selection interval. However, if a time-selection interval is specified, the partition clause in the `technical configuration` section of the CDS entity definition must include and enable `WITH PARTITIONING ON ANY COLUMNS`, as illustrated in the following example:

Sample Code

Partitions with time-selection interval

```
entity MyStorageEntity {
  ...
} technical configuration {
    partition by range(column)
      (partition '00000000' is current, partition ...)
       with partitioning on any columns on;
};
```

In contrast to the SAP HANA SQL syntax, CDS does not required a time-selection interval to appear only once and at the very first position in the range specification. However, note that multiple time-selection ranges are combined into a single time range and inserted as the first range specification in the resulting SQL statement `CREATE COLUMN TABLE ... PARTITION BY ...` that is used to create the table in the database. In CDS, the following restrictions apply for time-selection partitions:

- The time-selection interval must be declared in default storage.
- The underlying type of the range expression must be a string with a minimum length of eight characters
- The value of the '`is current`' range must be an eight character sequence, for example, '00000000'
- All other interval values must be enclosed in single quotes ('`<value>`') and be eight characters long.
- Time selection cannot be combined with a YEAR/MONTH range expression
- In multi-level partitions, time selection must not appear in the first-level partition definition
- The '`WITH PARTITIONING ON ANY COLUMNS ON`' clause cannot be used in combination with '`KEEPING EXISTING LAYOUT`' .

The following table shows which CDS data types are supported in the various multi- or extended-storage contexts:

CDS Data Type Compatibility Matrix

CDS Data Type	Multi-Storage Range Expression	Multi-Storage Partition Payload	Extended Storage Table
sap.cds::String	Yes	Yes	Yes
sap.cds::LargeString	-	Yes	Yes
sap.cds::Binary	Yes	Yes	Yes
sap.cds::LargeBinary	-	Yes	Yes
sap.cds::Integer	Yes	Yes	Yes
sap.cds::Integer32	Yes	Yes	Yes
sap.cds::Integer64	Yes	Yes	Yes
sap.cds::Decimal	Yes	Yes	Yes
sap.cds::DecimalFloat	-	-	Yes
sap.cds::BinaryFloat	-	Yes	Yes
sap.cds::LocalDate	Yes	Yes	Yes
sap.cds::LocalTime	-	Yes	Yes
sap.cds::UTCDateTime	Yes	Yes	Yes
sap.cds::UTCTimestamp	-	-	Yes
sap.cds::Boolean	-	-	-

CDS Data Type	Multi-Storage Range Expression	Multi-Storage Partition Payload	Extended Storage Table
sap.cds::hana.ALPHANUM	-	-	-
sap.cds::hana.SMALLINT	Yes	Yes	Yes
sap.cds::hana.TINYINT	Yes	Yes	Yes
sap.cds::hana.SMALLDECIMAL	-	-	Yes
sap.cds::hana.REAL	-	Yes	Yes
sap.cds::hana.CHAR	Yes	Yes	Yes
sap.cds::hana.NCHAR	Yes	Yes	Yes
sap.cds::hana.VARCHAR	Yes	Yes	Yes
sap.cds::hana.CLOB	-	Yes	Yes
sap.cds::hana.SHORTTEXT	-	-	-
sap.cds::hana.TEXT	-	-	-
sap.cds::hana.BINTEXT	-	-	-
sap.cds::hana.BINARY	-	Yes	Yes
sap.cds::hana.ST_POINT	-	-	-
sap.cds::hana.ST_GEOMETRY	-	-	-

Migration Disabled

If the technical configuration of an entity contains the clause "`migration disabled`", the activation of the CDS source is only allowed if changes in the entity definition do not lead to a migration of the table. If a migration is required, the activation fails and the changes need to be made manually.

Sample Code

```
entity MyEntity {
  key id : Integer;
  name : String(100);
  value : Decimal(10,2);
} technical configuration {
  migration disabled;
};
```

If the entity defined in the previous example is changed in a way that allows CDS to adapt the corresponding table via `ALTER` statements, the activation of the modified table will succeed. This is typically the case for adding or removing elements (as long they are not "`key`" elements) or adding and removing indexes, as illustrated in the following example.

Sample Code

```
entity MyEntity {
  key id : Integer;
  name : String(120);
  name2 : String(80);
  someTime: LocalTime;
} technical configuration {
  migration disabled;
};
```

Changing an element **type** in the way illustrated in the following example is not allowed; activating the following CDS document fails because the change of type in elements "`name`" and"`name2`" mean that a migration is required, which is explicitly forbidden by the (`migration disabled` clause.

Sample Code

```
entity MyEntity {
  key id : Integer;
  name : String(120);
  name2 : String(80);
  someTime: LocalTime;
} technical configuration {
  migration disabled;
};
```

Grouping

In the `technical configuration` of an entity definition, you can specify the grouping information for the generated table, as illustrated in the following example:

Sample Code

Grouping Specification

```
entity MyEntity {
   key id : Integer;
   a : Integer;
   b : Integer;
   t : String(100);
   s {
       u : String(100);
   };
} technical configuration {
   <grouping_option>;
};
```

The code required in the `<grouping_option>` is identical to the corresponding clause in the standard `HANA SQL CREATE TABLE` statement, as illustrated in the following example:

Sample Code

Grouping Specification

```
entity MyEntity {
   key id : Integer;
   a : Integer;
   b : Integer;
   t : String(100);
```

```
} technical configuration {
   group type Foo group subtype Bar group name Wheeeeezz;
};
```

You must set the `group type`, the group `group subtype`, and the `group name`.

Unload Priority

In the `technical configuration` of an entity definition, you can specify the unload priority for the generated table, as illustrated in the following example:

Sample Code

Unload Priority Specification

```
entity MyEntity {
   key id : Integer;
   a : Integer;
   b : Integer;
   t : String(100);
   s {
       u : String(100);
   };
} technical configuration {
   unload priority <integer_literal>;
};
```

`unload priority` specifies the priority with which a table is unloaded from memory. The priority can be set between 0 (zero) and 9 (nine), where 0 means “cannot be unloaded” and 9 means “earliest unload”.

Auto-Merge Option

In the `technical configuration` of an entity definition, you can specify any automatic-merge options for the generated table, as illustrated in the following example:

Sample Code

Auto-Merge Option

```
entity MyEntity {
   key id : Integer;
   a : Integer;
   b : Integer;
   t : String(100);
   s {
       u : String(100);
   };
} technical configuration {
   [no] auto merge;
};
```

Note

`auto merge;` triggers an automatic delta merge; `no auto merge;` disables the automatic delta merge operation.

Spatial Types *

The following example shows how to use the spatial type `ST_POINT` in a CDS entity definition. In the example entity `Person`, each person has a home address and a business address, each of which is accessible via the corresponding associations. In the `Address` entity, the geo-spatial coordinates for each person are stored in element `loc` using the spatial type `ST_POINT` (*).

Sample Code

```
context SpatialData {
    entity Person {
        key id : Integer;
        name : String(100);
        homeAddress : Association[1] to Address;
        officeAddress : Association[1] to Address;
    };
    entity Address {
        key id : Integer;
        street_number : Integer;
        street_name : String(100);
        zip : String(10);
        city : String(100);
        loc : hana.ST_POINT(4326);
    };
    view CommuteDistance as select from Person {
        name,
        homeAddress.loc.ST_Distance(officeAddress.loc) as distance
    };
};
```

Series Data *

CDS enables you to create a table to store series data by defining an entity that includes a `series ()` clause as an table option and then defining the appropriate paremeters and options.

Note

The `period for series` must be unique and should not be affected by any shift in timestamps.

Sample Code

```
context SeriesData {
    entity MySeriesEntity1 {
        key setId : Integer;
        t : UTCTimestamp;
        value : Decimal(10,4);
        series (
            series key (setId)
            period for series (t)
            equidistant increment by interval 0.1 second
        );
    };
    entity MySeriesEntity2 {
        key setId : Integer;
        t : UTCTimestamp;
        value : Decimal(10,4);
```

```
        series (
            series key (setId)
            period for series (t)
            equidistant piecewise
        );
    };
};
```

CDS also supports the creation of a series table called `equidistant piecewise` using Formula-Encoded Timestamps (FET). This enables support for data that is not loaded in an order that ensures good compression. There is no a-priori restriction on the timestamps that are stored, but the data is expected to be well approximated as piecewise linear with some jitter. The timestamps do not have a single slope/offset throughout the table; rather, they can change within and among series in the table.

> ! Restriction
>
> The `equidistant piecewise` specification can only be used in CDS; it cannot be used to create a table with the SQL command `CREATE TABLE`.

When a series table is defined as `equidistant piecewise`, the following restrictions apply:

1. The `period` includes one column (instant); there is no support for interval periods.
2. There is no support for `missing elements`. These could logically be defined if the period includes an interval start and end. Missing elements then occur when we have adjacent rows where the end of the `interval` **does not** equal the start of the `interval`.
3. The type of the period column must map to the one of the following types: `DATE`, `SECONDDATE`, or `TIMESTAMP`.

> ⚠ Caution
>
> (*) For information about the capabilities available for your license and installation scenario, refer to the Feature Scope Description for SAP HANA.

5.3.4.4 Create a CDS User-Defined Structure in XS Advanced

Define a design-time custom **structured type** using the Core Data Services (CDS) syntax.

Prerequisites

To complete this task successfully, note the following prerequisites:

- You must have access to an SAP HANA system.
- You must have access to the SAP Web IDE for SAP HANA

> i Note
>
> The permissions defined in the XS advanced role collection `XS_AUTHORIZATION_USER` must be assigned to the user who wants to access the tools included in the SAP Web IDE for SAP HANA.

- You must have already a development workspace and a multi-target application (MTA) project.
- You must already have created a database module for your MTA application project.
- You must already have set up an HDI container for the CDS artifacts

  > **i Note**
  >
  > A container setup file (`.hdiconfig`) is required to define which plug-ins to use to create the corresponding catalog objects from the design-time artifacts when the multi-target application (or just the database module) is deployed.

- You must have access to the SAP HANA XS advanced run-time tools that enable you to view the contents of the catalog.

  > **i Note**
  >
  > The permissions defined in the XS advanced role collection `XS_AUTHORIZATION_USER` must be assigned to the user who wants to access the SAP HANA run-time tools.

Context

A structured type is a data type comprising a list of attributes, each of which has its own data type. SAP HANA Extended Application Services for XS advanced model (XS advanced) enables you to use the CDS syntax to create a user-defined structured type as a design-time file that is transportable. Deploying the CDS document containing the type definition creates the corresponding types in the database catalog. To create a CDS document that defines one or more structured types, perform the following steps:

Procedure

1. Start the SAP HANA Web IDE for SAP HANA.

   The SAP Web IDE for SAP HANA is available at the following URL:

   `https://<HANA_HOST>:53075/`

   > **→ Tip**
   >
   > To display the URL for the SAP Web IDE for SAP HANA, open a command shell, log on to the XS advanced run time, and run the following command:
   >
   > `xs app webide --urls`

2. Open the application project to which you want to add a CDS-compliant, user-defined structured type.
3. Create the CDS entity-definition file.

   Browse to the folder in the database module in your project workspace, for example, `<MyApp1>/HDB/src` where you want to create the new CDS type-definition file and perform the following steps:

   a. Right-click the folder where you want to save the CDS type-definition file and choose *New* > *CDS Artifact* in the context-sensitive pop-up menu.

b. Enter the name of the entity-definition file in the *File Name* box, for example, `MyStructuredType`.

> → Tip
>
> If you use the available setup Wizards to create your design-time artifacts, the correct file extensions is added automatically. The file extension is used to determine which plug-in to use to create the corresponding run-time object during deployment. CDS artifacts have the file extension `.hdbcds`, for example, `MyStructuredType.hdbcds`.

c. Choose *Finish* to save the new CDS type-definition file in the database module of your local project workspace.

4. Define the details of the new CDS structured type.

   If the new type-definition file is not automatically displayed by the file-creation wizard, double-click the type-definition file you created in the previous step, for example, `MyStructuredType.hdbcds`, and add the type-definition code to the file:

   > ℹ Note
   >
   > The following code example is provided for illustration purposes only.

   ```
   table type MyStructuredType
   {
       aNumber : Integer;
       someText : String(80);
       otherText : String(80);
   };
   ```

5. Save the CDS type-definition file.

   Saving the definition persists the file in your local workspace; it does not create any objects in the database catalog.

6. Activate the changes in the catalog.

   To activate the new entity definition and generate a corresponding table in the catalog, use the *Build* feature.

   a. Right-click the new database module in your application project.

   b. In the context-sensitive pop-up menu, choose *Build*.

   > → Tip
   >
   > You can follow progress of the build in the console at the bottom of the CDS editor.

7. Check that the new custom structured type has been successfully created in the catalog.

   > → Tip
   >
   > A selection of run-time tools is available in the *Database Explorer* perspective of SAP Web IDE for SAP HANA at the following location:
   >
   > *Tools* > *Database Explorer*

   In XS advanced, your database run-time objects are located in the HDI container created for your multi-target application's database module; you need to locate and bind to this application-specific container if you want to view its contents. The container name contains the name of the user logged into the SAP Web

IDE for SAP HANA, the name of the database module containing the CDS design-time entities, and the string *-hdi-container*, for example:

`<XS_UserName>-ctetig24[...]-<DB_Module>-hdi-container`

To bind to the HDI container, in the SAP HANA run-time *Catalog* tool, right-click *Catalog* in the catalog list, and in the *Search HDI Containers* dialog, locate the container to which you want to bind, and choose *Bind*.

Related Information

Set up an SAP HDI Container for XS Advanced Applications [page 237]
Create the Data Persistence Artifacts with CDS in XS Advanced [page 311]

5.3.4.4.1 CDS User-Defined Data Types in XS Advanced

User-defined data types reference existing structured types (for example, user-defined) or the individual types (for example, field, type, or context) used in another data-type definition.

You can use the *type* keyword to define a new data type in CDS-compliant DDL syntax. You can define the data type in the following ways:

- Using allowed structured types (for example, user-defined)
- Referencing another data type

In the following example, the element definition `field2 : MyType1;` specifies a new element `field2` that is based on the specification in the user-defined data type `MyType1`.

> **i** Note
>
> If you are using a CDS document to define a single CDS-compliant user-defined data type, the name of the CDS document must match the name of the top-level data type defined in the CDS document. In the following example, you must save the data-type definition “MyType1” in the CDS document `MyType1.hdbcds`.

```
namespace com.acme.myapp1;
type MyType1 {
      field1 : Integer;
      field2 : String(40);
      field3 : Decimal(22,11);
      field4 : Binary(11);
    };
```

In the following example, you must save the data-type definition “MyType2” in the CDS document `MyType2.hdbcds`; the document contains a using directive pointing to the data-type “MyType1” defined in CDS document `MyType1.hdbdd`.

```
namespace com.acme.myapp1;
using com.acme.myapp1::MyType1;
type MyType2 {
    field1 : String(50);
    field2 : MyType1;
```

```
};
```

In the following example, you must save the data-type definition “MyType3” in the CDS document `MyType3.hdbcds`; the document contains a using directive pointing to the data-type “MyType2” defined in CDS document `MyType2.hdbdd`.

```
namespace com.acme.myapp1;
using com.acme.myapp1::MyType2;
type MyType3 {
    field1 : UTCTimestamp;
    field2 : MyType2;
};
```

The following code example shows how to use the `type of` keyword to define an element using the definition specified in another user-defined data-type field. For example, `field4 : type of field3;` indicates that, like `field3`, `field4` is a `LocalDate` data type.

```
namespace com.acme.myapp1;
using com.acme.myapp1::MyType1;
entity MyEntity1 {
  key id  : Integer;
  field1  : MyType3;
  field2  : String(24);
  field3  : LocalDate;
  field4  : type of field3;
  field5  : type of MyType1.field2;
  field6  : type of InnerCtx.CtxType.b;  // context reference
};
```

You can use the `type of` keyword in the following ways:

- Define a new element (`field4`) using the definition specified in another user-defined element `field3`:
  `field4 : type of field3;`
- Define a new element `field5` using the definition specified in a **field** (`field2`) that belongs to another user-defined data type (`MyType1`):
  `field5 : type of MyType1.field2;`
- Define a new element (`field6`) using an existing field (`b`) that belongs to a data type (`CtxType`) in another **context** (`InnerCtx`):
  `field6 : type of InnerCtx.CtxType.b;`

The following code example shows you how to define nested contexts (`MyContext.InnerCtx`) and refer to data types defined by a user in the specified context.

```
namespace com.acme.myapp1;
context MyContext {
   // Nested contexts
   context InnerCtx {
      Entity MyEntity {
         …
      };
      Type CtxType {
        a : Integer;
        b : String(59);
      };
   };
   type MyType1 {
      field1 : Integer;
      field2 : String(40);
      field3 : Decimal(22,11);
      field4 : Binary(11);
```

```
   };
   type MyType2 {
      field1 : String(50);
      field2 : MyType1;
   };
   type MyType3 {
      field1 : UTCTimestamp;
      field2 : MyType2;
   };
   entity MyEntity1 {
      key id  : Integer;
      field1  : MyType3 not null;
      field2  : String(24);
      field3  : LocalDate;
      field4  : type of field3;
      field5  : type of MyType1.field2;
      field6  : type of InnerCtx.CtxType.b;  // refers to nested context
      field7  : InnerCtx.CtxType;           //  more context references
   } technical configuration {
        unique index IndexA on (field1) asc;
   };
};
```

Restrictions

CDS name resolution does not distinguish between CDS `elements` and CDS `types`. If you define a CDS element based on a CDS data type that has the same name as the new CDS element, CDS displays an error message and the deployment of the objects defined in the CDS document fails.

> ⚠ Caution
>
> In an CDS document, you cannot define a CDS element using a CDS type of the same name; you must specify the **context** where the target type is defined, for example, `MyContext.doobidoo`.

The following example defines an association between a CDS element and a CDS data `type` both of which are named `doobidoo`. The result is an error when resolving the names in the CDS document; CDS expects a type named `doobidoo` but finds an CDS entity element with the same name that is **not** a type.

```
context MyContext2 {
  type doobidoo : Integer;
  entity MyEntity {
    key id : Integer;
    doobidoo : doobidoo;  // error: type expected; doobidoo is not a type
  };
};
```

The following example works, since the explicit reference to the context where the type definition is located (`MyContext.doobidoo`) enables CDS to resolve the definition target.

```
context MyContext {
  type doobidoo : Integer;
  entity MyEntity {
    key id : Integer;
    doobidoo : MyContext.doobidoo;  // OK
  };
};
```

> **i Note**
>
> To prevent name clashes between artifacts that **are** types and those that **have** a type assigned to them, make sure you keep to strict naming conventions. For example, use an **uppercase** first letter for `MyEntity`, `MyView` and `MyType`; use a lowercase first letter for elements `myElement`.

Related Information

Create a CDS User-Defined Structure in XS Advanced [page 372]

5.3.4.4.2 CDS Structured Type Definition in XS Advanced

A structured type is a data type comprising a list of attributes, each of which has its own data type. The attributes of the structured type can be defined manually in the structured type itself and reused either by another structured type or an entity.

SAP HANA Extended Application Services advanced model (XS advanced) enables you to create a database structured type as a design-time file. All design-time files including your structured-type definition can be transported to other SAP HANA systems and deployed there to create the same catalog objects as those created in the original SAP HANA system. You can define the structured type using CDS-compliant DDL.

When a CDS document is deployed as part of a database module, the deployment process generates a corresponding catalog object for each of the artifacts defined in the CDS document; the location in the catalog is determined by the type of object generated. For example, the corresponding table type for a CDS type definition is generated in the following catalog location:

`<SID>` > *Catalog* > `<SCHEMA_NAME>` > *Procedures* > *Table Types*

Structured User-Defined Types

In a structured user-defined type, you can define original types (`aNumber` in the following example) or reference existing types defined elsewhere in the same type definition or another, separate type definition (`MyString80`). If you define multiple types in a single CDS document, for example, in a parent context, each structure-type definition must be separated by a semi-colon (;).

The type `MyString80` is defined in the following CDS document:

```
namespace Package1.Package2;
type MyString80: String(80);
```

A using directive is required to resolve the reference to the data type specified in `otherText : MyString80;`, as illustrated in the following example:

```
namespace Package1.Package2;
using Package1.Package2::MyString80;  //contains definition of MyString80
```

```
type MyStruct
{
  aNumber   : Integer;
  someText  : String(80);
  otherText : MyString80;  // defined in a separate type
};
```

> **Note**
>
> If you are using a CDS document to specify a single CDS-compliant data type, the name of the CDS document (`MyStruct.hdbcds`) must match the name of the top-level data type defined in the CDS document, for example, with the *type* keyword.

Nested Structured Types

Since user-defined types can make use of other user-defined types, you can build nested structured types, as illustrated in the following example:

```
namespace Package1.Package2;
using Package1.Package2::MyString80;
using Package1.Package2::MyStruct;
@Schema: 'MYSCHEMA'
context NestedStructs {
    type MyNestedStruct
    {
        name : MyString80;
        nested : MyStruct; // defined in a separate type
    };
    type MyDeepNestedStruct
    {
        text : LargeString;
        nested : MyNestedStruct;
    };
    type MyOtherInt    : type of MyStruct.aNumber; // => Integer
    type MyOtherStruct : type of MyDeepNestedStruct.nested.nested; // => MyStruct
};
```

The example above shows how you can use the `type of` keyword to define a type based on an existing type that is already defined in another user-defined structured type.

Generated Table Types

For each structured type, a SAP HANA table type is generated, whose name is built by concatenating the following elements of the CDS document containing the structured-type definition and separating the elements by a dot delimiter (.):

- the name space (for example, `Pack1.Pack2`)
- the names of all artifacts that enclose the type (for example, `MyModel`)
- the name of the type itself (for example, `MyNestedStruct`)

```
create type "Pack1.Pack2::MyModel.MyNestedStruct" as table(
  name             nvarchar(80),
```

```
  nested.aNumber   integer,
  nested.someText  nvarchar(80),
  nested.otherText nvarchar(80)
);
```

The columns of the table type are built by flattening the elements of the type. Elements with structured types are mapped to one column per nested element, with the column names built by concatenating the element names and separating the names by dots ".".

Table types are only generated for direct structure definitions; in the following example, this would include: `MyStruct`, `MyNestedStruct`, and `MyDeepNestedStruct`. No table types are generated for **derived** types that are based on structured types; in the following example, the derived types include: `MyS`, `MyOtherInt`, `MyOtherStruct`.

Example

```
namespace Pack1."pack-age2";
context MyModel
{
  type MyInteger  : Integer;
  type MyString80 : String(80);
  type MyDecimal  : Decimal(10,2);
  type MyStruct
  {
    aNumber   : Integer;
    someText  : String(80);
    otherText : MyString80;  // defined in example above
  };
  type MyS           : MyStruct;
  type MyOtherInt    : type of MyStruct.aNumber;
  type MyOtherStruct : type of MyDeepNestedStruct.nested.nested;
  type MyNestedStruct
  {
    name   : MyString80;
    nested : MyS;
  };
  type MyDeepNestedStruct
  {
    text   : LargeString;
    nested : MyNestedStruct;
  };
};
```

Related Information

Create a CDS User-Defined Structure in XS Advanced [page 372]
CDS User-Defined Data Types in XS Advanced [page 375]

5.3.4.4.3 CDS Structured Types in XS Advanced

A structured type is a data type comprising a list of attributes, each of which has its own data type. The attributes of the structured type can be defined manually in the structured type itself and reused either by another structured type or an entity.

Example

```
namespace examples;
context StructuredTypes {
    type MyOtherInt : type of MyStruct.aNumber; // => Integer
    type MyOtherStruct : type of MyDeepNestedStruct.nested.nested; // => MyStruct
    type EmptyStruct { };
    type MyStruct
    {
        aNumber : Integer;
        aText : String(80);
        anotherText : MyString80; // defined in a separate type
    };

    entity E {
        a : Integer;
        s : EmptyStruct;
    };

    type MyString80 : String(80);
    type MyS : MyStruct;
    type MyNestedStruct
    {
        name : MyString80;
        nested : MyS;
    };
    type MyDeepNestedStruct
    {
        text : LargeString;
        nested : MyNestedStruct;
    };
};
```

type

In a structured user-defined type, you can define original types (`aNumber` in the following example) or reference existing types defined elsewhere in the same type definition or another, separate type definition, for example, `MyString80` in the following code snippet. If you define multiple types in a single CDS document, each structure definition must be separated by a semi-colon (;).

```
type MyStruct
{
    aNumber     : Integer;
    aText       : String(80);
    anotherText : MyString80;  // defined in a separate type
};
```

You can define structured types that do not contain any elements, for example, using the keywords `type EmptyStruct { };`. In the example, below the generated table for entity “E” contains only one column: “a”.

> **!** Restriction
>
> It is not possible to generate an SAP HANA table type for an empty structured type.

```
type EmptyStruct { };
entity E {
    a : Integer;
    s : EmptyStruct;
};
```

type of

You can define a type based on an existing type that is already defined in another user-defined structured type, for example, by using the `type of` keyword, as illustrated in the following example:

```
Context StructuredTypes
{
    type MyOtherInt    : type of MyStruct.aNumber;                 // =>
Integer
    type MyOtherStruct : type of MyDeepNestedStruct.nested.nested;  // =>
MyStruct
};
```

Related Information

Create a CDS User-Defined Structure in XS Advanced [page 372]
CDS User-Defined Data Types in XS Advanced [page 375]
CDS Structured Type Definition in XS Advanced [page 378]

5.3.4.4.4 CDS Primitive Data Types

In the Data Definition Language (DDL), primitive (or core) data types are the basic building blocks that you use to define *entities* or *structure types* with DDL.

When you are specifying a design-time table (entity) or a view definition using the CDS syntax, you use data types such as *String*, *Binary*, or *Integer* to specify the type of content in the entity columns. CDS supports the use of the following primitive data types:

- DDL data types [page 383]
- Native SAP HANA data types [page 384]

The following table lists all currently supported simple DDL primitive data types. Additional information provided in this table includes the SQL syntax required as well as the equivalent SQL and EDM names for the listed types.

Supported SAP HANA DDL Primitive Types

Name	Description	SQL Literal Syntax	SQL Name	EDM Name
String (n)	Variable-length Unicode string with a specified maximum length of n=1-1333 characters (5000 for SAP HANA specific objects). Default = maximum length. String length (n) is mandatory.	'text with “quote”'	NVARCHAR	String
LargeString	Variable length string of up to 2 GB (no comparison)	'text with “quote”'	NCLOB	String
Binary(n)	Variable length byte string with user-defined length limit of up to 4000 bytes. Binary length (n) is mandatory.	x'01Cafe', X'01Cafe'	VARBINARY	Binary
LargeBinary	Variable length byte string of up to 2 GB (no comparison)	x'01Cafe', X'01Cafe'	BLOB	Binary
Integer	Respective container's standard signed integer. Signed 32 bit integers in 2's complement, -2**31 .. 2**31-1. Default=NULL	13, -1234567	INTEGER	Int64
Integer64	Signed 64-bit integer with a value range of -2^63 to 2^63-1. Default=NULL.	13, -1234567	BIGINT	Int64
Decimal(p, s)	Decimal number with fixed precision (p) in range of 1 to 34 and fixed scale (s) in range of 0 to p. Values for precision and scale are mandatory.	12.345, -9.876	DECIMAL(p, s)	Decimal
DecimalFloat	Decimal floating-point number (IEEE 754-2008) with 34 mantissa digits; range is roughly ±1e-6143 through ±9.99e+6144	12.345, -9.876	DECIMAL	Decimal
BinaryFloat	Binary floating-point number (IEEE 754), 8 bytes (roughly 16 decimal digits precision); range is roughly ±2.2207e-308 through ±1.7977e+308	1.2, -3.4, 5.6e+7	DOUBLE	Double
LocalDate	Local date with values ranging from 0001-01-01 through 9999-12-31	date'1234-12-31'	DATE	DateTimeOffset Combines date and time; with time zone must be converted to offset

Name	Description	SQL Literal Syntax	SQL Name	EDM Name
LocalTime	Time values (with seconds precision) and values ranging from 00:00:00 through 24:00:00	time'23:59:59', time'12:15'	TIME	Time For duration/ period of time (==xsd:duration). Use DateTimeOffset if there is a date, too.
UTCDateTime	UTC date and time (with seconds precision) and values ranging from 0001-01-01 00:00:00 through 9999-12-31 23:59:59	timestamp'2011-12-31 23:59:59'	SECONDDATE	DateTimeOffset Values ending with "Z" for UTC. Values before 1753-01-01T00:00:00 are not supported; transmitted as NULL.
UTCTimestamp	UTC date and time (with a precision of 0.1 microseconds) and values ranging from 0001-01-01 00:00:00 through 9999-12-31 23:59:59.9999999, and a special initial value	timestamp'2011-12-31 23:59:59.7654321'	TIMESTAMP	DateTimeOffset With Precision = "7"
Boolean	Represents the concept of binary-valued logic	true, false, unknown (null)	BOOLEAN	Boolean

The following table lists all the **native** SAP HANA primitive data types that CDS supports. The information provided in this table also includes the SQL syntax required (where appropriate) as well as the equivalent SQL and EDM names for the listed types.

> **i** Note
>
> * In CDS, the name of SAP HANA data types are prefixed with the word "hana", for example, `hana.ALPHANUM`, or `hana.SMALLINT`, or `hana.TINYINT`.

Supported Native SAP HANA Data Types

Data Type *	Description	SQL Literal Syntax	SQL Name	EDM Name
ALPHANUM	Variable-length character string with special comparison	-	ALPHANUMERIC	-
SMALLINT	Signed 16-bit integer	-32768, 32767	SMALLINT	Int16
TINYINT	Unsigned 8-bit integer	0, 255	TINYINT	Byte
REAL	32-bit binary floating-point number	-	REAL	Single

Data Type *	Description	SQL Literal Syntax	SQL Name	EDM Name
SMALLDECIMAL	64-bit decimal floating-point number	-	SMALLDECIMAL	Decimal
VARCHAR	Variable-length ASCII character string with user-definable length limit n	-	VARCHAR	String
CLOB	Large variable-length ASCII character string, no comparison	-	CLOB	String
BINARY	Byte string of fixed length n	-	BINARY	Blob
ST_POINT	0-dimensional geometry representing a single location	-	-	-
ST_GEOMETRY	Maximal supertype of the geometry type hierarchy; includes `ST_POINT`	-	-	-

The following example shows the **native** SAP HANA data types that CDS supports; the code example also illustrates the mandatory syntax.

> **i** Note
>
> Support for the geo-spatial types `ST_POINT` and `ST_GEOMETRY` is limited: these types can only be used for the definition of elements in types and entities. It is not possible to define a CDS view that selects an element based on a geo-spatial type from a CDS entity.

```
@nokey
entity SomeTypes {
  a : hana.ALPHANUM(10);
  b : hana.SMALLINT;
  c : hana.TINYINT;
  d : hana.SMALLDECIMAL;
  e : hana.REAL;
  h : hana.VARCHAR(10);
  i : hana.CLOB;
  j : hana.BINARY(10);
  k : hana.ST_POINT;
  l : hana.ST_GEOMETRY;
};
```

Related Information

Create a User-Defined Structures Type in CDS
Create a CDS User-Defined Structure in XS Advanced [page 372]

5.3.4.5 Create a CDS Association in XS Advanced

You create associations in a CDS entity definition, which is a design-time file in SAP HANA.

Prerequisites

To complete this task successfully, note the following prerequisites:

- You must have access to an SAP HANA system.
- You must have access to the SAP Web IDE for SAP HANA

  > **i** Note
  >
  > The permissions defined in the XS advanced role collection `XS_AUTHORIZATION_USER` must be assigned to the user who wants to access the tools included in the SAP Web IDE for SAP HANA.

- You must have already a development workspace and a multi-target application (MTA) project.
- You must already have created a database module for your MTA project.
- You must already have set up an HDI container for the CDS artifacts

  > **i** Note
  >
  > A container setup file (`.hdiconfig`) is required to define which plug-ins to use to create the corresponding catalog objects from the design-time artifacts when the multi-target application (or just the database module) is deployed.

Context

Associations define relationships between entities (tables). SAP HANA Extended Application Services for XS advanced (XS advanced) enables you to use the CDS syntax to create associations between entities. The association is defined in the entity definiton itself. To create an association between CDS entities, perform the following steps:

Procedure

1. Start the SAP HANA Web IDE for SAP HANA.

   The SAP Web IDE for SAP HANA is available at the following URL:

   `https://<HANA_HOST>:53075/`

   > → Tip
   >
   > To display the URL for the SAP Web IDE for SAP HANA, open a command shell, log on to the XS advanced run time, and run the following command:

```
xs app webide --urls
```

2. Open the application project to which you want to add your CDS association.
3. Create the CDS entity-definition file that will contain the associations you define.

   Browse to the folder in the database module in your application's project workspace where you want to create the new CDS entity-definition file, for example, `<MyApp1>/HDB/src`, and perform the following steps:

   a. Right-click the folder where you want to save the CDS entity-definition file and choose *New* > *CDS Artifact* in the context-sensitive pop-up menu.
   b. Enter the name of the entity-definition file in the *File Name* box, for example, `MyEntity`.

      > → Tip
      >
      > If you use the available setup Wizards to create your design-time artifacts, the correct file extensions is added automatically. The file extension is used to determine which plug-in to use to create the corresponding run-time object during deployment. CDS artifacts have the file extension `.hdbcds`, for example, `MyEntity.hdbcds`.

   c. Choose *Finish* to save the new CDS entity-definition file in the database module of your local project workspace.
4. Define the underlying CDS entities and structured types.

   If the new entity-definition file is not automatically displayed by the file-creation wizard, double-click the entity-definition file you created in the previous step, for example, `MyEntity.hdbcds`, and add the entity-definition code to the file:

   > ℹ Note
   >
   > The following code example is provided for illustration purposes only.

```
econtext MyEntity1 {
    type StreetAddress {
        name : String(80);
        number : Integer;
    };
    type CountryAddress {
        name : String(80);
        code : String(3);
    };
    entity Address {
        key id : Integer;
        street : StreetAddress;
        zipCode : Integer;
        city : String(80);
        country : CountryAddress;
        type : String(10); // home, office
    };
    entity Person
    {
        key id : Integer;
        addressId : Integer;
    };

};
```

5. Define a one-to-**one** association between CDS entities.

In the same entity-definition file you edited in the previous step, for example, `MyEntity.hdbcds`, add the code for the one-to-one association between the entity `Person` and the entity `Address`, as illustrated below:

> **i Note**
>
> This example does not specify cardinality or foreign keys, so the cardinality is set to the default 0..1, and the target entity's primary key (the element id) is used as foreign key.

```
entity Person
  {
   key id : Integer;
   address1 : Association to Address;
   addressId : Integer;
};
```

6. Define an unmanaged association with cardinality one-to-**many** between CDS entities.

   In the same entity-definition file you edited in the previous step, for example, `MyEntity.hdbcds`, add the code for the one-to-many association between the entity `Address` and the entity `Person`. The code should look something like the bold text illustrated in the following example:

```
entity Address {
   key id : Integer;
   street : StreetAddress;
   zipCode : Integer;
   city : String(80);
   country : CountryAddress;
   type : String(10); // home, office
   inhabitants : Association[*] to Person on inhabitants.addressId = id;
};
```

7. Save the CDS document.

   Saving the definition persists the file in your local workspace; it does not create any objects in the database catalog.

8. Activate the changes in the catalog.

   To activate the new entity definition and generate a corresponding table in the catalog, use the *Build* feature.

   a. Right-click the new database module in your application project.

   b. In the context-sensitive pop-up menu, choose *Build*.

   > **→ Tip**
   >
   > You can follow progress of the build in the console at the bottom of the CDS editor.

Related Information

Set up an SAP HDI Container for XS Advanced Applications [page 237]
Create a CDS Document (XS Advanced) [page 313]

5.3.4.5.1 CDS Associations

Associations define relationships between entities.

Associations are specified by adding an element to a source entity with an association **type** that points to a target entity, complemented by optional information defining cardinality and which keys to use.

> **i** Note
>
> CDS supports both managed and unmanaged associations.

SAP HANA Extended Application Services (SAP HANA XS) enables you to use associations in CDS entities or CDS views. The syntax for **simple** associations in a CDS document is illustrated in the following example:

```
namespace samples;
@Schema: 'MYSCHEMA'         // XS classic *only*
context SimpleAssociations {
    type StreetAddress {
        name : String(80);
        number : Integer;
    };
    type CountryAddress {
        name : String(80);
        code : String(3);
    };
    entity Address {
        key id : Integer;
        street : StreetAddress;
        zipCode : Integer;
        city : String(80);
        country : CountryAddress;
        type : String(10); // home, office
    };
    entity Person
    {
        key id : Integer;
        // address1,2,3 are to-one associations
        address1 : Association to Address;
        address2 : Association to Address { id };
        address3 : Association[1] to Address { zipCode, street, country };
        // address4,5,6 are to-many associations
        address4 : Association[0..*] to Address { zipCode };
        address5 : Association[*] to Address { street.name };
        address6 : Association[*] to Address { street.name AS streetName,
        country.name AS countryName };
    };
};
```

Cardinality in Associations

When using an association to define a relationship between entities in a CDS document, you use the **cardinality** to specify the type of relation, for example, one-to-one (to-one) or one-to-many (to-n); the relationship is with respect to both the source and the target of the association.

The target cardinality is stated in the form of `[ min .. max ]`, where `max=*` denotes infinity. If no cardinality is specified, the default cardinality setting `[ 0..1 ]` is assumed. It is possible to specify the maximum cardinality of the source of the association in the form `[ maxs, min .. max]`, too, where `maxs = *` denotes infinity.

→ Tip

The information concerning the maximum cardinality is only used as a hint for optimizing the execution of the resulting JOIN.

The following examples illustrate how to express cardinality in an association definition:

```
namespace samples;
@Schema: 'MYSCHEMA'              // XS classic *only*
context AssociationCardinality {
    entity Associations {
        // To-one associations
        assoc1 : Association[0..1]    to target; // has no or one target instance
        assoc2 : Association          to target; // as assoc1, uses the default
[0..1]
        assoc3 : Association[1]       to target; // as assoc1; the default for
min is 0
        assoc4 : Association[1..1]    to target; // association has one target
instance
        // To-many associations
        assoc5 : Association[0..*]    to target{id1};
        assoc6 : Association[]        to target{id1}; // as assoc5, [] is short
for [0..*]
        assoc7 : Association[2..7]    to target{id1}; // any numbers are
possible; user provides
        assoc8 : Association[1, 0..*] to target{id1}; // additional info. about
source cardinality
    };
    // Required to make the example above work
    entity target {
        key id1 : Integer;
        key id2 : Integer;
    };
};
```

Target Entities in Associations

You use the `to` keyword in a CDS view definition to specify the target entity in an association, for example, the name of an entity defined in a CDS document. A qualified entity name is expected that refers to an existing entity. A target entity specification is mandatory; a default value is **not** assumed if no target entity is specified in an association relationship.

The entity `Address` specified as the target entity of an association could be expressed in any of the ways illustrated the following examples:

```
address1 : Association to Address;
address2 : Association to Address { id };
address3 : Association[1] to Address { zipCode, street, country };
```

Filter Conditions and Prefix Notation

When following an association (for example, in a view), it is now possible to apply a filter condition; the filter is merged into the `ON`-condition of the resulting `JOIN`. The following example shows how to get a list of

customers and then filter the list according to the sales orders that are currently “open” for each customer. In the example, the infix filter is inserted after the association orders to get only those orders that satisfy the condition `[status='open']`.

Sample Code

```
view C1 as select from Customer {
  name,
  orders[status='open'].id as orderId
};
```

The association `orders` is defined in the entity definition illustrated in the following code example:

Sample Code

```
entity Customer {
   key id : Integer;
   orders : Association[*] to SalesOrder on orders.cust_id = id;
   name : String(80);
};
entity SalesOrder {
   key id : Integer;
   cust_id : Integer;
   customer: Association[1] to Customer on customer.id = cust_id;
   items : Association[*] to Item on items.order_id = id;
   status: String(20);
   date : LocalDate;
};
entity Item {
   key id : Integer;
   order_id : Integer;
   salesOrder : Association[1] to SalesOrder on salesOrder.id = order_id;
   descr : String(100);
   price : Decimal(8,2);
};
```

→ Tip

For more information about filter conditions and prefixes in CDS views, see *CDS Views* and *CDS View Syntax Options*.

Foreign Keys in Associations

For **managed** associations, the relationship between source and target entity is defined by specifying a set of elements of the target entity that are used as a foreign key. If no foreign keys are specified explicitly, the elements of the target entity’s designated primary key are used. Elements of the target entity that reside inside substructures can be addressed via the respective path. If the chosen elements do not form a unique key of the target entity, the association has cardinality to-many. The following examples show how to express foreign keys in an association.

```
namespace samples;
using samples::SimpleAssociations.StreetAddress;
using samples::SimpleAssociations.CountryAddress;
using samples::SimpleAssociations.Address;
@Schema: 'MYSCHEMA'           // XS classic *only*
```

```
context ForeignKeys {
    entity Person
    {
        key id : Integer;
        // address1,2,3 are to-one associations
        address1 : Association to Address;
        address2 : Association to Address { id };
        address3 : Association[1] to Address { zipCode, street, country };
        // address4,5,6 are to-many associations
        address4 : Association[0..*] to Address { zipCode };
        address5 : Association[*] to Address { street.name };
        address6 : Association[*] to Address { street.name AS streetName,
        country.name AS countryName };
    };
    entity Header {
        key id : Integer;
        toItems : Association[*] to Item on toItems.head.id = id;
    };
    entity Item {
        key id : Integer;
        head : Association[1] to Header { id };
        // <...>
    };
};
```

- `address1`
  No foreign keys are specified: the target entity's primary key (the element `id`) is used as foreign key.
- `address2`
  Explicitly specifies the foreign key (the element `id`); this definition is similar to `address1`.
- `address3`
  The foreign key elements to be used for the association are explicitly specified, namely: `zipcode` and the structured elements `street` and `country`.
- `address4`
  Uses only `zipcode` as the foreign key. Since `zipcode` is not a unique key for entity `Address`, this association has cardinality “to-many”.
- `address5`
  Uses the subelement `name` of the structured element `street` as a foreign key. This is not a unique key and, as a result, `address4` has cardinality “to-many”.
- `address6`
  Uses the subelement `name` of both the structured elements `street` and `country` as foreign key fields. The names of the foreign key fields must be unique, so an alias is required here. The foreign key is not unique, so `address6` is a “to-many” association.

You can use foreign keys of managed associations in the definition of other associations. In the following example, the appearance of association `head` in the ON condition is allowed; the compiler recognizes that the field `head.id` is actually part of the entity `Item` and, as a result, can be obtained without following the association `head`.

Sample Code

```
entity Header {
  key id : Integer;
  toItems : Association[*] to Item on toItems.head.id = id;
};

entity Item {
  key id : Integer;
  head : Association[1] to Header { id };
  ...
```

```
};
```

Restrictions

CDS name resolution does not distinguish between CDS `associations` and CDS `entities`. If you define a CDS association with a CDS entity that has the same name as the new CDS association, CDS displays an error message and the activation of the CDS document fails.

> ⚠ Caution
>
> In an CDS document, to define an association with a CDS entity of the same name, you must specify the **context** where the target entity is defined, for example, `Mycontext.Address3`.

The following code shows some examples of associations with a CDS entity that has the same (or a similar) name. Case sensitivity ("a", "A") is important; in CDS documents, `address` is not the same as `Address`. In the case of `Address2`, where the association name and the entity name are identical, the result is an error; when resolving the element names, CDS expects an entity named `Address2` but finds a CDS association with the same name instead. `MyContext.Address3` is allowed, since the target entity can be resolved due to the absolute path to its location in the CDS document.

```
context MyContext {
  entity Address  {…}
  entity Address1 {…}
  entity Address2 {…}
  entity Address3 {…}
  entity Person
  {
    key id   : Integer;
    address  : Association to Address;        // OK: "address" ≠ "Address”
    address1 : Association to Address1;       // OK: "address1" ≠ "Address1”
    Address2 : Association to Address2;       // Error: association name =
entity name
    Address3 : Association to MyContext.Address3;  //OK: full path to Address3

  };
};
```

Example: Complex (One-to-Many) Association

The following example shows a more complex association (to-many) between the entity “`Header`” and the entity “`Item`”.

```
namespace samples;
@Schema: 'MYSCHEMA'        // XS classic *only*
context ComplexAssociation {
    Entity Header {
        key PurchaseOrderId: BusinessKey;
        Items: Association [0..*] to Item on
Items.PurchaseOrderId=PurchaseOrderId;
        "History": HistoryT;
        NoteId: BusinessKey null;
        PartnerId: BusinessKey;
```

```
        Currency: CurrencyT;
        GrossAmount: AmountT;
        NetAmount: AmountT;
        TaxAmount: AmountT;
        LifecycleStatus: StatusT;
        ApprovalStatus: StatusT;
        ConfirmStatus: StatusT;
        OrderingStatus: StatusT;
        InvoicingStatus: StatusT;
    } technical configuration {
        column store;
    };
    Entity Item {
        key PurchaseOrderId: BusinessKey;
        key PurchaseOrderItem: BusinessKey;
        ToHeader: Association [1] to Header on
ToHeader.PurchaseOrderId=PurchaseOrderId;
        ProductId: BusinessKey;
        NoteId: BusinessKey null;
        Currency: CurrencyT;
        GrossAmount: AmountT;
        NetAmount: AmountT;
        TaxAmount: AmountT;
        Quantity: QuantityT;
        QuantityUnit: UnitT;
        DeliveryDate: SDate;
    } technical configuration {
        column store;
    };
    define view POView as SELECT from Header {
        Items.PurchaseOrderId as poId,
        Items.PurchaseOrderItem as poItem,
        PartnerId,
        Items.ProductId
    };
    // Missing types from the example above
    type BusinessKey: String(50);
    type HistoryT: LargeString;
    type CurrencyT: String(3);
    type AmountT: Decimal(15, 2);
    type StatusT: String(1);
    type QuantityT: Integer;
    type UnitT: String(5);
    type SDate: LocalDate;
};
```

Related Information

Create an Association in CDS

Create a CDS Association in XS Advanced [page 386]

5.3.4.5.2 CDS Association Syntax Options

Associations define relationships between entities.

Example: Managed Associations

```
Association [ <cardinality> ]  to <targetEntity> [ <forwardLink> ]
```

Example: Unmanaged Associations

```
Association [ <cardinality> ]  to <targetEntity> <unmanagedJoin>
```

Overview

Associations are specified by adding an element to a source entity with an association **type** that points to a target entity, complemented by optional information defining cardinality and which keys to use.

> **i** Note
>
> CDS supports both managed and unmanaged associations.

SAP HANA Extended Application Services (SAP HANA XS) enables you to use associations in the definition of a CDS entity or a CDS view. When defining an association, bear in mind the following points:

- `<Cardinality>` [page 395]
  The relationship between the source and target in the association, for example, one-to-one, one-to-many, many-to-one
- `<targetEntity>` [page 397]
  The target entity for the association
- `<forwardLink>` [page 398]
  The foreign keys to use in a managed association, for example, element names in the target entity
- `<unmanagedJoin>` [page 399]
  Unmanaged associations only; the `ON` condition specifies the elements of the source and target elements and entities to use in the association

Association Cardinality

When using an association to define a relationship between entities in a CDS view; you use the **cardinality** to specify the type of relation, for example:

- one-to-one (to-one)
- one-to-many (to-n)

The relationship is with respect to both the source and the target of the association. The following code example illustrates the syntax required to define the cardinality of an association in a CDS view:

```
[  [ ( maxs | * ) , ]                // source cardinality
   [ min .. ] ( max | * )            // target cardinality
]
```

In the most simple form, only the target cardinality is stated using the syntax `[ min .. max ]`, where `max=*` denotes infinity. Note that `[]` is short for `[ 0..* ]`. If no cardinality is specified, the default cardinality setting `[ 0..1 ]` is assumed. It is possible to specify the maximum cardinality of the source of the association in the form `[ maxs, min .. max]`, where `maxs = *` denotes infinity.

The following examples illustrate how to express cardinality in an association definition:

```
namespace samples;
@Schema: 'MYSCHEMA'              // XS classic *only*
context AssociationCardinality {
    entity Associations {
        // To-one associations
        assoc1 : Association[0..1]    to target;
        assoc2 : Association          to target;
        assoc3 : Association[1]       to target;
        assoc4 : Association[1..1]    to target; // association has one target
instance
        // To-many associations
        assoc5 : Association[0..*]    to target{id1};
        assoc6 : Association[]        to target{id1}; // as assoc4, [] is short
for [0..*]
        assoc7 : Association[2..7]    to target{id1}; // any numbers are
possible; user provides
        assoc8 : Association[1, 0..*] to target{id1}; // additional info. about
source cardinality
    };
    // Required to make the example above work
    entity target {
        key id1 : Integer;
        key id2 : Integer;
    };
};
```

The following table describes the various cardinality expressions illustrated in the example above:

Association Cardinality Syntax Examples

Association	Cardinality	Explanation
`assoc1`	[0..1]	The association has no or one target instance
`assoc2`		Like `assoc1`, this association has no or one target instance and uses the default [0..1]
`assoc3`	[1]	Like `assoc1`, this association has no or one target instance; the default for min is 0
`assoc4`	[1..1]	The association has one target instance
`assoc5`	[0..*]	The association has no, one, or multiple target instances
`assoc6`	[]	Like `assoc4`, [] is short for [0..*] (the association has no, one, or multiple target instances)

Association	Cardinality	Explanation
`assoc7`	[2..7]	Any numbers are possible; the user provides
`assoc8`	[1, 0..*]	The association has no, one, or multiple target instances and includes additional information about the source cardinality

When an infix filter effectively reduces the cardinality of a “to-N” association to “to-1”, this can be expressed explicitly in the filter, for example:

```
assoc[1: <cond> ]
```

Specifying the cardinality in the filter in this way enables you to use the association in the `WHERE` clause, where “to-N” associations are not normally allowed.

Sample Code

```
namespace samples;
@Schema: 'MYSCHEMA'             // XS classic *only*
context CardinalityByInfixFilter {
    entity Person {
        key id : Integer;
        name : String(100);
        address : Association[*] to Address on address.personId = id;
    };
    entity Address {
        key id : Integer;
        personId : Integer;
        type : String(20); // home, business, vacation, ...
        street : String(100);
        city : String(100);
    };
    view V as select from Person {
        name
    } where address[1: type='home'].city = 'Accra';
};
```

Association Target

You use the `to` keyword in a CDS view definition to specify the target entity in an association, for example, the name of an entity defined in a CDS document. A qualified entity name is expected that refers to an existing entity. A target entity specification is mandatory; a default value is **not** assumed if no target entity is specified in an association relationship.

```
Association[ <cardinality> ]  to <targetEntity> [ <forwardLink> ]
```

The target entity `Address` specified as the target entity of an association could be expressed as illustrated the following examples:

```
address1 : Association to Address;
address2 : Association to Address { id };
address3 : Association[1] to Address { zipCode, street, country };
```

Association Keys

In the relational model, associations are mapped to foreign-key relationships. For **managed** associations, the relation between source and target entity is defined by specifying a set of elements of the target entity that are used as a foreign key, as expressed in the `forwardLink` element of the following code example:

```
Association[ <cardinality> ]  to <targetEntity> [ <forwardLink> ]
```

The `forwardLink` element of the association could be expressed as follows:

```
<forwardLink>  =  { <foreignKeys> }
<foreignKeys>  = <targetKeyElement> [ AS <alias> ] [ , <foreignKeys> ]
<targetKeyElement> = <elementName> ( . <elementName> )*
```

If no foreign keys are specified explicitly, the elements of the target entity's designated primary key are used. Elements of the target entity that reside inside substructures can be addressed by means of the respective path. If the chosen elements do not form a unique key of the target entity, the association has cardinality to-many. The following examples show how to express foreign keys in an association.

```
entity Person
{
  key id : Integer;
  // address1,2,3 are to-one associations
  address1 : Association to Address;
  address2 : Association to Address { id };
  address3 : Association[1] to Address { zipCode, street, country };
  // address4,5,6 are to-many associations
  address4 : Association[0..*] to Address { zipCode };
  address5 : Association[*] to Address { street.name };
  address6 : Association[*] to Address { street.name AS streetName,
                                         country.name AS countryName };
};
```

Association Syntax Options

Association	Keys	Explanation
`address1`		No foreign keys are specified: the target entity's primary key (the element `id`) is used as foreign key.
`address2`	`{ id }`	Explicitly specifies the foreign key (the element `id`); this definition is identical to `address1`.
`address3`	`{ zipCode, street, country }`	The foreign key elements to be used for the association are explicitly specified, namely: `zipcode` and the structured elements `street` and `country`.
`address4`	`{ zipCode }`	Uses only `zipcode` as the foreign key. Since `zipcode` is not a unique key for entity `Address`, this association has cardinality "to-many".
`address5`	`{ street.name }`	Uses the sub-element `name` of the structured element `street` as a foreign key. This is not a unique key and, as a result, `address4` has cardinality "to-many".

Association	Keys	Explanation
`address6`	`{ street.name AS streetName, country.name AS countryName }`	Uses the sub-element `name` of both the structured elements `street` and `country` as foreign key fields. The names of the foreign key fields must be unique, so an alias is required here. The foreign key is not unique, so `address6` is a “to-many” association.

You can now use foreign keys of managed associations in the definition of other associations. In the following example, the compiler recognizes that the field `toCountry.cid` is part of the foreign key of the association `toLocation` and, as a result, physically present in the entity `Company`.

Sample Code

```
namespace samples;
@Schema: 'MYSCHEMA'         // XS classic *only*
context AssociationKeys {
    entity Country {
        key c_id : String(3);
        // <...>
    };
    entity Region {
        key r_id : Integer;
        key toCountry : Association[1] to Country { c_id };
        // <...>
    };
    entity Company {
        key id : Integer;
        toLocation : Association[1] to Region { r_id, toCountry.c_id };
        // <...>
    };
};
```

Unmanaged Associations

Unmanaged associations are based on existing elements of the source and target entity; no fields are generated. In the `ON` condition, only elements of the source or the target entity can be used; it is not possible to use other associations. The `ON` condition may contain any kind of expression - all expressions supported in views can also be used in the `ON` condition of an unmanaged association.

Note

The names in the `ON` condition are resolved in the scope of the source entity; elements of the target entity are accessed through the association itself .

In the following example, the association `inhabitants` relates the element `id` of the source entity `Room` with the element `officeId` in the target entity `Employee`. The target element `officeId` is accessed through the name of the association itself.

```
namespace samples;
@Schema: 'MYSCHEMA'             // XS classic *only*
context UnmanagedAssociations {
    entity Employee {
```

```
        key id : Integer;
        officeId : Integer;
        // <...>
    };
    entity Room {
        key id : Integer;
        inhabitants : Association[*] to Employee on inhabitants.officeId = id;
        // <...>
    };
    entity Thing {
        key id : Integer;
        parentId : Integer;
        parent : Association[1] to Thing on parent.id = parentId;
        children : Association[*] to Thing on children.parentId = id;
        // <...>
    };
};
```

The following example defines two related **unmanaged** associations:

- `parent`
  The unmanaged association `parent` uses a cardinality of `[1]` to create a relation between the element `parentId` and the target element `id`. The target element `id` is accessed through the name of the association itself.
- `children`
  The unmanaged association `children` creates a relation between the element `id` and the target element `parentId`. The target element `parentId` is accessed through the name of the association itself.

```
entity Thing {
  key id   : Integer;
  parentId : Integer;
  parent   : Association[1] to Thing on parent.id = parentId;
  children : Association[*] to Thing on children.parentId = id;
  ...
};
```

Constants in Associations

The usage of constants is no longer restricted to annotation assignments and default values for entity elements. With SPS 11, you can use constants in the "`ON`"-condition of unmanaged associations, as illustrated in the following example:

Sample Code

```
context MyContext {
  const MyIntConst      : Integer      = 7;
  const MyStringConst   : String(10)   = 'bright';
  const MyDecConst      : Decimal(4,2) = 3.14;
  const MyDateTimeConst : UTCDateTime  = '2015-09-30 14:33';
  entity MyEntity {
    key id : Integer;
    a : Integer;
    b : String(100);
    c : Decimal(20,10);
    d : UTCDateTime;
    your : Association[1] to YourEntity on your.a - a < MyIntConst;
  };
  entity YourEntity {
    key id : Integer;
    a : Integer;
  };
  entity HerEntity {
    key id : Integer;
```

```
    t : String(20);
  };
  view MyView as select from MyEntity
           inner join HerEntity on locate (b, :MyStringConst) > 0
  {
    a + :MyIntConst as x,
    b || ' is ' || :MyStringConst as y,
    c * sin(:MyDecConst) as z
  } where d < :MyContext.MyDateTimeConst;
};
```

Related Information

Create a CDS Association in XS Advanced [page 386]
CDS Associations [page 389]

5.3.4.6 Create a CDS View in XS Advanced

Define a design-time **view** using the Core Data Services (CDS) syntax.

Prerequisites

To complete this task successfully, note the following prerequisites:

- You must have access to an SAP HANA system.
- You must have access to the SAP Web IDE for SAP HANA

  > **i** Note
  >
  > The permissions defined in the XS advanced role collection `XS_AUTHORIZATION_USER` must be assigned to the user who wants to access the tools included in the SAP Web IDE for SAP HANA.

- You must have already a development workspace and a multi-target application (MTA) project.
- You must already have created a database module for your MTA application project.
- You must already have set up an HDI container for the CDS artifacts

  > **i** Note
  >
  > A container setup file (`.hdiconfig`) is required to define which plug-ins to use to create the corresponding catalog objects from the design-time artifacts when the multi-target application (or just the database module) is deployed.

- You must have access to the SAP HANA XS advanced run-time tools that enable you to view the contents of the catalog.

> **Note**
>
> The permissions defined in the XS advanced role collection `XS_AUTHORIZATION_USER` must be assigned to the user who wants to access the SAP HANA run-time tools.

Context

A view is a virtual table based on the dynamic results returned in response to an SQL statement. SAP HANA Extended Application Services for XS advanced model (XS advanced) enables you to use CDS syntax to create a database view as a design-time file. You can use this design-time view definition to generate the corresponding catalog object when you deploy the application that contains the view-definition artifact (or just the application's database module).

> **Note**
>
> The code examples provided are for illustration purposes only.

Procedure

1. Start the SAP HANA Web IDE for SAP HANA.

   The SAP Web IDE for SAP HANA is available at the following URL:

   `https://<HANA_HOST>:53075/`

   > **Tip**
   >
   > To display the URL for the SAP Web IDE for SAP HANA, open a command shell, log on to the XS advanced run time, and run the following command:
   >
   > `xs app webide --urls`

2. Open the application project to which you want to add your CDS entity.
3. Create the CDS document that will contain the view-definition.

   Browse to the folder in the database module in your application's project workspace, for example, `<MyApp1>/HDB/src` where you want to create the new CDS document with the view-definition file and perform the following steps:

   a. Right-click the folder where you want to save the CDS entity-definition file and choose *New* > *CDS Artifact* in the context-sensitive pop-up menu.
   b. Enter the name of the view-definition file in the *File Name* box, for example, `MyViewContext`.

      > **Tip**
      >
      > If you use the available setup Wizards to create your design-time artifacts, the correct file extensions is added automatically. The file extension is used to determine which plug-in to use to create the corresponding run-time object during deployment. CDS artifacts have the file extension `.hdbcds`, for example, `MyViewContext.hdbcds`.

c. Choose *Finish* to save the new CDS view-definition file in the database module of your application's local project workspace.

4. Define the underlying CDS entities and structured types for the SQL view.

   If the new CDS document is not automatically displayed by the file-creation wizard, double-click the CDS file you created in the previous step, for example, `MyViewContext.hdbcds`, and add the code that defines the underlying table entities and structured types:

```
context MyViewContext {
    type StreetAddress {
        name : String(80);
        number : Integer;
    };
    type CountryAddress {
        name : String(80);
        code : String(3);
    };
    entity Address {
        key id : Integer;
        street : StreetAddress;
        zipCode : Integer;
        city : String(80);
        country : CountryAddress;
        type : String(10); // home, office
    } technical configuration {
        column store;
    };
};
```

5. Define an SQL view as a projection of a CDS entity.

   In the same CDS document you edited in the previous step, for example, `MyViewContext.hdbcds`, add the code for the view `AddressView` to the end of the document below the entity `Address`.

   > **i** Note
   >
   > In CDS, a view is an entity without an its own persistence; it is defined as a projection of other entities.

```
view AddressView as select from Address
    {
        id,
        street.name,
        street.number
    };
};
```

6. Save the CDS document with the view-definition.

   Saving the definition persists the file in your local workspace; it does not create any objects in the database catalog.

7. Deploy the new view (and corresponding tables and types) in the catalog.

   To activate the CDS artifacts defined in the CDS document and generate the corresponding objects in the catalog, use the *Build* feature.

   a. Right-click the new database module in your application project.

   b. In the context-sensitive pop-up menu, choose *Build*.

      > → Tip
      >
      > You can follow the build progress in the console at the bottom of the CDS editor.

8. Check that the new table, type, and view objects have been successfully created in the catalog.

> → Tip
>
> A selection of run-time tools is available in the *Database Explorer* perspective of SAP web IDE for SAP HANA at the following location:
>
> *Tools* > *Database Explorer*

In XS advanced, your database run-time objects are located in the HDI container created for your multi-target application's database module; you need to locate and bind to this application-specific container to view its contents. The container name contains the name of the user logged into the SAP Web IDE for SAP HANA, the name of the database module containing the CDS design-time entities, and the string *-hdi-container*, for example:

`<XS_UserName>`*-ctetig24[...]-*`<DB_Module>`*-hdi-container*

To bind to the HDI container, in the SAP HANA run-time *Catalog* tool, right-click *Catalog* in the catalog list, and in the *Search HDI Containers* dialog, locate the container to which you want to bind, and choose *Bind*.

Related Information

Set up an SAP HDI Container for XS Advanced Applications [page 237]
Create the Data Persistence Artifacts with CDS in XS Advanced [page 311]
Create a CDS Document (XS Advanced) [page 313]
CDS View Syntax Options [page 406]

5.3.4.6.1 CDS Views

A view is an entity that is not persistent; it is defined as the projection of other entities. SAP HANA Extended Application Services (SAP HANA XS) enables you to create a CDS view as a design-time file in the repository.

SAP HANA Extended Application Services (SAP HANA XS) enables you to define a view in a CDS document, which you store as design-time file in the repository. Repository files can be read by applications that you develop. In addition, all repository files including your view definition can be transported to other SAP HANA systems, for example, in a delivery unit.

If your application refers to the design-time version of a view from the repository rather than the runtime version in the catalog, for example, by using the explicit path to the repository file (with suffix), any changes to the repository version of the file are visible as soon as they are committed to the repository. There is no need to wait for the repository to activate a runtime version of the view.

To define a transportable view using the CDS-compliant view specifications, use something like the code illustrated in the following example:

```
context Views {
    VIEW AddressView AS SELECT FROM Address {
        id,
        street.name,
```

```
        street.number
    };
<...>
}
```

When a CDS document is activated, the activation process generates a corresponding catalog object for each of the artifacts defined in the document; the location in the catalog is determined by the type of object generated. For example, in SAP HANA XS classic the corresponding catalog object for a CDS view definition is generated in the following location:

Views defined in a CDS document can make use of the following SQL features:

- CDS Type definition
- Expressions and functions (for example, "`a + b as theSum`")
- Aggregates, "GROUP BY", and "HAVING" clauses
- Associations (including filters and prefixes)
- ORDER BY, CASE, UNION, JOIN, and TOP
- With Parameters
- Select Distinct
- Spatial Data (for example, "`ST_Distance`")

> → Tip
>
> For more information about the syntax required when using these SQL features in a CDS view, see *CDS View Syntax Options* in *Related Information*.

Type Definition

In a CDS view definition, you can explicitly specify the `type` of a select item, as illustrated in the following example:

> Sample Code

```
type MyInteger : Integer;
entity E {
  a : MyInteger;
  b : MyInteger;
};
view V as select from E {
  a,
  a+b as s1,
  a+b as s2 : MyInteger
};
```

In the example of different `type` definitions, the following is true:

- `a,`
  Has type "`MyInteger`"
- `a+b as s1,`
  Has type "`Integer`" and any information about the user-defined `type` is lost

- `a+b as s2 : MyInteger`
  Has type “`MyInteger`”, which is explicitly specified

> **i Note**
>
> If necessary, a `CAST` function is added to the generated view in SAP HANA; this ensures that the `select` item's type in the SAP HANA view is the SAP HANA “type” corresponding to the explicitly specified CDS type.

Related Information

Create a CDS View in XS Advanced [page 401]
CDS View Syntax Options [page 406]
CDS Associations [page 389]

5.3.4.6.2 CDS View Syntax Options

SAP HANA XS includes a dedicated, CDS-compliant syntax, which you must adhere to when using a CDS document to define a view as a design-time artifact.

Example

> **i Note**
>
> The following example is intended for illustration purposes only and might contain syntactical errors. For further details about the keywords illustrated, click the links provided.

```
context views {
const x : Integer = 4;
const y : Integer = 5;
const Z : Integer = 6;
VIEW MyView1 AS SELECT FROM Employee
{
  a + b  AS theSum
};
VIEW MyView2 AS SELECT FROM Employee
{ officeId.building,
  officeId.floor,
  officeId.roomNumber,
  office.capacity,
  count(id) AS seatsTaken,
  count(id)/office.capacity as occupancyRate
} WHERE officeId.building = 1
  GROUP BY officeId.building,
           officeId.floor,
           officeId.roomNumber,
           office.capacity,
           office.type
```

```
  HAVING office.type = 'office' AND count(id)/office.capacity < 0.5;
VIEW MyView3 AS SELECT FROM Employee
{ orgUnit,
  salary
} ORDER BY salary DESC;
VIEW MyView4 AS SELECT FROM Employee {
    CASE
        WHEN a < 10 then 'small'
        WHEN 10 <= a AND a < 100 THEN 'medium'
        ELSE 'large'
    END AS size
};
VIEW MyView5 AS
  SELECT FROM E1 { a, b, c}
  UNION
  SELECT FROM E2 { z, x, y};
VIEW MyView6 AS SELECT FROM Customer {
  name,
  orders[status='open'].{ id   as orderId,
                          date as orderDate,
                          items[price>200].{ descr,
                                             price }  }
};
VIEW MyView7 as
  select from E { a, b, c}
  order by a limit 10 offset 30;
VIEW V_join as select from E join (F as X full outer join G on X.id = G.id) on
E.id = c {
  a, b, c
};
VIEW V_top as select from E TOP 10 { a, b, c};
VIEW V_dist as select from E distinct { a };
VIEW V_param with parameters PAR1: Integer, PAR2: MyUserDefinedType, PAR3: type
of E.elt
      as select from MyEntity {
        id,
        elt };
VIEW V_type as select from E {
  a,
  a+b as s1,
  a+b as s2 : MyInteger
};
view VE as select from E mixin {
  f : Association[1] to VF on f.vy = $projection.vb;
 } into {
  a as va,
  b as vb,
  f as vf
 };
VIEW SpatialView1 as select from Person {
    name,
    homeAddress.street_name   || ', ' || homeAddress.city   as home,
    officeAddress.street_name || ', ' || officeAddress.city as office,
    round( homeAddress.loc.ST_Distance(officeAddress.loc, 'meter')/1000, 1) as
distanceHomeToWork,
    round( homeAddress.loc.ST_Distance(NEW ST_POINT(8.644072, 49.292910),
'meter')/1000, 1) as distFromSAP03
};
}
```

Expressions and Functions

In a CDS view definition you can use any of the functions and expressions listed in the following example:

```
View MyView9 AS SELECT FROM SampleEntity
{
  a + b  AS theSum,
  a - b  AS theDifference,
  a * b  AS theProduct,
  a / b  AS theQuotient,
  -a     AS theUnaryMinus,
  c || d AS theConcatenation
};
```

> **Note**
>
> When expressions are used in a view element, an alias must be specified, for example, `AS theSum`.

Aggregates

In a CDS view definition, you can use the following aggregates:

- AVG
- COUNT
- MIN
- MAX
- SUM
- STDDEV
- VAR

The following example shows how to use aggregates and expressions to collect information about headcount and salary per organizational unit for all employees hired from 2011 to now.

```
VIEW MyView10 AS SELECT FROM Employee
{
  orgUnit,
  count(id)   AS headCount,
  sum(salary) AS totalSalary,
  max(salary) AS maxSalary
}
  WHERE joinDate > date'2011-01-01'
  GROUP BY orgUnit;
```

> **Note**
>
> Expressions are not allowed in the `GROUP BY` clause.

Constants in Views

With SPS 11, you can use constants in the views, as illustrated in “MyView” at the end of the following example:

Sample Code

```
context MyContext {
  const MyIntConst      : Integer      = 7;
  const MyStringConst   : String(10)   = 'bright';
  const MyDecConst      : Decimal(4,2) = 3.14;
  const MyDateTimeConst : UTCDateTime  = '2015-09-30 14:33';
  entity MyEntity {
    key id : Integer;
    a : Integer;
    b : String(100);
    c : Decimal(20,10);
    d : UTCDateTime;
    your : Association[1] to YourEntity on your.a - a < MyIntConst;
  };
  entity YourEntity {
    key id : Integer;
    a : Integer;
  };
  entity HerEntity {
    key id : Integer;
    t : String(20);
  };
  view MyView as select from MyEntity
           inner join HerEntity on locate (b, :MyStringConst) > 0
  {
    a + :MyIntConst as x,
    b || ' is ' || :MyStringConst as y,
    c * sin(:MyDecConst) as z
  } where d < :MyContext.MyDateTimeConst;
};
```

When constants are used in a view definition, their name must be prefixed with the scope operator “`:`”. Usually names that appear in a query are resolved as alias or element names. The scope operator instructs the compiler to resolve the name outside of the query.

Sample Code

```
context NameResolution {
  const a : Integer = 4;
  const b : Integer = 5;
  const c : Integer = 6;
  entity E {
    key id : Integer;
    a : Integer;
    c : Integer;
  };
  view V as select from E {
    a    as a1,
    b,
    :a   as a2,
    E.a  as a3,
    :E,
    :E.a as a4,
    :c
  };
}
```

The following table explains how the constants used in view “V” are resolved.

Constant Declaration and Result

Constant Expression	Result	Comments
`a as a1,`	Success	“a” is resolved in the space of alias and element names, for example, element “a” of entity “E”.
`b,`	Error	There is no alias and no element with name “b” in entity “E”
`:a as a2,`	Success	Scope operator “:” instructs the compiler to search for element “a” outside of the query (finds the constant “a”).
`E.a as a3,`	Success	“E” is resolved in the space of alias and element names, so this matches element “a” of entity “Entity” .
`:E,`	Error	Error: no access to “E” via “:”
`:E.a as a4,`	Error	Error; no access to “E” (or any of its elements) via “:”
`:c`	Error	Error: there is no alias for “c”.

SELECT

In the following example of an association in a `SELECT` list, a view compiles a list of all employees; the list includes the employee's name, the capacity of the employee's office, and the color of the carpet in the office. The association follows the to-one association office from entity `Employee` to entity `Room` to collect the relevant information about the office.

```
VIEW MyView11 AS SELECT FROM Employee
{
  name.last,
  office.capacity,
  office.carpetColor
};
```

Subqueries

You can define subqueries in a CDS view, as illustrated in the following example:

! Restriction

For use in XS advanced only; subqueries are not supported in XS classic

Code Syntax

```
select from (select from F {a as x, b as y}) as Q {
  x+y as xy,
    (select from E {a} where b=Q.y) as a
} where x < all (select from E{b})
```

> **Note**
>
> In a **correlated** subquery, elements of outer queries must always be addressed by means of a table alias.

WHERE

The following example shows how the syntax required in the `WHERE` clause used in a CDS view definition. In this example, the `WHERE` clause is used in an association to restrict the result set according to information located in the association's target. Further filtering of the result set can be defined with the `AND` modifier.

```
VIEW EmployeesInRoom_ABC_3_4 AS SELECT FROM Employee
{
  name.last
} WHERE officeId.building = 'ABC'
    AND officeId.floor    = 3
    AND officeId.number   = 4;
```

FROM

The following example shows the syntax required when using the FROM clause in a CDS view definition. This example shows an association that lists the license plates of all company cars.

```
VIEW CompanyCarLicensePlates AS SELECT FROM Employee.companyCar
{
  licensePlate
};
```

In the `FROM` clause, you can use the following elements:

- an entity or a view defined in the same CDS source file
- a native SAP HANA table or view that is available in the schema specified in the schema annotation (`@Schema` in the corresponding CDS document)

If a CDS view references a native SAP HANA table, the table and column names must be specified using their effective SAP HANA names.

```
create table foo (
  bar    : Integer,
  "gloo" : Integer
)
```

This means that if a table (`foo`) or its columns (`bar` and `"gloo"` were created **without** using quotation marks (""), the corresponding uppercase names for the table or columns must be used in the CDS document, as illustrated in the following example.

```
VIEW MyViewOnNative as SELECT FROM FOO
{
  BAR,
  gloo
};
```

GROUP BY

The following example shows the syntax required when using the `GROUP BY` clause in a CDS view definition. This example shows an association in a view that compiles a list of all offices that are less than 50% occupied.

```
VIEW V11 AS SELECT FROM Employee
{
  officeId.building,
  officeId.floor,
  officeId.roomNumber,
  office.capacity,
  count(id) as seatsTaken,
  count(id)/office.capacity as occupancyRate
} GROUP BY officeId.building,
           officeId.floor,
           officeId.roomNumber,
           office.capacity,
           office.type
  HAVING office.type = 'office' AND count(id)/capacity < 0.5;
```

HAVING

The following example shows the syntax required when using the `HAVING` clause in a CDS view definition. This example shows a view with an association that compiles a list of all offices that are less than 50% occupied.

```
VIEW V11 AS SELECT FROM Employee
{
  officeId.building,
  officeId.floor,
  officeId.roomNumber,
  office.capacity,
  count(id)          as seatsTaken,
  count(id)/office.capacity as occupancyRate
} GROUP BY officeId.building,
           officeId.floor,
           officeId.roomNumber,
           office.capacity,
           office.type
  HAVING office.type = 'office' AND count(id)/capacity < 0.5;
```

ORDER BY

The `ORDER BY` operator enables you to list results according to an expression or position, for example `salary`.

```
VIEW MyView3 AS SELECT FROM Employee
{
  orgUnit,
  salary
} ORDER BY salary DESC;
```

In the same way as with plain SQL, the `ASC` and `DESC` operators enable you to sort the list order as follows.

- `ASC`

Display the result set in **ascending** order

- `DESC`
  Display the result set in **descending** order

LIMIT/OFFSET

You can use the SQL clauses `LIMIT` and `OFFSET` in a CDS query. The `LIMIT <INTEGER> [OFFSET <INTEGER>]` operator enables you to restrict the number of output records to display to a specified "limit"; the `OFFSET <INTEGER>` specifies the number of records to skip before displaying the records according to the defined `LIMIT`.

```
VIEW MyViewV AS SELECT FROM E
  { a, b, c}
  order by a limit 10 offset 30;
```

CASE

In the same way as in plain SQL, you can use the `case` expression in a CDS view definition to introduce `IF-THEN-ELSE` conditions without the need to use procedures.

```
entity MyEntity12 {
key id : Integer;
     a : Integer;
     color : String(1);
};

VIEW MyView12 AS SELECT FROM MyEntity12 {
    id,
    CASE color     // defined in MyEntity12
        WHEN 'R' THEN 'red'
        WHEN 'G' THEN 'green'
        WHEN 'B' THEN 'blue'
        ELSE 'black'
    END AS color,
    CASE
        WHEN a < 10 then 'small'
        WHEN 10 <= a AND a < 100 THEN 'medium'
        ELSE 'large'
    END AS size
};
```

In the first example of usage of the `CASE` operator, `CASE color` shows a "switched" CASE (one table column and multiple values). The second example of CASE usage shows a "conditional" CASE with multiple arbitrary conditions, possibly referring to different table columns.

UNION

Enables multiple select statements to be combined but return only one result set. `UNION` works in the same way as the SAP HANA SQL command of the same name; it selects all unique records from all select statements

by removing duplicates found from different select statements.The signature of the result view is equal to the signature of the first `SELECT` in the union.

> **Note**
>
> View `MyView5` has elements `a`, `b`, and `c`.

```
entity E1 {
  key a : Integer;
  b : String(20);
  c : LocalDate;
};
entity E2 {
  key x : String(20);
  y : LocalDate;
  z : Integer;
};
VIEW MyView5 AS
  SELECT FROM E1 { a, b, c}
  UNION
  SELECT FROM E2 { z, x, y};
```

JOIN

You can include a JOIN clause in a CDS view definition; the following JOIN types are supported:

- [INNER] JOIN
- LEFT [OUTER] JOIN
- RIGHT [OUTER] JOIN
- FULL [OUTER] JOIN
- CROSS JOIN

The following example shows a simple join.

> **Sample Code**

```
entity E {
  key id : Integer;
  a : Integer;
};
entity F {
  key id : Integer;
  b : Integer;
};
entity G {
  key id : Integer;
  c : Integer;
};
view V_join as select from E join (F as X full outer join G on X.id = G.id)
on E.id = c {
  a, b, c
};
```

TOP

You can use the SQL clause `TOP` in a CDS query, as illustrated in the following example:

Sample Code

```
view V_top as select from E TOP 10 { a, b, c};
```

! Restriction

It is not permitted to use `TOP` in combination with the `LIMIT` clause in a CDS query.

SELECT DISTINCT

CDS now supports the `SELECT DISTINCT` semantic, which enables you to specify that only one copy of each set of duplicate records **selected** should be returned. The position of the `DISTINCT` keyword is important; it must appear directly in front of the curly brace, as illustrated in the following example:

Sample Code

```
entity E {
  key id : Integer;
  a : Integer;
};
entity F {
  key id : Integer;
  b : Integer;
};
entity G {
  key id : Integer;
  c : Integer;
};
view V_dist as select from E distinct { a };
```

With Parameters

You can define parameters for use in a CDS view; this allows you to pass additional values to modify the results of the query at run time. Parameters must be defined in the view definition before the query block, as illustrated in the following example:

! Restriction

For use in XS advanced only; views with parameters are not supported in XS classic.

Sample Code

Parameters in a CDS View

```
context MyContext
{
  entity MyEntity1 {
     id: Integer;
     elt: String(100); };
  entity MyEntity2 {
     id: Integer;
     elt: String(100); };
  type MyUserDefinedType: type of E.elt;
  view MyParamView with parameters PAR1: Integer,
                                   PAR2: MyUserDefinedType,
                                   PAR3: type of E.elt
      as select from MyEntity {
        id,
        elt };
```

Note

Keywords are case insensitive.

Parameters in View Queries

Parameters can be used in a query at any position where an expression is allowed. A parameter is referred to inside a query by prefixing the parameter name either with the colon Scope operator ':' or the string "`$parameters`" .

Tip

If no matching parameter can be found, the scope operator "escapes" from the query and attempts to resolve the identifier outside the query.

Sample Code

Using Parameters in a View Query

```
view ExampleView with parameters PAR1: Integer,
                                 PAR2: UserDefinedType,
                                 PAR3: type of E.elt
       as select from SomeEntity
            left outer join SomeOtherEntity
              on SomeEntity.id < SomeOtherEntity.id + :PAR1
    {
      id + :PAR1 as idWithOffset,
      elt,
      :PAR1,
      $parameters.PAR3
    } where elt != $parameters.PAR2;
```

Invoking a View with Parameters

Parameters are passed to views as a comma-separated list in parentheses. Optional filter expressions must then follow the parameter list.

! Restriction

It is not allowed to use a query as value expression. Nor is it allowed to provide a parameter list in the `ON` condition of an association definition to a parameterized view. This is because the association definition establishes the relationship between the two entities but makes no assumptions about the run-time conditions. For the same reason, it is not allowed to specify filter conditions in those `ON` conditions.

The following example shows two entities `SourceEntity` and `TargetEntity` and a parameterized view `TargetWindowView`, which selects from `TargetEntity`. An association is established between `SourceEntity` and `TargetEntity`.

‹≡› Sample Code

```
entity SourceEntity {
   id: Integer;
   someElementOfSourceEntity: String(100);
   toTargetViaParamView: association to TargetWindowView on
      toTargetViaParamView.targetId = id;
   };
entity TargetEntity {
   targetId: Integer;
   someElementOfTargetEntity: String(100);
   };

view TargetWindowView with parameters LOWER_LIMIT: Integer
   as select from TargetEntity {
      targetId,
      someElementOfTargetEntity
   } where targetId > :LOWER_LIMIT
     and targetId <= :LOWER_LIMIT + 10;
```

It is now possible to query `SourceEntity` in a view; it is also possible to follow the association to `TargetWindowView`, for example, by providing the required parameters, as illustrated in the following example:

‹≡› Sample Code

Query a Parameterized CDS View (with Association)

```
view SourceConsumption with parameters CUSTOMER_ID: Integer
   as select from SourceEntity {
      someElementOfSourceEntity,
      toTargetViaParamView(LOWER_LIMIT:
$parameters.CUSTOMER_ID).someElementOfTargetEntity
   };
```

It is also possible to follow the association in the `FROM` clause; this provides access only to the elements of the target artifact:

‹≡› Sample Code

Follow an Association in the FROM Clause

```
view ConsumptionView with parameters CUSTOMER_ID: Integer
   as select from SourceEntity.toTargetViaParamView(LOWER_LIMIT: :CUSTOMER_ID)
   {
      id,
      someElementOfTargetEntity
```

```
    };
```

You can select directly from the view with parameters, adding a free `JOIN` expression, as illustrated in the following example:

‹≡› Sample Code

Select from a Parameterized View with JOIN Expression

```
view ConsumptionView with parameters CUSTOMER_ID: Integer
   as select from TargetWindowView(LOWER_LIMIT: :CUSTOMER_ID) as TWV_ALIAS
      RIGHT OUTER JOIN ... ON TWV_ALIAS.targetId ....
{
   ...
};
```

Annotations in Parameter Definitions

Parameter definitions can be annotated in the same way as any other artifact in CDS; the annotations must be prepended to the parameter name. Multiple annotations are separated either by whitespace or new-line characters.

→ Tip

To improve readability and comprehension, it is recommended to include only one annotation assignment per line.

In the following example, the view `TargetWindowView` selects from the entity `TargetEntity`; the annotation `@positiveValuesOnly` is not checked; and the `targetId` is required for the `ON` condition in the entity `SourceEntity`.

‹≡› Sample Code

Annotation Assignments to Parameter Definitions in CDS Views

```
annotation remark: String(100);

view TargetWindowView with parameters
   @remark: 'This is an arbitrary annotation'
   @positiveValuesOnly: true
   LOWER_LIMIT: Integer
as select from TargetEntity
{
  targetId,
  ....
} where targetId > :LOWER_LIMIT and targetId <= :LOWER_LIMIT + 10;
```

Associations, Filters, and Prefixes

You can define an association as a view element, for example, by defining an ad-hoc association in the `mixin` clause and then adding the association to the `SELECT` list, as illustrated in the following example:

> **! Restriction**
>
> XS classic does not support the use of ad-hoc associations in a view's `SELECT` list.

> **Sample Code**
>
> Associations as View Elements

```
entity E {
  a : Integer;
  b : Integer;
 };
entity F {
  x : Integer;
  y : Integer;
 };
view VE as select from E mixin {
  f : Association[1] to VF on f.vy = $projection.vb;
 } into {
  a as va,
  b as vb,
  f as vf
 };
view VF as select from F {
  x as vx,
  y as vy
 };
```

In the `ON` condition of this type of association in a view, it is necessary to use the pseudo-identifier `$projection` to specify that the following element name must be resolved in the `select` list of the view ("`VE`") rather than in the entity ("`E`") in the `FROM` clause

Filter Conditions

It is possible to apply a filter condition when resolving associations between entities; the filter is merged into the `ON`-condition of the resulting `JOIN`. The following example shows how to get a list of customers and then filter the list according to the sales orders that are currently "open" for each customer. In the example, the filter is inserted after the association `orders`; this ensures that the list displayed by the view only contains those orders that satisfy the condition `[status='open']`.

> **Sample Code**

```
view C1 as select from Customer {
  name,
  orders[status='open'].id as orderId
};
```

The following example shows how to use the prefix notation to ensure that the compiler understands that there is only one association (`orders`) to resolve but with multiple elements (`id` and `date`):

Sample Code

```
view C1 as select from Customer {
  name,
  orders[status='open'].{ id   as orderId,
                          date as orderDate
                        }
};
```

→ Tip

Filter conditions and prefixes can be nested.

The following example shows how to use the associations `orders` and `items` in a view that displays a list of customers with open sales orders for items with a price greater than 200.

Sample Code

```
view C2 as select from Customer {
  name,
  orders[status='open'].{ id   as orderId,
                          date as orderDate,
                          items[price>200].{ descr,
                                             price
                                           }
                        }
};
```

Prefix Notation

The prefix notation can also be used without filters. The following example shows how to get a list of all customers with details of their sales orders. In this example, all uses of the association `orders` are combined so that there is only one `JOIN` to the table `SalesOrder`. Similarly, both uses of the association `items` are combined, and there is only one `JOIN` to the table `Item`.

Sample Code

```
view C3 as select from Customer {
  name,
  orders.id          as orderId,
  orders.date        as orderDate,
  orders.items.descr as itemDescr,
  orders.items.price as itemPrice
};
```

The example above can be expressed more elegantly by combining the associations `orders` and `items` using the following prefix notation:

Sample Code

```
view C1 as select from Customer {
  name,
  orders.{ id   as orderId,
           date as orderDate,
```

```
            items. { descr as itemDescr,
                     price as itemPrice
                   }
          }
};
```

Type Definition

In a CDS view definition, you can explicitly specify the `type` of a select item, as illustrated in the following example:

> **! Restriction**
>
> For use in XS advanced only; assigning an explicit CDS type to an item in a `SELECT` list is not supported in XS classic.

> **Sample Code**

```
type MyInteger : Integer;
entity E {
  a : MyInteger;
  b : MyInteger;
};
view V as select from E {
  a,
  a+b as s1,
  a+b as s2 : MyInteger
};
```

In the example of different `type` definitions, the following is true:

- `a,`
  Has type "`MyInteger`"
- `a+b as s1,`
  Has type "`Integer`" and any information about the user-defined `type` is lost
- `a+b as s2 : MyInteger`
  Has type "`MyInteger`", which is explicitly specified

> **i Note**
>
> If necessary, a `CAST` function is added to the generated view in SAP HANA; this ensures that the `select` item's type in the SAP HANA view is the SAP HANA "type" corresponding to the explicitly specified CDS type.

Spatial Functions

The following view (`SpatialView1`) displays a list of all persons selected from the entity `Person` and uses the spatial function `ST_Distance` (`*`) to include information such as the distance between each person's

home and business address (`distanceHomeToWork`), and the distance between their home address and the building SAP03 (`distFromSAP03`). The value for both distances is measured in kilometers, which is rounded up and displayed to one decimal point.

Sample Code

```
view SpatialView1 as select from Person {
    name,
    homeAddress.street_name   || ', ' || homeAddress.city   as home,
    officeAddress.street_name || ', ' || officeAddress.city as office,
    round( homeAddress.loc.ST_Distance(officeAddress.loc, 'meter')/1000, 1)
as distanceHomeToWork,
    round( homeAddress.loc.ST_Distance(NEW ST_POINT(8.644072, 49.292910),
'meter')/1000, 1) as distFromSAP03
};
```

Caution

(*) For information about the capabilities available for your license and installation scenario, refer to the Feature Scope Description for SAP HANA.

Related Information

Create a View in CDS
Create a View in CDS
Spatial Types and Functions [page 422]

5.3.4.6.3 Spatial Types and Functions

CDS supports the use of Geographic Information Systems (GIS) functions and element types in CDS-compliant entities and views.

Spatial data is data that describes the position, shape, and orientation of objects in a defined space; the data is represented as two-dimensional geometries in the form of points, line strings, and polygons. The following examples shows how to use the spatial function `ST_Distance` in a CDS view. The underlying spatial data used in the view is defined in a CDS entity using the type `ST_POINT`.

The following example, the CDS entity `Address` is used to store geo-spatial coordinates in element `loc` of type `ST_POINT`:

Sample Code

```
namespace samples;
@Schema: 'MYSCHEMA'
context Spatial {
    entity Person {
        key id : Integer;
        name : String(100);
        homeAddress : Association[1] to Address;
        officeAddress : Association[1] to Address;
```

```
    };
    entity Address {
        key id : Integer;
        street_number : Integer;
        street_name : String(100);
        zip : String(10);
        city : String(100);
        loc : hana.ST_POINT(4326);
    };
    view GeoView1 as select from Person {
        name,
        homeAddress.street_name || ', ' || homeAddress.city as home,
        officeAddress.street_name || ', ' || officeAddress.city as office,
        round( homeAddress.loc.ST_Distance(officeAddress.loc, 'meter')/1000,
1) as distanceHomeToWork,
        round( homeAddress.loc.ST_Distance(NEW ST_POINT(8.644072, 49.292910),
'meter')/1000, 1) as distFromSAP03
    };
};
```

The view `GeoView1` is used to display a list of all persons using the spatial function `ST_Distance` to include information such as the distance between each person's home and business address (`distanceHomeToWork`), and the distance between their home address and the building SAP03 (`distFromSAP03`). The value for both distances is measured in kilometers.

> ⚠ Caution
>
> (*) For information about the capabilities available for your license and installation scenario, refer to the Feature Scope Description for SAP HANA.

Related Information

Create a View in CDS
Create a View in CDS
CDS View Syntax Options [page 406]
CDS Entity Syntax Options
CDS Primitive Data Types [page 382]

5.3.4.7 Create a CDS Extension

Define the artifacts required to extend an existing CDS model.

Prerequisites

- You have access to the Web-based development tools included with SAP Web IDE for SAP HANA

- Each CDS extension package must have the following elements:
  - The package descriptor (`.package.hdbcds`)
    The package descriptor for a CDS extension has no name, only the suffix. Its contents must conform to the required syntax.
  - The CDS extension descriptor (`myCDSExtension.hdbcds`)
    The extension descriptor's contents must conform to the required syntax.

Context

In this simple CRM scenario, the base application consists of a "type" named `Address` and an entity named `Customer`. In the first extension package, `banking`, we add a new "`type`" named `BankingAccount` and a new "`element`" named `account` to the entity `Customer`. In a further extension package named `onlineBanking` that depends on the package `banking` we add a new element to type `BankingAccount` and add a new element to `type` `Address`.

Sample Code

```
<myCDSBankingApp>
|- db/                                    # Database deployment artifacts
|  |- package.json
|  \- src/
|     |- .hdiconfig                       # HDI build plug-in configuration
|     |- .hdinamespace                    # HDI run-time name-space config
|     |- Address.hdbcds                   # Address type definition
|     |- CRM.hdbcds                       # Address details
|     |- banking/                         # CDS extension package
|     |  |- package.hdbcds                # CDS extension package
descriptor
|     |  \- BankingExtension.hdbcds       # CDS extension description
|     \- onlineBanking/                   # CDS extension package
|        |- package.hdbcds                # CDS extension package
descriptor
|        \- OnlineBankingExtension.hdbcds  # CDS extension description
|- web/
|- js/
|- xs-security.json
\- mtad.yaml
```

Procedure

1. Start the SAP HANA Web IDE for SAP HANA.

   The SAP Web IDE for SAP HANA is available at the following URL:

   `https://<HANA_HOST>:53075/`

   → Tip

   To display the URL for the SAP Web IDE for SAP HANA, open a command shell, log on to the XS advanced run time, and run the following command:

```
xs app webide --urls
```

2. Display the application project to which you want to add a CDS document.

   In XS advanced, SAP Web IDE for SAP HANA creates an application within a context of a project. If you do not already have a project, there are a number of ways to create one, for example: by importing it, cloning it, or creating a new one from scratch.

   a. In the SAP Web IDE for SAP HANA, choose *File* > *New* > *Project from Template*.
   b. Choose the project template type.

      Currently, there is only one type of project template available, namely: *Multi-Target Application Project*. Select *Multi-Target Application Project* and choose *Next*.
   c. Type a name for the new MTA project (for example, **myCDSApp** and choose *Next* to confirm.
   d. Specify details of the new MTA project and choose *Next* to confirm.
   e. Create the new MTA project; choose *Finish*.

3. Navigate to the database module of the application for which you want to create CDS extensions.

   In SAP Web IDE for SAP HANA, database artifacts such as the ones defined in a CDS document belong in the MTA's database "module".

   > → Tip
   >
   > If you do not already have a database module, right-click the root folder of your new MTA project and, in the context menu, choose *New* > *HDB Module*. Name the new database model **db**.

4. Create the base database application using CDS.

   The base CDS application in `myCDSBankingApp/src/` must contain the following artifacts:

   > ℹ Note
   >
   > For the purposes of this tutorial, we are using the base CDS documents `Address.hdbcds` and `CRM.hdbcds`; an existing application would have different CDS documents. However, the `.hdinamespace` and `.hdiconfig` files would be present.

   - `.hdinamespace`
     The name-space definition to use when deploying the database application
   - `.hdiconfig`
     The list of plug-ins to use to create catalog objects when deploying the database application
   - `Address.hdbcds`
     A CDS document containing the definition of the CDS data "`type`" `Address`
   - `CRM.hdbcds`
     A CDS document named `CDM.hdbcds` which contains the definition of the CDS "`entity`" `Customer`

   a. Define the name space that applies to this CDS application.

      The name space to use for the deployment of the CDS application is defined in the configuration file `.hdinamespace`; in this case, it should look like the following example:

      > ℹ Note
      >
      > The "`append`" value ensures that the name-space rule applies to *all* subfolders in the CDS application structure.

Sample Code

`.hdinamespace`

```
{
  "name": "",
  "subfolder": "append"
}
```

b. Create the CDS data type `Address.hdbcds`.

Navigate to the application's database module `db/`, right-click the folder `db/src/` and choose *New* > *CDS Artifact* in the context menu.

The CDS data type definition should look like the following example:

Sample Code

```
type Address {
  zipCode : String(5);
  city : String(40);
  street : String(40);
  nr : String(10);
};
```

c. Create the CDS document `CRM.hdbcds`.

Navigate to the `src/` folder in your application's database module `db/`, right-click the folder `db/src/` and choose *New* > *CDS Artifact* in the context menu.

The CDS definition for the `Customer` entity (table) should look like the following example:

Sample Code

```
using Address;
context CRM {
  entity Customer {
    name : String(40);
    address : Address;
  };
};
```

5. Create a CDS extension called "`banking`".

The CDS extension `banking` must contain the following artifacts:

- `.package.hdbcds`
  A CDS document containing the definition of the CDS extension package `banking/`
- `BankingExtension.hdbcds`
  A CDS document containing the definition of the CDS extension `BankingExtension`

a. Create a new folder for the CDS extension `banking/`.

Navigate to the `src/` folder in your application's database module `db/`, right-click the folder `db/src/` and choose *New* > *Folder* in the context menu. Name the new folder "`banking`".

b. Create the CDS extension package descriptor `.package.hdbcds`.

> **Note**
>
> The leading dot (.) in the extension-package file name is mandatory.

Navigate to the `src/` folder in your application's database module `db/`, right-click the folder `db/src/banking` and choose *New* > *CDS Artifact* in the context menu. Name the new artifact `.package.hdbcds`.

The CDS extension **package** definition should look like the following example:

> **Sample Code**

```
package banking;
```

c. Create the CDS extension descriptor `BankingExtension.hdbcds`.

Navigate to the `src/` folder in your application's database module `db/`, right-click the folder `db/src/banking` and choose *New* > *CDS Artifact* in the context menu.

The CDS extension definition `BankingExtension.hdbcds` should look like the following example:

> **Sample Code**

```
namespace banking;
in package banking;
using CRM;
using CRM.Customer;
extend context CRM with {
  type BankingAccount {
    BIC : String(8);
    IBAN : String(120);
  };
};
extend entity Customer with {
  account: CRM.BankingAccount;
};
```

6. Create a CDS extension called "`onlineBanking`".

The CDS extension `onlineBanking` must contain the following artifacts:

- `.package.hdbcds`
  A CDS document containing the description of the CDS extension package `onlineBanking/`
- `BankingExtension.hdbcds`
  A CDS document containing the description of the CDS extension `OnlineBankingExtension`

a. Create a new folder for the CDS extension `onlineBanking/`.

Navigate to the `src/` folder in your application's database module `db/`, right-click the folder `db/src/` and choose *New* > *Folder* in the context menu. Name the new folder "`onlineBanking`".

b. Create the CDS extension package descriptor `.package.hdbcds`.

> **Note**
>
> The leading dot (.) in the package file name is mandatory.

Navigate to the `src/` folder in your application's database module `db/`, right-click the folder `db/src/onlineBanking` and choose *New* > *CDS Artifact* in the context menu. Name the new artifact `.package.hdbcds`.

The CDS extension **package** descriptor should look like the following example:

Sample Code

```
package onlineBanking depends on banking;
```

c. Create the CDS extension descriptor `BankingExtension.hdbcds`.

Navigate to the `src/` folder in your application's database module `db/`, right-click the folder `db/src/onlineBanking` and choose *New* > *CDS Artifact* in the context menu. Name the new CDS artifact `BankingExtension.hdbcds`

The CDS extension definition `BankingExtension.hdbcds` should look like the following example:

Sample Code

```
namespace onlineBanking;
in package onlineBanking;
using Address;
using CRM.BankingAccount;
extend type Address with {
  email : String(60);
};
extend type BankingAccount with {
  PIN : String(5);
};
```

7. Build the CDS application's database module.

Building a database module activates the data model and creates corresponding object in the database catalog for each artifact defined in the CDS document. In this case, the build creates all the CDS artifacts for the base CDS application as well as the artifacts defined in the two extension packages.

In SAP Web IDE for SAP H ANA, right-click the CDS application's database module (`<myCDSapp>/db`) and choose *Build* > *Build* in the context-sensitive menu.

If the builder displays the message `(Builder) Build of /<myCDSapp>/db` completed in the SAP Web IDE for SAP HANA console, the data-model was successfully activated in a SAP HANA database container, and can now be used to store and retrieve data.

Related Information

The CDS Extension Descriptor [page 429]
The CDS Extension Descriptor Syntax [page 432]
The CDS Extension Package Descriptor [page 436]
The CDS Extension Package Descriptor Syntax [page 437]

5.3.4.7.1 The CDS Extension Descriptor

Defines in a separate file the properties required to modify an existing CDS artifact definition.

The CDS extension mechanism allows you to add properties to existing artifact definitions without modifying the original source files. In this way, you can split the definition of an artifact across multiple files each of which can have a different life cycle and code owner. For example, a customer can add a new element to an existing entity definition by the following statement:

‹≡› Sample Code

CDS Artifact Extension Syntax

```
extend EntityE with {
  newElement: Integer;
}
```

In the example above, the code illustrated shows how to define a new **element** inside an existing entity (`EntityE`) artifact.

i Note

The `extend` statement changes an existing artifact; it does not define any additional artifact.

It is essential to ensure that additional element definitions specified in custom extensions do not break the existing definitions of the base application. This is achieved by adapting the name-search rules and by additional checks for the `extend` statements. For the definition of these rules and checks, it is necessary to define the relationship between an `extend` statement and the artifact definitions, as well as the relationship between an `extend` statement and any additional `extend` statements.

Organization of Extensions

When you extend an SAP application, you typically add new elements to entities or views; these additional elements usually work together and can, themselves, require additional artifacts, for example, “types” used as element “types”. To facilitate the process, we define an extension package (or package for short), which is a set of extend statements, normal artifact definitions (for example, “types” which are used in an `extend` declaration), and extension relationships (also known as “dependencies”). Each CDS source file belongs to exactly one package; all the definitions in this file contribute to that one (single) package. However, a “package” typically contains contributions from multiple CDS source files.

→ Tip

It is also possible to use a package to define a clear structure for an application, even if no extensions are involved.

Package Hierarchies

The extension mechanism can be used by developers as well as SAP industry solutions, partners, and customers. A productive system is likely to have more than one package; some packages might be independent from each other; some packages might depend on other packages. With such a model, we get an acyclic directed graph, with the base application and the extension packages as nodes and the dependencies as edges. This induces a partial order on the packages with the base application as lowest package (for simplicity we also call the base application a package). There is not necessarily a single top package (here: the final customer extension).

It is essential to ensure that which package is semantically self-contained and self-explanatory; avoid defining "micro" packages which can be technically applied individually but have no independent business value.

> ! Restriction
>
> Cyclic dependencies between extension packages are not allowed.

Package Definition

It is necessary to specify which `extend` statements and normal artifact definitions belong to which package and, in addition, on which other packages a package depends. A package is considered to be a normal CDS artifact; it has a name, and a corresponding definition, and its use can be found in the CDS Catalog. An extension package is defined by a special CDS source file with the file suffix `.package.hdbcds`.

> i Note
>
> The full stop (.) **before** the extension-package file name is mandatory.

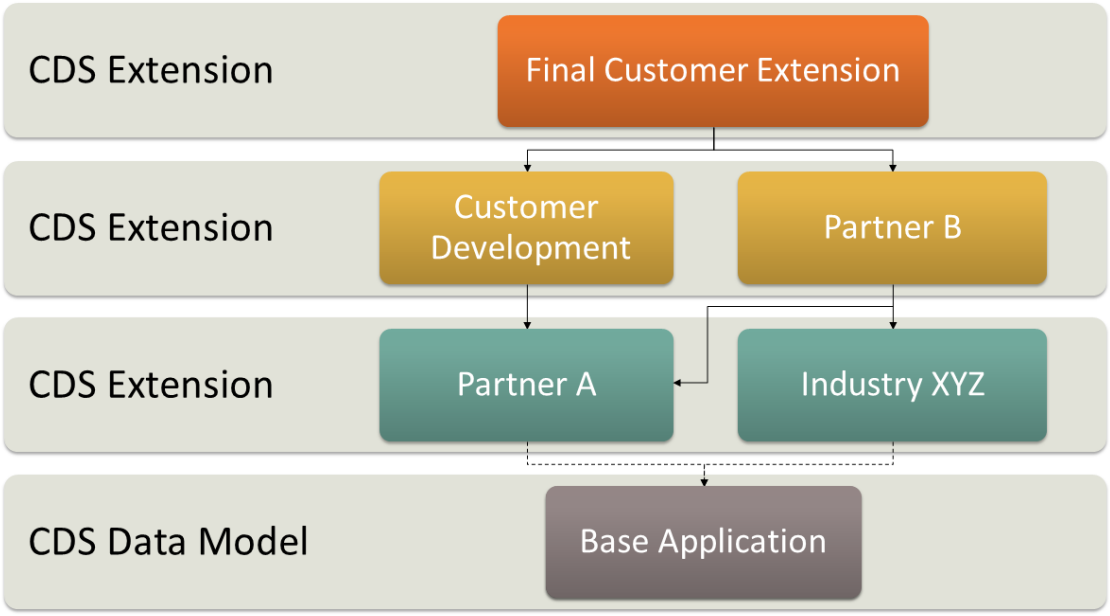
The following simple code example illustrates the syntax required for defining a CDS extension package and its dependencies:

> ⥱ Sample Code
>
> CDS Extension Package Syntax

```
source = packageDefinition
packageDefinition = "package" packageName (
  "depends" "on" packageName ( "," packageName )* )? ";"
packageName = identifier ( "." identifier )*
```

The name of the package defined in the file must be identical to the name space that is applicable for the file (as specified in the relevant HDI container-configuration file (`.hdinamespace`).

> → Tip
>
> The base package is not explicitly defined; it contains all CDS sources that are not explicitly assigned to a package.



To define a package hierarchy according to the diagram above, the following package definition files need to be provided (the names are just an example and do not confirm to any recommended naming convention):

Package Hierarchy and Definition Files

Package Directory (Namespace)	Content of Package Definition File (.package.hdbcds)
`myapp.partners.A`	`package myapp.partners.A;`
`myapp.industries.XYZ`	`package myapp.industries.XYZ;`
`myapp.customDev`	`package myapp.customDev depends on myapp.partners.A;`
`myapp.partners.B`	`package myapp.partners.B depends on myapp.partners.A, myapp.industries.XYZ;`
`myapp.customer`	`package myapp.customer depends on myapp.customDev, myapp.partners.B;`

Package and Source-File Assignment

To ensure that all the definitions in the CDS source, both normal artifact definitions and extend statements, belong to the respective package, you must assign a CDS source file to an extension package. To assign a CDS

source to an extension package, add the statement `in package` to the beginning of the CDS source file, as illustrated in the following example:

Sample Code

CDS Extension Package Assignment

```
in package <packageName>;
```

Note

The `in package` statement must be placed at the beginning of the CDS document, for example, before (or after) the name-space declaration, if present, but in all cases **before** all other statements in the CDS document.

Related Information

Create a CDS Extension [page 423]
The CDS Extension Descriptor Syntax [page 432]
The CDS Extension Package Descriptor [page 436]

5.3.4.7.2 The CDS Extension Descriptor Syntax

The syntax required to define a CDS extension artifact.

The CDS extension mechanism allows you to add properties to existing artifact definitions without modifying the original source files. The content extension of the CDS extension descriptor must conform to the following format:

Note

The following example of a CDS document for XS advanced is incomplete; it is intended for illustration purposes only.

Sample Code

CDS Artifact Extension Syntax

```
namespace banking;
in package banking;
using CRM;
using CRM.Customer;
extend context CRM with {
  type BankingAccount {
    BIC : String(8);
    IBAN : String(120);
  };
};
extend entity Customer with {
  account: CRM.BankingAccount;
```

```
};
extend type Address with {
  email : String(60);
};
extend view MyView with {
  a,
  b as newB,
  ass[y=2].x as elemViaAssoc,
  sum(t) as aNewAggregate
};
```

! Restriction

If a CDS artifact is defined in a package, it cannot be extended in the same package. In addition, the same CDS artifact cannot be extended twice in the **same** package.

in package

Use the keywords "`in package`" to assign a CDS source document to a CDS extension package. This ensures that all the definitions in the CDS source files, both normal artifact definitions and extend statements, belong to the named package.

Sample Code

CDS Artifact Extension Syntax

```
namespace banking;
in package banking;
```

i Note

The `in package` keyword must be inserted at the beginning of the CDS document: before or after the namespace declaration, if present, but always before all other statements.

using

All artifacts that are to be extended in a CDS source document must be made available with the `using` declaration.

Sample Code

```
namespace banking;
in package banking;
using CRM;
using CRM.Customer;
```

Extending Elements in CDS Entities or Structured Types

You can use a CDS extension to add new elements to an existing CDS entity, as illustrated in the following example:

Sample Code

Extend a CDS Entity with New Elements

```
extend entity MyEntity with {
  name1 : type1;
  name2 : type2;
  ...
};
```

The SAP HANA table generated for the CDS entity contains all specified extension elements. New elements can also be added to an existing `structured type`. It does not matter whether the original type is defined by means of the keyword "`type`" or "`table type`". The following example shows how to extend a table type:

Sample Code

Extend a CDS Structured Type

```
extend type MyType with {
  name1 : type1;
  name2 : type2;
  ...
};
```

Note

For a "`table type`", the generated SAP HANA table type contains all extension elements. This kind of extension does not work for scalar types.

Extending SELECT Items in a CDS View

You can use a CDS extension to add new `SELECT` items to an existing CDS view, as illustrated in the following example:

Sample Code

Extend a CDS View

```
extend view MyView with {
  a,
  b as newB,
  ass[y=2].x as elemViaAssoc,
  sum(t) as aNewAggregate
};
```

The SAP HANA view generated for the CDS view contains all extension items added to the `SELECT` clause.

! Restriction

It is not possible to extend any part of a view definition other than the `SELECT` clause.

Adding Artifacts to a CDS Context

You can use a CDS extension to add new artifacts (for example, tables, types, or views) to an existing CDS context, as illustrated in the following example:

‹≡› Sample Code

Extend CDS Context with New Artifacts

```
extend context MyContext with {
  type T1 : Integer;
  type S1 {
    a : INteger;
    b : String;
  };
  entity E1 {
    elem1 : Integer;
    elem2 : S1;
  };
  view V1 as select from E1 {, elem1, elem2 };
};
```

Extending CDS Annotations

You can use a CDS extension to add new `@annotations` to an existing CDS artifact or element; the syntax you use to add the annotations differs according to whether you are adding them to a CDS artifact or an element, as illustrated in the following examples:

‹≡› Sample Code

Extending CDS Artifacts with Annotations

```
@MyIntegerAnnotation : 44
extend entity MyEntity;
```

‹≡› Sample Code

Extending CDS Elements with Annotations

```
extend entity MyEntity with {
  @MyIntegerAnnotation : 45
  extend baseElement;
};
```

Sample Code

Extending CDS Entities with Annotated Elements

```
extend entity MyEntity with {
  @MyIntegerAnnotation : 45
  newElement : String(88);
};
```

→ Tip

Adding associations to elements of structured types and to `SELECT` items in views works in the same way.

Extending a CDS Entity's Technical Configuration

You can use a CDS extension to add new elements to the `technical configuration` section of an existing CDS entity, as illustrated in the following example:

Sample Code

Extending the CDS Entity's Technical Configuration

```
extend entity MyEntity with technical configuration {
  partition by hash (baseElement) partitions 2;
};
```

→ Tip

You can use the same method to extend a CDS entity with additional indexes.

Related Information

The CDS Extension Descriptor [page 429]
Create a CDS Extension [page 423]
The CDS Extension Package Descriptor [page 436]
The CDS Extension Package Descriptor Syntax [page 437]

5.3.4.7.3 The CDS Extension Package Descriptor

In the context of a CDS extension scenario, it is necessary to specify which `extend` statements and normal CDS artifact definitions belong to which package and, in addition, on which other packages a CDS extension package depends.

A CDS extension package is a normal CDS artifact; it has a name, and both its definition and its use can be found in the CDS Catalog. A package is defined by a special CDS source file named `.package.hdbcds`. The syntax for defining a package and its dependencies is illustrated in the following example:

Sample Code

```
package <onlineBanking> depends on <banking>;
```

Related Information

The CDS Extension Package Descriptor Syntax [page 437]
Create a CDS Extension [page 423]

5.3.4.7.4 The CDS Extension Package Descriptor Syntax

Required syntax for the CDS extension descriptor.

A CDS extension package is defined in a CDS extension package descriptor, which is a special CDS source file named `.package.hdbcds`, as illustrated in the following example:

Note

The leading dot (.) in the file name for the CDS extension package descriptor is mandatory.

Sample Code

Extension Package Descriptor (`.package.hdbcds`)

```
package <onlineBanking> depends on <banking>;
```

In this example, the package `<onlineBanking>` depends on another CDS extension package `<banking>`, which contains extensions for a CDS base application `<myCDSapp>`.

The syntax for defining a package and its dependencies is illustrated in the following example:

Sample Code

CDS Extension **Package** Descriptor Syntax

```
source = packageDefinition
packageDefinition = "package" packageName (
    "depends" "on" packageName ( "," packageName )* )? ";"
packageName = identifier ( "." identifier )*
```

The name of the package defined in the file must be identical to the name space that is applicable for the file (as specified in the application's corresponding `.hdinamespace` file).

> **Note**
>
> It is not necessary to explicitly define the base package for the CDS application; it contains all those CDS sources that are not explicitly assigned to a package.

packageDefinition

The syntax for defining a dependency between CDS extension packages is illustrated in the following example:

> **Sample Code**
>
> One-to-One Package Definition (`.package.hdbcds`)

```
package <onlineBanking> depends on <banking>;
```

A package can depend on multiple packages, as illustrated in the following example:

> **Sample Code**
>
> One-to-Many Package Definition (`.package.hdbcds`)

```
package <onlineBanking> depends on (<banking>, <banking1>, <[...]>);
```

The following example illustrates the syntax required in a package-dependency statement specified in the CDS extension package descriptor (`.package.hdbcds`)

> **Sample Code**
>
> CDS Extension **Package** Descriptor Syntax

```
source = packageDefinition
packageDefinition = "package" packageName (
    "depends" "on" packageName ( "," packageName )* )? ";"

packageName = identifier ( "." identifier )*
```

packageName

The syntax required when specifying the name of a CDS extension package in a package-dependency definition is illustrated in the following example:

> **Sample Code**
>
> One-to-One Package Definition (`.package.hdbcds`)

```
package <onlineBanking> depends on <banking>;
```

A package name can include a full stop (.), for example, to express a full path in a name space, as illustrated in the following example:

Sample Code

One-to-Many Package Definition (`.package.hdbcds`)

```
package <src.onlineBanking> depends on (<src.banking>);
```

The following example illustrates the syntax required when defining the name of a package in the CDS extension package descriptor (`.package.hdbcds`)

Sample Code

CDS Extension **Package** Descriptor Syntax

```
source = packageDefinition
packageDefinition = "package" packageName (
    "depends" "on" packageName ( "," packageName )* )? ";"
packageName = identifier ( "." identifier )*
```

Related Information

Create a CDS Extension [page 423]
The CDS Extension Descriptor [page 429]
The CDS Extension Descriptor Syntax [page 432]
The CDS Extension Package Descriptor [page 436]

5.3.4.8 Create a CDS Access-Policy Document (XS Advanced)

As an XS advanced developer, you want to set instance-based authorizations for accessing data in the SAP HANA database.

Context

CDS access-policy documents are coded in the Data Control Language (DCL). CDS access policies have the same file extension as CDS documents. In a CDS access policy document, you can create CDS roles and CDS aspects for instance-based authorizations. It is also possible to use `grant` statements on CDS views and CDS roles and include the statements in the definition of the CDS roles. The `grant` statements determine which data set a user is authorized to access.

Procedure

1. Start the SAP HANA Web IDE for SAP HANA.

   The SAP Web IDE for SAP HANA is available at the following default URL:

   `https://<HANA_HOST>:53075/`

   > → Tip
   >
   > If the SAP Web IDE for SAP HANA is running on a non-default port, open a command shell, log on to the XS advanced run time, and run the following command:
   >
   > `xs app webide --urls`

2. Open the application project to which you want to add your CDS access policy.
3. Open your multi-target application project, or if required, create a new project.
4. Define a new CDS access-policy document.

   A CDS access-policy document is a file with a CDS file extension (`.hdbcds`). Enter a file name, for example `MyAccessPolicy` and choose *Finish*.

   Define the CDS role in your CDS access policy. Save the CDS access policy file to your local project workspace.

   > ⚠ Caution
   >
   > The code examples included in this document for XS advanced are sometimes syntactically incomplete; as a general rule, code examples are intended for illustration purposes only.

   > Sample Code

   ```
   namespace com.sap.dcl.example;
   using com.sap.dcl.example::<DDL_file_name> as ddl;

   AccessPolicy MyAccessPolicy {

     role salesOrderCountryUsa {
       grant select on ddl.salesOrderView
           where customerCountry = 'USA';
     };
   }
   ```

5. Check the syntax of CDS sources.

   Check the syntax of CDS access policies in the same way as you would check the syntax of CDS DDL sources.

6. Activate your CDS access-policy document in the same way as you would activate DDL sources.

   After the activation of the CDS DCL source files (CDS access-policy document and source file with DDL code), the DCL artifacts you defined previously (for example, roles) are added to the SAP HANA database catalog.

   - You can find the generated database role in the *Security* section of your SAP HANA database. In this example, the naming convention of the roles is as follows:
     `<project_path.project>::<DCL_source_file>.<role_name>`

```
com.sap.dcl.example::dcl.salesOrderCounryUsa
```

Results

The database roles are also available in the SAP HANA Web IDE for SAP HANA.

> **i Note**
>
> This kind of database roles need to be assigned to the technical database user of the XS advanced application. To assign database roles to users, the XS advanced developer needs the HDI container's `default_access_role`. For more information about the default access role, see *Related Links*.

Related Information

Create a CDS Document (XS Advanced) [page 313]
The Default Access Role for HDI Containers [page 268]
Configure a Database Connection [page 762]

5.3.4.8.1 CDS Access Policies in XS Advanced

As a developer in XS advanced, you can set instance-based authorizations for accessing data in the SAP HANA database. You define access policies using the CDS data control language (DCL). Write CDS access policy documents using DCL code where you define CDS roles. Instance-based authorizations are based on CDS views.

Prerequisites

An XS advanced developer must have the standard developer authorization profile to assign roles to SAP HANA users.

> **⚠ Caution**
>
> The code examples included in this document for XS advanced are sometimes syntactically incomplete; as a general rule, code examples are intended for illustration purposes only.

CDS Documents for Access Policies Defined with Data Control Language

If you want to create access policies using DCL, you must define the policies in CDS documents. CDS documents are CDS source files with the suffix `.hdbcds` which are located in the `db` folder of the HDI container.

You use DCL code to define access-control logic for Core Data Services (CDS) views from the SAP HANA database. Definitions in DCL code enable you to filter access to data in the database based on static values, aspects, and based on external attributes.

⚠ Caution

If you do **not** create and deploy CDS access policies for the CDS view, any user who can access the CDS view has access to all data returned by the CDS view.

CDS documents with DCL code or DDL code only differ in their top level element:

- DCL documents have `AccessPolicy` as top level element (whereas CDS documents with DDL code usually have `Context` as top level element).
- You can define CDS aspects and CDS roles within an `AccessPolicy` keyword section, as illustrated in the examples below.

When XS advanced developers compile an access-policy definition using a CDS document with DCL code, they generate corresponding database catalog artifacts, for example, roles.

Sample Code

```
namespace com.sap.dcl.example;
using com.sap.dcl.example::<DDL_file_name> as ddl;

AccessPolicy <DCL_file_name> {
  role salesOrderCountryUsa {
    grant select on ddl.salesOrderView
        where customerCountry = 'USA';
  };
}
```

In the CDS source file, the XS advanced developer must explicitly specify the structured privilege check for a CDS view, as illustrated in the following example.

! Restriction

In XS advanced, the `@schema` annotation cannot be used in either CDS documents with DCL code or CDS documents with DDL code.

Source Code

```
entity salesOrder {
    key id                : Integer;
    toCustomer            : Association[0..1] to businessPartner;
    currencyCode          : String(5);
    grossAmount           : Integer;
    document              : LargeBinary;
};

  view salesOrderView as select from salesOrder
  {
    key id as id,
    toCustomer.toAddress.country as customerCountry,
    currencyCode,
    grossAmount,
    document
  } with structured privilege check;
```

→ Tip

In XS advanced, it is recommended to use `key` fields in CDS views when creating instance-based authorization checks.

The keys in CDS views should be defined in a similar way to the key definition for DDL entities, for the following reasons:

- Enhanced system performance
- The field types `LargeBinary` and `LargeString` must only be used in CDS views with key fields so that instance-based authorizations are possible.

ⓘ Note

Only CDS **views** can be protected by a `grant` statement; CDS entities are not supported. You can use DCL for calculation views, but it is mandatory to include the calculation view in the CDS access policy, for example, by including a `using` statement at the start of the document. Only calculation views generated outside of CDS can be protected by a `grant` statement. A definition of calculation views is not supported by DDL.

Related Information

Create a CDS Access-Policy Document (XS Advanced) [page 439]
Create a CDS Aspect (XS Advanced) [page 443]
Create a CDS Role (XS Advanced) [page 445]

5.3.4.9 Create a CDS Aspect (XS Advanced)

Use CDS aspects to create a CDS role that references external dynamic criteria.

Context

CDS aspects associate an attribute with permitted values of a user. The values are taken, for example, from a CDS entity or a CDS view, which is defined in a CDS DDL document. A `grant` statement in a role can use one or more “aspects” in a `where` clause.

⚠ Caution

The code examples included in this document for XS advanced are sometimes syntactically incomplete; as a general rule, code examples are intended for illustration purposes only.

Procedure

1. Start the SAP HANA Web IDE for SAP HANA.

   The SAP Web IDE for SAP HANA is available at the following default URL:

   `https://<HANA_HOST>:53075/`

   > → Tip
   >
   > If the SAP Web IDE for SAP HANA is running on a non-default port, open a command shell, log on to the XS advanced run time, and run the following command:
   >
   > `xs app webide --urls`

2. Open the application project to which you want to add your CDS aspect.
3. Open your multi-target application project, or if required, create a new project.
4. Open the CDS access-policy document.

   Define the CDS aspects in your CDS access-policy document. Save the DCL definition of your CDS aspects to your local project workspace.

   In this example, you want to enable users to access only the sales orders of their own country. For this, you need the country information so that you can use it in the respective role. You define a CDS aspect that associates `country` from the `address` CDS entity with the corresponding employee's login name. This CDS aspect is then used by the corresponding CDS role.

   > Sample Code

   ```
   namespace com.sap.dcl.example;
   using com.sap.dcl.example::<DDL_file_name> as ddl;

   AccessPolicy <DCL_file_name> {

     aspect aspCountry as
       select from ddl.address { country }
         where $user in toEmployee.loginName;
     role salesOrderCountryOwn {
       // grant based on an aspect
       grant select on ddl.salesOrderView
           where customerCountry = aspect :aspCountry;
     };
   }
   ```

5. Check the syntax of CDS sources.

   Check the syntax of CDS access policies in the same way as you would check the syntax of CDS DDL sources.

6. Activate your CDS access policy document.

Related Information

Create a CDS Document (XS Advanced) [page 313]
Create a CDS Role (XS Advanced) [page 445]

5.3.4.10 Create a CDS Role (XS Advanced)

Set instance-based authorizations for accessing data in the SAP HANA database.

Context

CDS access-policy documents contain a set of CDS role definitions coded in the Data Control Language (DCL). You can use CDS roles to create instance-based authorizations. A role can contain CDS “aspects”, and it is also possible to use `grant` statements on CDS views and CDS roles and include the statements in the definition of the CDS roles. The database roles generated from the CDS role definition determine which data sets a user is authorized to access.

> ⚠ Caution
>
> The code examples included in this document for XS advanced are sometimes syntactically incomplete; as a general rule, code examples are intended for illustration purposes only.

> ✤ Example
>
> If you provide authorization to access sales orders, you can add conditions that filter the view users have on the sales orders, for example, as follows:
>
> - View the sales orders of all countries.
> - View only the sales orders of one specific country, for example: the USA. The CDS role in the CDS access policy document contains a fixed value for “USA”.
> - View only the sales orders of one specific country. The CDS role in the CDS access policy document contains a CDS aspect associated with the country value of the CDS entity called `address`.
> - View only the sales order of his or her country. Here, external user information in XS advanced provides the country attribute that is used in the CDS DCL definition.

Procedure

1. Start the SAP HANA Web IDE for SAP HANA.

   The SAP Web IDE for SAP HANA is available by default at the following default URL:

   ```
   https://<HANA_HOST>:53075/
   ```

   > → Tip
   >
   > If the SAP Web IDE for SAP HANA is running on a non-default port, open a command shell, log on to the XS advanced run time, and run the following command:
   >
   > ```
   > xs app webide --urls
   > ```

2. Open the application project to which you want to add your CDS role.

3. Open your multi-target application project, or if required, create a new project.
4. Open the CDS access policy document.

   Define the CDS roles in your CDS access policy. Save the DCL definition of your CDS roles to your local project workspace.

   Sample Code

```
namespace com.sap.dcl.example;
using com.sap.dcl.example::<DDL_file_name> as ddl;

AccessPolicy <DCL_file_name> {
  role salesOrderCountryAll {
    // grant all countries
    grant select on ddl.salesOrderView;
  };
  aspect aspCountry as
    select from ddl.address { country }
      where $user in toEmployee.loginName;

  role salesOrderCountryUsa {
    // grant of a static value
    grant select on ddl.salesOrderView
        where customerCountry = 'USA';
  };
  role salesOrderCountryOwnAndUsa {
    // grant based on an aspect
    grant select on ddl.salesOrderView
        where customerCountry = aspect :aspCountry;
    // grant based on an external attribute
    grant select on ddl.salesOrderView
        where customerCountry in $env.user.country;

    // grant of another role
    grant salesOrderCountryUsa;
  };
};
```

5. Check the syntax of CDS sources.

   Check the syntax of the CDS roles in the same way as you would check the syntax of CDS DDL sources.
6. Activate your CDS access policy document in the same way as you would activate DDL sources.

   After the activation of the CDS access policy document with the CDS roles, the DCL artifacts you defined are added to the SAP HANA database catalog.

   - After activation, the generated database roles are located in the *Security* section of your SAP HANA database. In this example, the naming convention of the roles is as follows:
     `<project_path>.<project>::<DCL_source_file>.<role_name>`
     `com.sap.dcl.example::dcl.salesOrderCountryAll`
     `com.sap.dcl.example::dcl.salesOrderCountryUsa`
     `com.sap.dcl.example::dcl.salesOrderCountryOwnAndUsa`

Results

The database roles are also available in the SAP HANA Web IDE for SAP HANA.

> **i Note**
>
> Roles are assigned to the technical database user of the XS advanced application. To assign roles to users, you need the `default_access_role` for the HDI container. For information, see *Related Links*).

Related Information

Create a CDS Document (XS Advanced) [page 313]
The Default Access Role for HDI Containers [page 268]

5.3.4.10.1 CDS Role Syntax Options

The options available for modeling instance-based authorizations in CDS roles.

Overview

It is possible to combine `grant` statements in a role definition or use `grant` statements of multiple types. The following list shows the different ways in which `grant` statements can be used to define a CDS role:

- Using a grant on a CDS view with static values
- Using a grant on CDS aspects that associate a dynamic element of a CDS view (in a CDS DDL document)
- Using a grant on external attributes
- Using a grant of another role

CDS aspects can also be included in CDS role definitions and can be referenced in grants by using the following syntax:

```
aspect :<role_name>.<aspect_name>
```

The database roles generated from the CDS role definition determine which data set a user is authorized to access. If multiple roles are assigned to a user, the specified user has the authorizations defined in all of the assigned roles.

> **⚠ Caution**
>
> The code examples included in this document for XS advanced are sometimes syntactically incomplete; as a general rule, code examples are intended for illustration purposes only.

CDS Role Using Grant with Static Values

Sample Code

```
grant select on ddl.salesOrderView
    where customerCountry = 'USA';
```

In this CDS role, you use a `where` clause to directly refer to the `customerCountry` element defined in the CDS view called `salesOrderView`. The grant only allows a view on sales orders with the `customerCountry` value USA.

CDS Role Using Grant with CDS Aspects

Sample Code

```
grant select on ddl.salesOrderView
    where customerCountry = aspect :aspCountry;
```

In this CDS role, you use the previously defined CDS aspect `aspCountry`. It uses the country element of the address information that comes with the CDS entity called `address`. Since users have different addresses, this data is variable and dynamic. The `customerCountry` element defined in the CDS view called `salesOrderView` is associated with the CDS aspect `aspCountry`. This CDS role definition allows users to view the sales orders of the `customerCountry` value provided by their address, that is; the sales orders of their own country.

CDS Role Using Grant with External Attributes

Sample Code

```
grant select on ddl.salesOrderView
    where customerCountry in $env.user.country;
```

In this CDS role, you directly use external attributes that come with the user data of the XS advanced token for the database. The `customerCountry` element defined in the CDS view called `salesOrderView` is associated with the country attribute of the user's login data. Access is enabled to the external attribute using `$env.user.<external_attribute>`. This grant allows users to view the sales orders of the `customerCountry` value provided by the XS advanced token.

Note

It is also possible to use XS advanced attributes that have multiple values.

CDS Role Using Grant of Another Role

Sample Code

```
grant salesOrderCountryUsa;
```

This grant includes the already defined CDS role `salesOrderCountryUsa`. Here, the CDS role `salesOrderCountryUsa` is already defined, for example, in the same CDS access-policy document. It is also possible to define CDS roles in separate CDS DCL files. This grant with an included CDS role allows users to also view the sales orders of the USA.

Note

You can also use a `grant` of native SAP HANA roles, for example, defined in design-time artifacts with the file extension `.hdbrole`. However, you must introduce the native SAP HANA role by including a `using` declaration first. Only then is it possible to integrate the native role using the `grant` statement.

Related Information

The Application Security Descriptor [page 996]

5.4 Maintaining JSON Collections in the SAP HANA Document Store

The SAP HANA Document Store (DocStore) is used to store collections which contain one or more JSON artifacts (documents).

The SAP HANA DocStore is a place where you can collect JSON documents, that is; files with content that is formatted according to the rules defined in the JavaScript Object Notation. The DocStore allows native operations on JSON documents, for example: filtering and aggregation, as well as joins with SAP HANA column- or row-store tables.

You can use standard SQL commands in an SQL console to maintain the document collections in the SAP HANA DocStore and, in addition, the JSON documents that make up the collection, and any values defined in the individual JSON documents.

In this section you can find information about creating and maintaining JSON documents in collections stored in the SAP HANA DocStore, for example:

- Create a SAP HANA Documentation Store.
- Add collections to (and remove them from) the DocStore.
- Maintain JSON documents in DocStore collections.

Related Information

Create a JSON Document Store [page 450]
Add and Remove JSON Collections in the Document Store [page 452]

5.4.1 Create a JSON Document Store

Set up a store in SAP HANA for your JSON documents.

Context

The SAP HANA Document Store (DocStore) is a place where you can collect JSON documents, that is; text files formatted according to the rules defined in the JavaScript Object Notation. The SAP HANA DocStore allows native operations on JSON documents, for example: filtering and aggregation, as well as joins with SAP HANA column- or row-store tables.

Procedure

1. Check that the DocStore is enabled.

   For multi-database instances, there can be 0 (none) or 1 (one) DocStore per tenant database; you can enable the DocStore in a database tenant by running the following command as administrator in the SQL console:

   ```
   ALTER DATABASE <tenant_database> ADD 'docstore'
   ```

   > **i** Note
   >
   > Enabling the JSON Document Store for a tenant starts an instance of the "`docstore`" service for the tenant. A tenant must not have more than one `docstore` process running.

2. Create a JSON collection in the DocStore using SQL.

   Open an SQL console and enter the following SQL command:

   ```
   create collection Customers;
   ```

3. Create a JSON collection using a design-time artifact.

   You can define a JSON collection in a design-time artifact with the suffix "`hdbcollection`", for example, `AUTHORS.hdbcollection`. The file format uses a DDL-style syntax which is equivalent to the corresponding `CREATE COLLECTION` SQL syntax, but without the leading `CREATE` command, as illustrated in the following example:

Code Syntax

`/src/AUTHORS.hdbcollection`

```
COLLECTION "AUTHORS"
```

The `hdbcollection` artifact must be located in the `db/` module of your XS advanced application. Assuming that the corresponding HDI plug-in is available, the collection defined in the design-time artifact is activated in the catalog at application-deployment time.

Note

If the DocStore is not enabled and available at deployment time, the deployment operation fails and no collection object is created in the database catalog.

Related Information

Maintaining JSON Collections in the SAP HANA Document Store [page 449]
Add and Remove JSON Collections in the Document Store [page 452]
Document Store Collections (.hdbcollection)

5.4.1.1 The SAP HANA JSON Document Store

The SAP HANA Document Store (DocStore) contains JSON artifacts grouped in collections.

The SAP HANA DocStore is a feature that you can use to store JSON artifacts. The DocStore allows you to perform all standard operations on the JSON documents, for example, filtering, aggregation, and joining the JSON documents with HANA column or row store tables.

The Document Store is an optional feature of the SAP HANA database which you have to create for each tenant database. Only a single DocStore process per tenant is possible. To create the DocStore you can use the `ALTER DATABASE` statement, as illustrated in the following example:

```
ALTER DATABASE <database name> ADD 'docstore'
```

For a single database container, it is necessary to enable the DocStore by setting the `docstore` configuration parameter, as illustrated in the following SQL example:

```
ALTER SYSTEM ALTER CONFIGURATION('daemon.ini', 'SYSTEM')
SET ('docstore', 'instances') = '1'
WITH RECONFIGURE;
```

Restriction

Only a single DocStore per instance is allowed.

The DocStore does not have a predetermined SQL port; all communication is routed through the standard index servers. For a multi-database installation, port numbers are assigned according to the rules of multi-database installations. For single database containers port 3<instanceNr>60 is used as default.

Related Information

Create a JSON Document Store [page 450]
Maintaining JSON Collections in the SAP HANA Document Store [page 449]
JSON Collections [page 454]

5.4.2 Add and Remove JSON Collections in the Document Store

Maintain new and existing collections of JSON documents in the SAP HANA DocStore.

Context

You can use standard SQL commands in an SQL console to maintain the document collections in the SAP HANA DocStore and, in addition, the JSON documents that make up the collection. For example, you can add, update, and remove JSON documents in a collection. You can also maintain the values defined in the individual JSON documents.

Procedure

1. Check that the DocStore is enabled.

   For multi-database instances, there can be 0 (none) or 1 (one) DocStore per tenant database; you can enable the DocStore in a database tenant by running the following command as administrator in the SQL console:

   ```
   ALTER DATABASE <database> ADD 'docstore'
   ```

2. Create a JSON collection in the DocStore.

   To create a collection of JSON documents called “CUSTOMERS” run the following command as administrator in an SQL console:

   ```
   CREATE COLLECTION CUSTOMERS;
   ```

3. Insert a JSON document into the collection.

Insert JSON-formatted information into the collection, as illustrated in the following example:

Sample Code

```
insert into Customers values({
    "firstName": 'Kofi',
    "familyName": 'Tetteh',
    "address": {
        "street": 'Nkrumah Close',
        "number": '10',
        "city": 'Accra'
        },
    "details": {
        "age": 43
        }
    });
```

Sample Code

```
insert into Customers values({
    "firstName": 'Kwame',
    "familyName": 'Asumang',
    "address": {
        "street": 'Independence Avenue',
        "number": '147',
        "city": 'Accra'
        },
    "details": {
        "age": 27
        }
    });
```

4. Run a query on the content of a JSON document in the DocStore.

   Search the JSON collection for information about specific addresses, as illustrated in the following example:

Sample Code

```
select * from Customers where "address"."city" = 'Accra';
```

Sample Code

```
select * from Customers where "details"."age" = 40;
```

5. Update individual JSON documents in a collection.

   To update a JSON documents stored in the JSON collection “CUSTOMERS”, use the `UPDATE` command as follows, which specifies what values to change (`SET`) in the JSON documents, where (`Customers`), and on what basis (age):

Sample Code

```
UPDATE Customers SET Customers = { "name": 'Kojo', "age": 50 } where
"details"."age" > 30;
```

The result of this change request in the JSON collection is as follows:

Sample Code

```
{ "firstName": "Kojo", "age": 50 }
{ "firstName": "Kwame", "familyName": "Asumang", "address": { "street":
"Independence Avenue", "number": "147", "city": "Accra" }, "details":
{ "age": 27 } }
```

6. Remove JSON documents from a collection.

   To remove JSON documents stored in the collection "CUSTOMERS", use the `DELETE FROM` command as follows, where all documents that match the condition defined in the `WHERE` clause are deleted:

   Sample Code

   ```
   DELETE FROM CUSTOMERS WHERE <condition>;
   ```

   To remove all documents from a JSON collection, use the following command, in which `WHERE true` is implied:

   Sample Code

   ```
   DELETE FROM CUSTOMERS;
   ```

7. Rename a JSON document collection.

   To rename an existing JSON collection, use the following command:

   Sample Code

   ```
   RENAME COLLECTION <collection_name> TO <new_collection_name>;
   ```

Related Information

Create a JSON Document Store [page 450]
Maintaining JSON Collections in the SAP HANA Document Store [page 449]

5.4.2.1 JSON Collections

JSON documents are grouped together as "collections " that are stored in the SAP HANA Documentation Store (DocStore).

Data defined in documents formatted according to the JavaScript Object Notation (JSON) is stored in JSON artifacts that can be grouped into so-called "collections" and stored in the SAP HANA DocStore. The content of a JSON document may be deeply structured, however, unlike XML, a JSON document does not have a schema. This means that any valid JSON data may be inserted without first declaring its structure.

Document collections and the JSON content can be maintained with SQL in the same way as data stored in tables. For example, you can insert data with the `INSERT` statement and query the document data with the `SELECT` command. You can also use a single query statement to read data from tables and collections at the same time, and you can combine standard tables and JSON collections by joining them in the same way as the `JOIN` between any other column- or row-store tables.

Technically, collections are stored in a dedicated binary format; in SQL however, they are defined as tables with their own sub-type, which you can see by selecting collections on the basis of the table type `'COLLECTION'`, as illustrated in the following example:

```
SELECT * FROM tables WHERE table_type = 'COLLECTION'
```

The following example is included here to illustrate how a collection of customer details can be created, updated and read. The `INSERT` statement illustrates the syntax and structure of nested JSON data and the corresponding `SELECT` statements show the use of path expressions to navigate the content of the JSON document:

```
create collection Customers;
```

```
insert into Customers values({
    "name": 'Paul',
        "address": {
        "street": 'Main Street 10',
        "city": 'Heidelberg'
        }
    });
```

```
select * from Customers where "address"."city" = 'Heidelberg';
```

```
select "address"."city", count(*) from Customers group by "address"."city";
```

JSON Documents

The content of a JSON document may be deeply structured, however, unlike XML, a JSON document does not have a schema. This means that any valid JSON data may be inserted without first declaring its structure.

Sample Code

JSON Nested Content

```
{
  "author": {
      "address": {
          "street": "Main Street"
          "number": 101
          },
          ...
      },
      ...
  }
```

To access the information in the nested structure defined in the JSON document, you can use the “dot notation” to define the path to the required value, for example: `"author"."address"."street"`

In a JSON document, it is also possible to define elements in an array, such as the one illustrated in `"addresses": [...]` in the following simple example:

Sample Code

JSON Nested Content

```
{
  "author": {
      "addresses": [{
          "street": "First Street"
          },{
          "street": "Second Street"
          } ]
      },
      ...
  }
```

To enable access to elements defined in an array, a standard index operator `[#]` is required. The index positions start at "1", which means the first element has the position 1. So, to access the value defined in the **second** address in the example above (`"Second Street"`), you would use something like the following example: `"author"."addresses"[2]."street"`

→ Tip

This notation can be used anywhere where it is necessary to access a value, for example, "Sort", "Filter", "Projection". If no value exists in the document for a given identifier a NULL value is returned instead.

Related Information

Document Store Collections (.hdbcollection)
The SAP HANA JSON Document Store [page 451]
Add and Remove JSON Collections in the Document Store [page 452]

5.4.3 Import JSON Documents into a DocStore Collection

Import JSON data from a CSV file into a collection in the SAP HANA DocStore.

Prerequisites

The success of the import operation depends on the following prerequisites:

- The SAP HANA DocStore is enabled
- The target document collection specified in the data-import definition is available in the DocStore, for example, `CUSTOMERS`.

→ Tip

You can define the collection in the SQL console or in an HDI design-time artifact, for example, `CUSTOMERS.hdbcollection`.

Context

You can use the HDI Table Data plug-in (`hdbtabledata`) to import one or more JSON documents stored in a CSV file; multiple JSON documents are separated by a line-break character, for example, `\n`.

⚠ Caution

Although JSON documents do not have a schema, it is not recommended to use different data types for the same identifier in different JSON documents. This could cause inconsistent results returned from queries.

Procedure

1. Create the source file containing the data to import into your collection in the SAP HANA DocStore.

   The source data (JSON documents) must be stored in a comma-separated values (CSV) file, which must be placed in your XS advanced application's database module. Since the CSV file must only contain **one** column, no comma is required, as there is only one value. In the CSV file, each line represents a single JSON document, and the line-break character is used to define the end of one JSON document and the start of the next, as illustrated in the following example:

   → Tip

   The line-break is automatically added by your text editor; it is not necessary to add the line-break character to the CSV file manually.

   ‹≡› Sample Code

   `sap::myJSONData.csv`

   ```
   {"name": 'Peter', "address": {"street": 'Main Street 10', "city":
   'Heidelberg'} } \n
   {"name": 'Kwame', "address": {"street": 'High Street 15', "city":
   'Accra'} } \n
   {"name": 'Nelson', "address": {"street": 'Low Street 101', "city":
   'Johannesburg'} }
   ```

2. Create the data-import definition file.

   In SAP HANA XS advanced, you can configure the import operation in a so-called Table Data definition - a design-time artifact with the hdbtabledata suffix, for example, `myJSONdata`. This file must also be located in the XS advanced application's database module.

   a. Specify the target column mapping for the data import.

Since the CSV file must only contain **one** column, the mapping should look as follows:

> ‘≡› Sample Code
>
> ```
> "column_mappings": { "MY_DocStore_COLLECTION": 1 },
> ```

b. Specify the type of table into which you want to import the JSON data.

   It is only possible to import JSON data into a "table" of type "COLLECTION". For this reason, in the table-data definition, use the `"is_collection_table"` parameter (default is `false`) to define the target table “type” as illustrated in the following example.

   > ‘≡› Sample Code
   >
   > ```
   > "is_collection_table": true,
   > ```

c. Specify details of the source data to be imported.

   You will need to specify the `"data_type":` (CSV) of the source file itself; the format of the data in the CSV file (`"dialect": "HANA_JSON"`); the name of the source CSV file (`"file_name":`), and indicate that the CSV file does not contain any header information (`"has_header": false`).

d. Lastly, it is mandatory to specify the name of the collection (`"target table":`) into which the JSON documents will be inserted.

The completed definition file for your JSON data import operation should look like the following example:

> ! Restriction
>
> It is not allowed to use multiple data sources to fill a collection table, for example, by using the `include_filter` parameter in the `"import_settings"` section of the table-data definition file.

It is only possible to import data into one, single collection from one, single `hdbtabledata` file, and the import operation always imports **all** the content included in the corresponding CSV file.

> ‘≡› Sample Code
>
> hdbtabledata File for JSON data import
>
> ```
> {
>   "format_version": 1,
>   "imports": [{
>      "column_mappings": {
>        "CUSTOMERS": 1
>      },
>      "is_collection_table": true,
>      "source_data": {
>        "data_type": "CSV",
>        "dialect": "HANA-JSON",
>        "file_name": "sap::myJSONcustomerData.csv",
>        "has_header": false
>      },
>      "target_table": "CUSTOMERS"
>   }]
> }
> ```

3. Start the import operation.

   To start the import of the JSON documents into the specified collection in the SAP HANA DocStore, build the XS advanced application's database module which contains the CSV and table-data definition artifacts.

For example, in the project navigation pane in SAP Web IDE for SAP HANA, right-click the *hdb* module and choose *Build* in the context menu.

> **i Note**
>
> Importing data into a collection table fails if the parameter `"is_collection_table" : false` is set (or not set at all).

Related Information

Document Store Collections (.hdbcollection)
Table Data (.hdbtabledata)
Maintaining JSON Collections in the SAP HANA Document Store [page 449]

5.5 Creating Procedures and Functions in XS Advanced

Database procedures and functions can be used to help manage the underlying data model.

As part of the process of defining the database persistence model for your application, you create database design-time artifacts such as tables and views, for example using Core Data Services (CDS). However, you can also create procedures and table or scalar functions, for example, using SQLScript; procedures can be used to insert data into (and remove data from) tables or views.

In XS advanced, each database artifact has a distinct file extension, for example, `myProcedure.hdbprocedure`, `mySynonym.hdbsynonym`, `myScalarFunction.hdbfunction`; the file extension determines which plug-in is used to generate the corresponding catalog objects.

> **→ Tip**
>
> Scalar and table functions have the same file extension `.hdbfunction`. The HDI uses the same plug-in to generate the catalog object.

For easier navigation and maintenance, it is recommended to place your procedures and functions in a separate folder in the application's database (`db/`) module, as illustrated in the following example:

> **Sample Code**

```
<MyAppName>
|- db/                              # Database deployment artifacts
|  |- package.json                  # Database details/dependencies
|  |- src/                          # Database artifacts: tables, views, etc.
|     |- .hdiconfig                 # HDI build plug-in configuration
|     |- .hdinamespace              # HDI run-time name-space configuration
|     |- data/                      # CDS table definition
|     |  |- myEntity.hdbcds         # CDS table definition
|     |  |- myDataType.hdbcds       # CDS type definition
|     |  |- myDoc.hdbcds            # CDS data model overview
|     |- procedures                 # Database procedures, functions, ...
|        |- myProc.hdbprocedure     # Database procedure
|        \- myFunc.hdbfunction      # Database function
```

```
|- web/
|  |- xs-app.json
|  \- resources/
|- js/
|  |- start.js
|  |- package.json
|  \- src/
|- security/
|  \- xs-security.json
\- mtad.yaml
```

Related Information

Create a Database Procedure in XS Advanced [page 470]
Create a Database Function in XS advanced [page 473]
SAP HDI Artifact Types and Build Plug-ins Reference
Creating Data Persistence Artifacts with CDS in XS Advanced [page 310]

5.5.1 Create the Underlying Data Artifacts for a Procedures

Create the data objects that you plan to use and modify with your functions and procedures.

Prerequisites

- You have access to a running SAP HANA system where the XS advanced run time is installed and configured
- You have access to the developer tools included in SAP Web IDE for SAP HANA
  If SAP Web IDE for SAP HANA is not available at the default URL `https://host.acme.com:53075`, log in to the XS advanced run time and use the command `xs app webide` to display the URL required to access the tool.

> ⚠ Caution
>
> The code examples included in this document for XS advanced are sometimes either syntactically incomplete or include additional definitions that you might not always need; as a general rule, code examples are intended for illustration purposes only.

Context

Database procedures and functions are used to automate actions that need to be regularly performed on underlying data objects, such as tables or views. In this task, we show you how to create some tables, views,

and data types that can then be used by the database procedures and functions you create in subsequent tasks. You also create a sequence to enable you to add items to a table in the correct order, and upload some initial data into the new tables you create.

Procedure

1. Start the SAP Web IDE for SAP HANA.
2. Display the application project to which you want to add a database procedure.

   An application is created within a context of a project. If you do not already have a project, there are a number of ways to create one, for example: by importing it, cloning it, or creating a new one from scratch.

   a. In the SAP Web IDE for SAP HANA, choose *File* > *New* > *Project from Template*.
   b. Choose the project template type.

      Currently, there is only one type of project template available, namely: *Multi-Target Application Project*. Select *Multi-Target Application Project* and choose *Next*.
   c. Type the name **`myworld`** for the new MTA project and choose *Next* to confirm.
   d. Specify details of the new MTA project and choose *Next* to confirm.
   e. Create the new MTA project; choose *Finish*.
3. Create a new application folder for your data artifacts.

   In the `db/` module of your application project, create a folder named `data` in the `src/` folder, for example, `myworld/db/src/data`. You use this folder to store your tables, types, views, sequences, and so on.
4. Create a CDS document to define your data model

   In this step we create a Core Data Services (CDS) document to define a couple of simple database tables: "header" and "item" for purchase orders. The procedure we define later is used to run queries on (and add content to) these tables.

   > → Tip
   >
   > If you do not already have a database module, right-click the root folder of your new MTA project and, in the context menu, choose *New* > *HDB Module*. Name the new database model **`db`**.

   a. Right-click the folder `myworld/db/src/data` and choose *New* > *CDS Artifact* in the context menu.
   b. Name the new CDS artifact **`PurchaseOrder`**.

      This CDS document will be used to define multiple database objects, for example:

      - Two tables ("`Header`" and "`Item`")
      - Some data types ("`BusinessKey`", "`SDate`", "`CurrencyT`", "`AmountT`", "`QuantityT`", "`UnitT`", and "`StatusT`"
      - A reusable structure "`HistoryT`"
      - An SQL view "`ItemView`"

      The setup Wizard adds the mandatory suffix for CDS artifacts (`.hdbcds`) to the new file name automatically.
5. Define the name space and context in which the entities "Header" and "Item" will be created.

As a first step, add the CDS code for the name space and context.

> ⚠ Caution
>
> The top-level element of a CDS document (in this case the context "`PurchaseOrder`") **must** match the name of the design-time artifact, for example, `PurchaseOrder.hdbcds`.

> Sample Code
>
> purchaseOrder.hdbcds

```
namespace myworld.db.data;
context PurchaseOrder {
};
```

6. Add some data types to the CDS document.

   Enter the CDS code for a collection of data types (for example, `Sdate`, `CurrencyT`, etc) to the CDS document; the types are referenced by the entities "Header" and "Item", which you add in a subsequent step.

> Sample Code

```
namespace myworld.db.data;
context PurchaseOrder {
type BusinessKey : String(10);
    type SDate : LocalDate;
    type CurrencyT : String(5);
    type AmountT : Decimal(15,2);
    type QuantityT : Decimal(13,3);
    type UnitT: String(3);
    type StatusT: String(1);

};
```

7. Add a reusable structure to the CDS document.

   Enter the CDS code for a reusable structure "`HistoryT`" to the CDS document; the elements in the reusable structure are referenced by the entities "Header" and "Item", which you add in a subsequent step.

> Sample Code

```
namespace myworld.db.data;
context PurchaseOrder {
type BusinessKey : String(10);
    type SDate : LocalDate;
    type CurrencyT : String(5);
    type AmountT : Decimal(15,2);
    type QuantityT : Decimal(13,3);
    type UnitT: String(3);
    type StatusT: String(1);

 Type HistoryT {
          CREATEDBY : BusinessKey;
          CREATEDAT : SDate;
          CHANGEDBY : BusinessKey;
          CHANGEDAT : SDate;
        };
};
```

8. Add some simple database tables to the database module.

   Enter the CDS code for the simple entities “Header” and “Item”, as illustrated in the following example.

   Sample Code

```
namespace myworld.db.data;
context PurchaseOrder {
type BusinessKey : String(10);
    type SDate : LocalDate;
    type CurrencyT : String(5);
    type AmountT : Decimal(15,2);
    type QuantityT : Decimal(13,3);
    type UnitT: String(3);
    type StatusT: String(1);

 Type HistoryT {
          CREATEDBY : BusinessKey;
          CREATEDAT : SDate;
          CHANGEDBY : BusinessKey;
          CHANGEDAT : SDate;
        };
    Entity Header {
        key  PURCHASEORDERID: BusinessKey;
        ITEMS: Association[*] to Item on ITEMS.PURCHASEORDERID =
PURCHASEORDERID;
        HISTORY: HistoryT;
        NOTEID: BusinessKey null;
        PARTNER: BusinessKey;
        CURRENCY: CurrencyT;
        GROSSAMOUNT: AmountT;
        NETAMOUNT: AmountT;
        TAXAMOUNT: AmountT;
        LIFECYCLESTATUS: StatusT;
        APPROVALSTATUS: StatusT;
        CONFIRMSTATUS: StatusT;
        ORDERINGSTATUS: StatusT;
        INVOICINGSTATUS: StatusT;
      } technical configuration {
          column store;
      };
    Entity Item {
        key  PURCHASEORDERID: BusinessKey;
        key  PURCHASEORDERITEM: BusinessKey;
        HEADER: Association[1] to Header on HEADER.PURCHASEORDERID =
PURCHASEORDERID;
        PRODUCT:  BusinessKey;
        NOTEID: BusinessKey null;
        CURRENCY: CurrencyT;
        GROSSAMOUNT: AmountT;
        NETAMOUNT: AmountT;
        TAXAMOUNT: AmountT;
        QUANTITY: QuantityT;
        QUANTITYUNIT: UnitT;
        DELIVERYDATE: SDate;
      } technical configuration {
          column store;
      };

};
```

9. Add an SQL view to the database model.

   Enter the CDS code for the simple view “ItemView”, as illustrated in the following example.

Sample Code

```
namespace myworld.db.data;
context PurchaseOrder {
type BusinessKey : String(10);
    type SDate : LocalDate;
    type CurrencyT : String(5);
    type AmountT : Decimal(15,2);
    type QuantityT : Decimal(13,3);
    type UnitT: String(3);
    type StatusT: String(1);

 Type HistoryT {
          CREATEDBY : BusinessKey;
          CREATEDAT : SDate;
          CHANGEDBY : BusinessKey;
          CHANGEDAT : SDate;
        };
    Entity Header {
        key  PURCHASEORDERID: BusinessKey;
        ITEMS: Association[*] to Item on ITEMS.PURCHASEORDERID =
PURCHASEORDERID;
        HISTORY: HistoryT;
        NOTEID: BusinessKey null;
        PARTNER: BusinessKey;
        CURRENCY: CurrencyT;
        GROSSAMOUNT: AmountT;
        NETAMOUNT: AmountT;
        TAXAMOUNT: AmountT;
        LIFECYCLESTATUS: StatusT;
        APPROVALSTATUS: StatusT;
        CONFIRMSTATUS: StatusT;
        ORDERINGSTATUS: StatusT;
        INVOICINGSTATUS: StatusT;
      } technical configuration {
          column store;
      };
    Entity Item {
        key  PURCHASEORDERID: BusinessKey;
        key  PURCHASEORDERITEM: BusinessKey;
        HEADER: Association[1] to Header on HEADER.PURCHASEORDERID =
PURCHASEORDERID;
        PRODUCT:  BusinessKey;
        NOTEID: BusinessKey null;
        CURRENCY: CurrencyT;
        GROSSAMOUNT: AmountT;
        NETAMOUNT: AmountT;
        TAXAMOUNT: AmountT;
        QUANTITY: QuantityT;
        QUANTITYUNIT: UnitT;
        DELIVERYDATE: SDate;
      } technical configuration {
          column store;
      };
   define view ItemView as SELECT from Item {
      PURCHASEORDERID as "PurchaseOrderItemId",
      PURCHASEORDERITEM as "ItemPos",
      HEADER.PARTNER as "PartnerId",
      PRODUCT as "ProductID",
      CURRENCY as "CurrencyCode",
      GROSSAMOUNT as "Amount",
      NETAMOUNT as "NetAmount",
      TAXAMOUNT as "TaxAmount",
      QUANTITY as "Quantity",
      QUANTITYUNIT as "QuantityUnit",
      DELIVERYDATE as "DeliveryDate1"
   } with structured privilege check;
```

```
};
```

10. Create a new file to define a database sequence.

    The tables “Header” and “Item” use a unique order id number as the primary key. As a result, you need a sequence to have an auto incrementing unique id generated for each time new data is inserted into the tables.

    a. Right-click the folder `myworld/db/src/data` and choose *New* > *File* in the context menu.
    b. Name the new file artifact **`orderId.hdbsequence`**.

       Enter the following code to create a non-cycling sequence within your schema starting from 200000000 and ending with 299999999. Make the sequence dependent upon the header table with the full package id on the front of the header table name.

       Sample Code

       ```
       SEQUENCE "myworld.db.data::orderId"
       INCREMENT BY 1
       START WITH 200000000
       MINVALUE 1
       MAXVALUE 2999999999
       NO CYCLE
       RESET BY SELECT IFNULL(MAX(PURCHASEORDERID), 0)+1
       FROM "myworld.db.data::PurchaseOrder.Header"
       ```

    c. Save the sequence file.

11. Load some initial data into the new tables `Header` and `Item`.

    In HDI, you can import data from a comma-separated-value file (CSV) into a database table; you need the following design-time artifacts:

    - CSV file (`myData.csv`)
      Holds the data you want to load into the database table.
    - Table definition file (`myTargetTable.hdbtabledata`)
      Specifies the target table for the data in the source `.csv` file to be uploaded.

    a. Create the `.hdbtabledata` file.

       In the database module's `data/` folder, create a file called `Purchase.hdbtabledata` and add the following code:

       Sample Code

       `Purchase.hdbtabledata`

       ```
       {
       "format_version": 1,
       "imports": [{
       "target_table": "myworld.db.data::PurchaseOrder.Header",
       "source_data": {
       “data_type": "CSV",
           "file_name": "myworld.db.data::header.csv",
           "has_header": false,
           "dialect": "HANA",
           "type_config": {
           "delimiter": ","
           }
       },
       ```

```
"import_settings": {
"import_columns": [
"PURCHASEORDERID",
    "NOTEID",
    "PARTNER",
    "CURRENCY",
    "GROSSAMOUNT",
    "NETAMOUNT",
    "TAXAMOUNT",
    "LIFECYCLESTATUS",
    "APPROVALSTATUS",
    "CONFIRMSTATUS",
    "ORDERINGSTATUS",
    "INVOICINGSTATUS"]
},
"column_mappings": {
    "PURCHASEORDERID": 1,
    "NOTEID": 6,
    "PARTNER": 7,
    "CURRENCY": 8,
    "GROSSAMOUNT": 9,
    "NETAMOUNT": 10,
    "TAXAMOUNT": 11,
    "LIFECYCLESTATUS": 12,
    "APPROVALSTATUS": 13,
    "CONFIRMSTATUS": 14,
    "ORDERINGSTATUS": 15,
    "INVOICINGSTATUS": 16
    }
},
{
"target_table": "myworld.db.data::PurchaseOrder.Item",
"source_data": {
    "data_type": "CSV",
    "file_name": "myworld.db.data::item.csv",
    "has_header": false,
    "dialect": "HANA",
    "type_config": {
    "delimiter": ","
    }
},
"import_settings": {
"import_columns": [
"PURCHASEORDERID",
    "PURCHASEORDERITEM",
    "PRODUCT",
    "NOTEID",
    "CURRENCY",
    "GROSSAMOUNT",
    "NETAMOUNT",
    "TAXAMOUNT",
    "QUANTITY",
    "QUANTITYUNIT" ]
    },
"column_mappings": {
    "PURCHASEORDERID": 1,
    "PURCHASEORDERITEM": 2,
    "PRODUCT": 3,
    "NOTEID": 4,
    "CURRENCY": 5,
    "GROSSAMOUNT": 6,
    "NETAMOUNT": 7,
    "TAXAMOUNT": 8,
    "QUANTITY": 9,
    "QUANTITYUNIT": 10
    }
}]
```

```
}
```

b. Create the `header.csv` file.

In the database module's `data/` folder, create a file called `header.csv` and add the following content and save the file:

Sample Code

`header.csv`

```
0500000000,0000000033,20120101,0000000033,20120101,9000000001,0100000000
,EUR,13224.47,11113,2111.47,N,I,I,I,I
0500000001,0000000033,20120102,0000000033,20120102,9000000001,0100000002
,EUR,12493.73,10498.94,1994.79,N,I,I,I,I
```

c. Create the `item.csv` file.

In the database module's `data/` folder, create a file called `item.csv` and add the following content and save the file:

Sample Code

`item.csv`

```
0500000000,0000000010,HT-1000,,EUR,1137.64,956,181.64,1,EA,20121204
0500000000,0000000020,HT-1091,,EUR,61.88,52,9.88,2,EA,20121204
0500000000,0000000030,HT-6100,,EUR,1116.22,938,178.22,2,EA,20121204
0500000000,0000000040,HT-1000,,EUR,2275.28,1912,363.28,2,EA,20121204
0500000000,0000000050,HT-1091,,EUR,92.82,78,14.82,3,EA,20121204
0500000000,0000000060,HT-6100,,EUR,1116.22,938,178.22,2,EA,20121204
0500000000,0000000070,HT-1000,,EUR,2275.28,1912,363.28,2,EA,20121204
0500000000,0000000080,HT-1091,,EUR,61.88,52,9.88,2,EA,20121204
0500000000,0000000090,HT-6100,,EUR,1674.33,1407,267.33,3,EA,20121204
0500000000,0000000100,HT-1000,,EUR,3412.92,2868,544.92,3,EA,20121204
0500000001,0000000010,HT-1100,,USD,213.96,179.8,34.16,2,EA,20121204
0500000001,0000000020,HT-2026,,USD,35.69,29.99,5.7,1,EA,20121204
0500000001,0000000030,HT-1002,,USD,3736.6,3140,596.6,2,EA,20121204
0500000001,0000000040,HT-1100,,USD,213.96,179.8,34.16,2,EA,20121204
0500000001,0000000050,HT-2026,,USD,71.38,59.98,11.4,2,EA,20121204
0500000001,0000000060,HT-1002,,USD,3736.6,3140,596.6,2,EA,20121204
0500000001,0000000070,HT-1100,,USD,320.94,269.7,51.24,3,EA,20121204
0500000001,0000000080,HT-2026,,USD,107.06,89.97,17.09,3,EA,20121204
0500000001,0000000090,HT-1002,,USD,3736.6,3140,596.6,2,EA,20121204
0500000001,0000000100,HT-1100,,USD,320.94,269.7,51.24,3,EA,20121204
```

12. Create a new database synonym.

A database synonym is required to enable access to any table or view outside of our application's container. In this step, to enable us to access the DUMMY view, you create **two** files: an synonym artifact (`.hdbsynonym`).

In the database module's `data/` folder, create a file called `general.hdbsynonym`, add the following content, and save the file:

Sample Code

`general.hdbsynonym`

```
{
  "myworld.db.data::DUMMY" : {
```

```
    "target" : {
            "schema" : "SYS",
            "object" : "DUMMY"
    }
  }
}
```

13. Create a new structured privilege.

    The structured privilege is the logical successor to the analytic privilege; it enables us to perform instance filtering for the CDS view `ItemView` that we created earlier.

    a. In the database module's `data` folder, create a new subfolder named `roles`, for example, `db/data/roles`.

    In the database module's `data/roles` folder, create a file called `PurchaseOrder.hdbstructuredprivilege`, add the following content, and save the file:

    Sample Code

    `PurchaseOrder.hdbstructuredprivilege`

```
STRUCTURED PRIVILEGE
  "myworld.db.roles::PO_VIEW_PRIVILEGE"
  FOR SELECT ON
  "myworld.db.data::PurchaseOrder.ItemView"
WHERE "CurrencyCode" = 'EUR'
```

14. Create a new role for the structured privilege.

    The technical user bound to the application's HDI container enables access to the HDI database objects from XS advanced. If you want to enable access by other database users (for example, external reporting tools) you must create a database role.

    In the database module's `data/roles` folder, create a file called `dev007.hdbrole`, add the following content, and save the file:

    Sample Code

    `dev007.hdbrole`

```
{
  "role":{
          "name": "myworld.db.roles::dev007",
          "object_privileges":[
          {
          "name": "myworld.db.data::PurchaseOrder.Header",
          "type": "TABLE",
          "privileges": [ "SELECT" ]
          },
          {
          "name": "myworld.db.data::PurchaseOrder.Item",
          "type": "TABLE",
          "privileges": [ "SELECT" ]
          },
          {
          "name": "myworld.db.procedures::getPOItems",
          "type": "PROCEDURE",
          "privileges": [ "EXECUTE" ]
          },
          {
          "name": "myworld.db.data::PurchaseOrder.ItemView",
          "type": "VIEW",
```

```
            "privileges": [ "SELECT" ]
            }
            ],
            "schema_analytic_privileges": [
            {
            "privileges":[ "myworld.db.roles::PO_VIEW_PRIVILEGE" ]
            }
          ]
    }
}
```

15. Check that the required plug-ins are available to build and deploy the database objects defined in the CDS document.

    In SAP HANA XS advanced, a file suffix (for example, `.hdbcds`) is linked to a specific plug-in, which is used to generate the corresponding catalog object from a design-time artifact.

    The container-configuration file `db/src/.hdiconfig` must contain (at least) the following entries:

    > **i** Note
    >
    > The version number of the plug-in can differ according to the installed version of SAP HANA.

    > Sample Code
    >
    > `db/src/.hdiconfig`

```
"hdbcds": {
     "plugin_name": "com.sap.hana.di.cds",
     "plugin_version": "12.1.0"
     },
"hdbsequence": {
     "plugin_name": "com.sap.hana.di.sequence",
     "plugin_version": "12.1.0"
     },
"hdbtabledata" : {
     "plugin_name" : "com.sap.hana.di.tabledata",
     "plugin_version": "12.1.0"
     },
"csv" : {
     "plugin_name" : "com.sap.hana.di.tabledata.source",
     "plugin_version": "12.1.0"
     },
"hdbsynonym" : {
     "plugin_name" : "com.sap.hana.di.hdbsynonym",
     "plugin_version": "12.1.0"
     },
"hdbstructuredprivilege" : {
     "plugin_name" : "com.sap.hana.di.structuredprivilege",
     "plugin_version": "12.1.0"
     },
"hdbrole" : {
     "plugin_name" : "com.sap.hana.di.role",
     "plugin_version": "12.1.0"
     },
"hdbroleconfig" : {
     "plugin_name" : "com.sap.hana.di.role.config",
     "plugin_version": "12.1.0"
     }
```

> → Tip
>
> By default, the `.hdiconfig` configuration file is hidden. To display it, choose *View* > *Show Hidden Files* in the menu bar.

16. Build the HDB module `db`.

    Building a database module activates the data model and creates the corresponding objects in the database catalog for each artifact defined in the CDS document. In this case, the build creates one database two tables named “Header” and “Item”, an SQL view named “ItemView”.

    a. In the SAP Web IDE for SAP HANA, choose *Build* > *Build*.

    If the builder displays the message `(Builder) Build of /myworld/db completed` in the SAP Web IDE console, the data-model was successfully activated in a HANA database container, and can now be used to store and retrieve data.

Related Information

Create a Database Procedure in XS Advanced [page 470]
Create a Database Function in XS advanced [page 473]

5.5.2 Create a Database Procedure in XS Advanced

Create, edit, and deploy procedures.

Prerequisites

- You have access to a running SAP HANA system where the XS advanced run time is installed and configured
- You have access to the developer tools included in SAP Web IDE for SAP HANA
  If SAP Web IDE for SAP HANA is not available at the default URL `https://host.acme.com:53075`, log in to the XS advanced run time and use the command `xs app webide` to display the URL required to access the tool.
- You have created the database tables `Header` and `Item` described in *Create the Underlying Data Artifacts for a Procedure*. For more information, see *Related Links* below.

> ⚠ Caution
>
> The code examples included in this document for XS advanced are sometimes syntactically incomplete; as a general rule, code examples are intended for illustration purposes only.

Context

In XS advanced a database procedure has the file extension `.hdbprocedure`, for example, `myProcedure.hdbprocedure`. To create and deploy a procedure in XS advanced using the SAP Web IDE for SAP HANA, perform the following steps:

Procedure

1. Start the SAP Web IDE for SAP HANA.
2. Display the application project to which you want to add a database procedure.

   An application is created within a context of a project. If you do not already have a project, there are a number of ways to create one, for example: by importing it, cloning it, or creating a new one from scratch.

   a. In the SAP Web IDE for SAP HANA, choose *File* > *New* > *Project from Template*.
   b. Choose the project template type.

      Currently, there is only one type of project template available, namely: *Multi-Target Application Project*. Select *Multi-Target Application Project* and choose *Next*.
   c. Type the name **`myworld`** for the new MTA project and choose *Next* to confirm.
   d. Specify details of the new MTA project and choose *Next* to confirm.
   e. Create the new MTA project; choose *Finish*.
3. Create a stored procedure that enables you to add content to a database table .

   The procedure you create in this step uses a `SELECT INNER JOIN` to retrieve data from two database tables, `Header` and `Item` (created in a previous task), filter the data by partner role, and join the results into the output parameter.

   a. Create a new folder for your stored procedures.

      In the `db/` module create a folder named `procedures` in the `src/` folder, for example, `myworld/db/src/procedures`.
   b. Create a new `.hdbprocedure` file for your stored procedure.

      Right-click the database-module folder `db/src/procedures`, choose *New* > *Procedure* from the context menu, and name the new stored procedure **`get_bp_addresses_by_role`**.

      The new stored procedure opens in the SAP Web IDE for SAP HANA.
   c. Add the following SQLScript code to the procedure file `get_bp_addresses_by_role.hdbprocedure` and save the file.

      Sample Code

```
PROCEDURE "myworld.db.procedures::get_bp_addresses_by_role" (
    in im_partnerrole nvarchar(3),
    out ex_bp_addresses "myworld.db.data::Procedures.tt_bp_addresses"
)
   LANGUAGE SQLSCRIPT
   SQL SECURITY INVOKER
   --DEFAULT SCHEMA <default_schema_name>
   READS SQL DATA AS
BEGIN
```

```
    /*********************************
        Write your procedure logic
    *********************************/
ex_bp_addresses =
     select a."PARTNERID", a."PARTNERROLE",
            a."EMAILADDRESS", a."COMPANYNAME",
            a."ADDRESSES.ADDRESSID" as "ADDRESSID",
            b."CITY", b."POSTALCODE", b."STREET"
     from "myworld.db.data::MD.BusinessPartner" as a
     inner join "myworld.db.data::MD.Addresses" as b
     on a."ADDRESSES.ADDRESSID" = b."ADDRESSID"
     where a."PARTNERROLE" = :im_partnerrole;
END
```

d. Build the application database module `db`.

   In the SAP Web IDE for SAP HANA, right-click the application module *myworld/db*, choose *Build* > *Build* in the context menu, and follow the build progress in the output displayed in the console pane.

   ```
   2:10:51 PM (Builder) Build of /myworld/db completed successfully.
   ```

4. Check that the required plug-in is available to build and deploy the stored procedure.

   In SAP HANA XS advanced, a file suffix (for example, `.hdbprocedure`) is linked to a specific plug-in, which is used to generate the corresponding catalog object from a design-time artifact.

   The container-configuration file `.hdiconfig` must contain the following entry (version numbers can differ):

   Sample Code

   ```
   "hdbprocedure": {
               "plugin_name": "com.sap.hana.di.procedure",
               "plugin_version": "11.1.0"
           },
   ```

   → Tip

   By default, the `.hdiconfig` file is hidden. To display it, choose *View* > *Show Hidden Files* in the menu bar.

5. Test the stored procedure `createCountry.hdbprocedure`.

   Use the database explorer to test the stored procedure and add some content to your database table `myworld.db::tinyf.country`.

   a. In the database browser, open the stored procedure `myworld.db::createCountry`.

      The stored procedure is located in the application container `<UserName>-<ID>-myworld-hdi-container` in the folder *Procedures*. Right-click the procedure and choose *Invoke Procedure with UI* in the context menu.

   b. Add a country that does **not** already exist in the database, for example, “Ireland”.

      In the *IM_COUNTRY* box, enter **`Ireland`**; In the *IM_CONTINENT* box, enter **`Europe`** and choose *Run* in the tool bar (or press F8).

   c. Check that the new country was added to the table *myworld.db::tinyf.country*.

The table is located in the application container `<UserName>-<ID>-myworld-hdi-container` in the folder *Tables*. Right-click the table *myworld.db::tinyf.country* and choose *Open Content* in the context menu.

The country you added (for example, "Ireland") should be displayed at the bottom of the table.

d. Add a country that already exists in the database, for example, "Ireland".

In the *IM_COUNTRY* box, enter **`Ireland`**; in the *IM_CONTINENT* box, enter **`Europe`** and choose *Run* in the tool bar (or press F8).

```
ERROR: Country Ireland already exists!
```

Related Information

Create a Database Function in XS advanced [page 473]
Creating Data Persistence Artifacts with CDS in XS Advanced [page 310]

5.5.3 Create a Database Function in XS advanced

Add a database function to your data model.

Prerequisites

- You have access to a running SAP HANA system where the XS advanced run time is installed and configured.
- You have access to the developer tools included in SAP Web IDE for SAP HANA
  If SAP Web IDE for SAP HANA is not available at the default URL `https://host.acme.com:53075`, log in to the XS advanced run time and use the command `xs app webide` to display the URL required to access the tool.

> ⚠ Caution
>
> The code examples included in this document for XS advanced are sometimes syntactically incomplete; as a general rule, code examples are intended for illustration purposes only.

Context

In SQL, a user-defined function (UDF) enables you to build complex logic into a single database object. A scalar UDF is a custom function that can be called in the SELECT and WHERE clauses of an SQL statement. In XS advanced a database function (scalar or table) has the file extension `.hdbfunction`, for example,

`myScalarFunction.hdbfunction`. To create and deploy a procedure in XS advanced using the SAP Web IDE for SAP HANA, perform the following steps:

Procedure

1. Start the SAP Web IDE for SAP HANA.
2. Display the application project to which you want to add a database function.

   An application is created within a context of a project. If you do not already have a project, there are a number of ways to create one, for example: by importing it, cloning it, or creating a new one from scratch.

   a. In the SAP Web IDE for SAP HANA, choose *File* > *New* > *Project from Template*.
   b. Choose the project template type.

      Currently, there is only one type of project template available, namely: *Multi-Target Application Project*. Select *Multi-Target Application Project* and choose *Next*.
   c. Type the name **`myworld`** for the new MTA project and choose *Next* to confirm.
   d. Specify details of the new MTA project and choose *Next* to confirm.
   e. Create the new MTA project; choose *Finish*.
3. Create a database function.

   Database functions must be created in the application project's database module, for example, `db`.
   A scalar user-defined function (UDF) has a list of input parameters and returns the scalar values specified in the `RETURNS <return parameter list>` option defined in the SQL function, for example, `decimal(15,2)`. The scalar UDF named "`apply_discount`" that you create in this tutorial performs the following actions:

   - Applies a discount to the stored product price
   - Calculates the sale price of a product including the suggested discount

   a. Create a new folder for your functions.

      If it does not already exist, create a folder named `functions` in the `src/` folder of your application project's `db/` module, for example, `myworld/db/src/functions`.
   b. Create a new `.hdbfunction` file for your scalar user-defined function.

      Right-click the application's database-module folder `db/src/procedures`, choose *New* > *Function* from the context menu, and name the new stored scalar function **`apply_discount`**.

      The new scalar function opens in the SAP Web IDE for SAP HANA.
   c. Add the following SQLScript code to the function-definition file `apply_discount.hdbfunction` and save the file.

      Sample Code

      ```
      FUNCTION "myworld.db::apply_discount" (im_price decimal(15,2),
      im_discount decimal(15,2) )
          RETURNS result decimal(15,2)
          LANGUAGE SQLSCRIPT
          SQL SECURITY INVOKER AS
      BEGIN
      result := :im_price - ( :im_price * :im_discount );
      ```

```
END;
```

d. Build the application database module `db`.

In the SAP Web IDE for SAP HANA, right-click the application module *myworld/db*, choose *Build* > *Build* in the context menu, and follow the build progress in the output displayed in the console pane.

```
2:10:51 PM (Builder) Build of /myworld/db completed successfully.
```

4. Check that the required plug-in is available to build and deploy the function.

In SAP HANA XS advanced, a file suffix (for example, `.hdbfunction`) is linked to a specific plug-in, which is used to generate the corresponding catalog object from a design-time artifact.

The container-configuration file `.hdiconfig` must contain the following entry (version numbers can differ):

Sample Code

```
"hdbfunction": {
            "plugin_name": "com.sap.hana.di.function",
            "plugin_version": "11.1.0"
        },
```

Tip

By default, the `.hdiconfig` configuration file is hidden. To display it, choose *View* > *Show Hidden Files* in the menu bar.

5. Test the new scalar function `apply_discount.hdbfunction`.

Use the SAP HANA database explorer included with the Web IDE to test the`apply_discount.hdbfunction` function.

a. Open the database explorer from the Web IDE by choosing *Tools* > *Database Explorer*.
b. If your HDI container is not listed in the database browser, add it by choosing *Add Database to Database Explorer* from the database browser toolbar.

The HDI container is named `<UserName>-<ID>-myworld-hdi-container`

c. In the database browser, open the function `apply_discount.hdbfunction`.

The function is located in the folder *Functions*.

d. The function opens in a tab on the right.

Related Information

Create a Database Procedure in XS Advanced [page 470]
Creating Procedures and Functions in XS Advanced [page 459]

5.6 Using Synonyms to Access External Schemas and Objects in XS Advanced

Deploy synonyms to enable cross-container access to external objects in XS advanced.

The information in this section covers the following basic application-development scenarios:

- Enable access to tables owned by a classical schema from an XS-advanced-based schema (HDI container). For example, enable access to an existing ERP schema from XS advanced.
- Enable access to tables owned by one XS-advanced-based schema from another XS-advanced-based schema.
  For example, enable one HDI container to access another HDI container, or enable an XS advanced Data Warehouse application to access a schema containing data in another XS advanced application's HDI container.

> **i Note**
>
> For more information about authorization checks on objects with dependencies, see the *Object Privileges* section in the *SAP HANA Security Guide*, which is listed in *Related Information* below.

Related Information

Enable Access to Objects in a Remote Classic Schema [page 476]
Enable Access to Objects in Another HDI Container [page 505]
SAP HANA Security Guide (Authorization, Object Privileges)

5.6.1 Enable Access to Objects in a Remote Classic Schema

Use a synonym to enable access to objects in a remote schema not managed by XS advanced (for example, ERP).

Prerequisites

To complete the steps described in this task, the following prerequisites apply:

- You have access to the XS advanced run-time environment
- You have access to SAP Web IDE for SAP HANA
- You have access to the XS command-line interface client
- You have access to a “classic” SAP database schema in a remote system (for example, SAP NetWeaver EPM)

Context

You can enable an XS advanced application to access objects in a remote schema; that is, a classic schema in a remote database. The target objects for the synonyms can be tables, views, functions, and procedures, as well as database sequences.

> **i** Note
>
> For illustration purposes only, the target (base) objects used in this example are taken from SAP NetWeaver EPM content (Sales Order Model). However, you can substitute the target objects specified in the examples with your own target objects. Similarly, you can substitute the synonyms, roles, etc. specified in the examples with your own artifacts.

Procedure

1. Create the underlying target objects in the external schema (for example, `EPM_DEV`) to which the XS advanced application's synonym points.

   The target objects can be tables, views, functions, procedures, and so on. You might have a number of tables in a schema, not all of which should be exposed by a synonym.

   > → Tip
   >
   > If the target objects already exist, you can skip this step.

2. Create the roles to be granted for external access by means of the synonym.

   The following roles are required here:

   - `external_access_g`
     Grant select privilege on schema `EPM_DEV` to "`EPM_XXX::external_access_g`" **with** grant option to allow access to the whole schema

     ```
     create role "EPM_XXX::external_access_g";
     grant select on SNWD_AD to "EPM_XXX::external_access_g" with grant option;
     grant select on SNWD_BPA to "EPM_XXX::external_access_g" with grant option;
     grant select on SNWD_PD to "EPM_XXX::external_access_g" with grant option;
     grant select on SNWD_PD_CATGOS to "EPM_XXX::external_access_g" with grant
     option;
     grant select on SNWD_SO to "EPM_XXX::external_access_g" with grant option;
     grant select on SNWD_SO_I to "EPM_XXX::external_access_g" with grant
     option;
     grant select on SNWD_SO_SL to "EPM_XXX::external_access_g" with grant
     option;
     ```

   - `external_access_appuser`
     Grant select privilege on schema `EPM_DEV` to "`EPM_XXX::external_access_appuser`" to enable access **without** grant option to the whole schema.

     ```
     create role "EPM_XXX::external_access_appuser";
     grant select on SNWD_AD to "EPM_XXX::external_access_appuser";
     grant select on SNWD_BPA to "EPM_XXX::external_access_appuser";
     grant select on SNWD_PD to "EPM_XXX::external_access_appuser";
     grant select on SNWD_PD_CATGOS to "EPM_XXX::external_access_appuser";
     grant select on SNWD_SO to "EPM_XXX::external_access_appuser";
     ```

```
grant select on SNWD_SO_I to "EPM_XXX::external_access_appuser";
grant select on SNWD_SO_SL to "EPM_XXX::external_access_appuser";
```

3. Create the synonym(s).

   You can create the synonym design-time object in a subfolder of your XS advanced application's database module, for example in `/<MyApp>/db/src/synonyms/SNWD.hdbsynonym`. In this example, we name the synonym definition `SNWD.hdbsynonym`, which contains multiple synonyms referencing tables/views in the target schema "`EPM_DEV`".

   > ### Sample Code
   >
   > XS Advanced Synonym Definition (`/MyApp/db/src/synonyms/SNWD.hdbsynonym`)
   >
   > ```
   > {
   >   "SNWD_AD": {
   >     "target": {
   >       "object": "SNWD_AD",
   >       "schema": "EPM_DEV"
   >     }
   >   },
   >   "SNWD_BPA": {
   >     "target": {
   >       "object": "SNWD_BPA",
   >       "schema": "EPM_DEV"
   >     }
   >   },
   > [... synonyms {} ...]
   >   "SNWD_SO_SL": {
   >     "target": {
   >       "object": "SNWD_SO_SL",
   >       "schema": "EPM_DEV"
   >     }
   >   }
   > }
   > ```

4. Grant access to the synonym's target objects (for example, table); the access can also be used by any views that "consume" the target objects.

   Access privileges are required on the target objects to which the synonym points. The type of access required (`SELECT/EXECUTE`) depends on the object type: `SELECT` for tables, views, and functions; `EXECUTE` for functions and procedures.

   The access mode is only relevant for target objects that are procedures and views. The access mode specifies the user whose privileges will be checked upon execution of the procedure. For procedures, the developer must explicitly specify the desired access mode (for example, `DEFINER/INVOKER`) when writing the procedure. Views are always DEFINER mode.

   Access can be enabled by referencing defined user roles with the `"roles"` property in an `.hdbgrants` file, as illustrated in the following example:

   > ### Caution
   >
   > To prevent unwanted schema access, for example, via "definer-mode", it is recommended to assign different roles to the object owner and the application user.

   > ### Sample Code
   >
   > `myApp/db/cfg/epm_dev.hdbgrants` File
   >
   > ```
   > {
   > ```

```
  "EPM_XXX-table-grantor": {
     "object_owner": {
           "roles": [
               "EPM_XXX::external_access_g"
           ]
     },
     "application_user": {
           "roles": [
               "EPM_XXX::external_access_appuser"
           ]
     }
  }
}
```

"`EPM_XXX-table-grantor`" is a symbolic name for a user-defined service that executes the `GRANT` statement. The user-defined service must be specified in the application project's development-descriptor (`mta.yaml`). "`application_user`" refers to a technical access role; it is not a reference to a real user.

The role specified for assignment to **"`object_owner`"** should contain the `SELECT WITH GRANT OPTION` privilege on the corresponding target table; the role specified for assignment to **"`application_user`"** should specify the `SELECT` privilege (**without** `GRANT OPTION`) for the corresponding target tables, view, sequences, and `EXECUTE WITH GRANT OPTION` for the corresponding functions and procedures to enable access to the target objects via the synonyms created.

5. Add references to **"`EPM_XXX-table-grantor`"** in the development descriptor (`mta.yaml`) for the application that needs to use the synonym (to access the target objects).

Sample Code

Application Development Descriptor (`MyApp/mta.yaml`)

```
_schema-version: '2.0'
ID: syn-hdi-classic-1
version: 0.0.1
modules:
  - name: db
    type: hdb
    path: db
    requires:                                        # db module needs:
      - name: hdi-container                          # where synonyms are
created
        properties:
          TARGET_CONTAINER: ~{hdi-container-service}

      - name: EPM_XXX-table-grantor                  # for executing grant
statement

resources:
  - name: hdi-container
    type: com.sap.xs.hdi-container
    properties:
      hdi-container-service: ${service-name}
  - name: EPM_XXX-table-grantor
    type: org.cloudfoundry.existing-service        # service created with
xs cups
```

6. Create a user-defined service (for example, `EPM_XXX-table-grantor`) in the XS advanced application developer space.

The "user provided service" enables you to assign roles automatically to the generated users of the HDI container. The service uses the permissions assigned to a specified user to connect to a database

and execute grant statements during application deployment. The grant statements are generated from content defined in an `.hdbgrants` file such as the one defined in a previous step.

> **i Note**
>
> XS advanced services have access to any other service running in the same organization and space. This means that any service running in the same organization and space as the user-provided service you create in this step can use the user-provided service to access external objects.

a. Open a command shell and log on to the XS CLI with XS administrator privileges.

```
xs-admin-login
```

b. Change to the target space where you want to create the user-defined service.

```
xs target –s <SPACE>
```

c. Create the user-defined service `EPM_XXX-table-grantor`.

```
xs cups EPM_XXX-table-grantor –p
"{\"host\":\"host.acme.com\",\"port\":\"30015\",\"certificate\":\"<myCertif
icate>\",
\"user\":\"EPM_DEV\",\"password\":\"Grant_123\",\"driver\":\"com.sap.db.jdb
c.Driver\",\"tags\":[\"hana\"] , \"schema\" : \"EPM_XXX\" }"
```

This command creates a user-defined service with the following configuration:

```
"EPM_XXX-table-grantor": [
  {
    "schema": "EPM_XXX",
    "password": "Grant_123",
    "driver": "com.sap.db.jdbc.Driver",
    "port": "30015",
    "host": "host.acme.com",
    "user": "EPM_DEV",
    "certificate":"<myCert>"
    "tags": [ "hana" ]
  }
]
```

- `"schema"`
  The database schema that contains the objects to which access is to be granted.
- `"user"/"password"`
  Connection details for a database user that has grant permissions for the objects in the schema.
- `"host"/"port"`
  Required for the connection to the database: `port` is the SQL port of the index server.
- `"driver"`
  Required to set up the client connection to the database
- `"certificate"`
  If the database is configured to only accept **secure** JDBC connections, then the grantor service requires an SSL certificate that must be included in the user-provided service, for example, using the `"certificate":"<myCertificate>"` parameter.

  > **→ Tip**
  >
  > Starting with version 3.0.0 of the HDI deployer service, the `"host"`, `"port"`, and `"certificate"` parameters are no longer required since they can be obtained from the

target container binding. In this case, you must only specify the `"user"`, `"password"`, and `"schema"` when creating the new user-provided service.

d. Check that the new user-defined service is running in the target space.

   Use the command `xs services` to display a list of services available in the current space; the following service should be visible:

   Output Code

```
xs services
Getting services in org "myorg" / space "DEV" as XSA_ADM...
Found services:
name                    service         plan      bound apps
-------------------------------------------------------------
EPM_XXX-table-grantor    user-provided
```

7. Build the XS advanced application's DB module.

   You can use the tools included in SAP Web IDE for SAP HANA.

8. Consume the synonym(s) created so far.

   One obvious and simple way to consume synonyms is to use a view (called `CUSTOMERS` in this example) on a table.

   Sample Code

   View `/myApp/db/src/CUSTOMERS.hdbview`

```
VIEW "CUSTOMERS" AS SELECT
POSTAL_CODE, CITY, STREET, BUILDING
FROM SNWD_AD
```

9. Check for the resulting objects in the database catalog.

   You can use the *Database Explorer* tools included in SAP Web IDE for SAP HANA.

10. Create a role definition that enables access to the view.

    → Tip

    The view is not **required** for the creation of a synonym; it is used here to test “consumption” of the synonym.

    Sample Code

    Role Definition `external_v_access.hdbrole`

```
{
  "role": {
    "name": "external_v_access",
    "object_privileges": [
      {
        "name":"CUSTOMERS",
        "type":"VIEW",
        "privileges":[ "SELECT" ]
      }
    ]
  }
```

```
}
```

11. Assign the role to consumers of the view.

    The new role `external_v_access.hdbrole` must be assigned to the to the container-external user (for example, `SYN_CONS`) who needs access to the view (in this example, `CUSTOMERS`).

12. Test access to the view.

    Both the XS advanced application user **and** the schema-external user should now have access to the view, which you can test by opening it in the *Database Explorer*.

    - XS advanced application user
      With the container's access role, this user has access to all objects in the application container's run-time schema.
    - Schema-external user
      Access to the view is granted by assignment of the role `external_v_access`.

Related Information

Database Synonyms in XS Advanced [page 482]
Syntax Options in the hdbgrants File [page 497]
Roles (.hdbrole and .hdbroleconfig)
Enable Access to Objects in Another HDI Container [page 505]

5.6.1.1 Database Synonyms in XS Advanced

You can use synonyms in XS advanced to enable access to objects that are not in the same schema or application container.

In general terms a synonym is an alias for a database object. Whenever you use a synonym, it is useful to try to mentally replace it by its base object: the object that is the target of the synonym. A synonym belongs to its own schema, which is typically (but not always) independent of the schema to which the target database object belongs. Synonyms can be created for tables, views, procedures, table functions, scalar functions and sequences. Synonyms are most commonly used to hide the real object names from consumers or to give a database object a more convenient name. In migration scenarios, it also sometimes necessary to be able to recreate the synonym after modifying it to point to another target object.

To create a synonym, a user needs the following privileges:

- SELECT
  For the tables, views, and sequences to which the synonym refers as base objects
- EXECUTE
  For procedures and functions to which the synonym refers as base object.

After creation, the synonym exists independently of the base target object. A synonym remains in existence even if its creator no longer has the privilege required to access the synonym's target database object or the base target object has been deleted.

> **i** Note
>
> A synonym can be “private” (belonging to a specific schema), or “public” (residing in the special schema `PUBLIC`).

Although a public synonym can be used by any user without explicitly specifying the schema name, the appropriate privilege for accessing the base object of a public synonym is still required. If public and private objects or synonyms have identical names, the private synonym is used.

> **!** Restriction
>
> It is not possible to use a synonym as the base (target) object for another synonym.

If you need to point a synonym to another synonym, you have to put a real database object in between the two synonyms. For example, you would need to create a view for a scenario where the second synonym's base object is a table or a table function. If the second synonym points to a procedure, you must insert an additional procedure between the synonyms. If the second synonym points to a scalar function, you must insert an additional scalar function. It is not possible to use a sequence as the base target object for another sequence with a synonym in between.

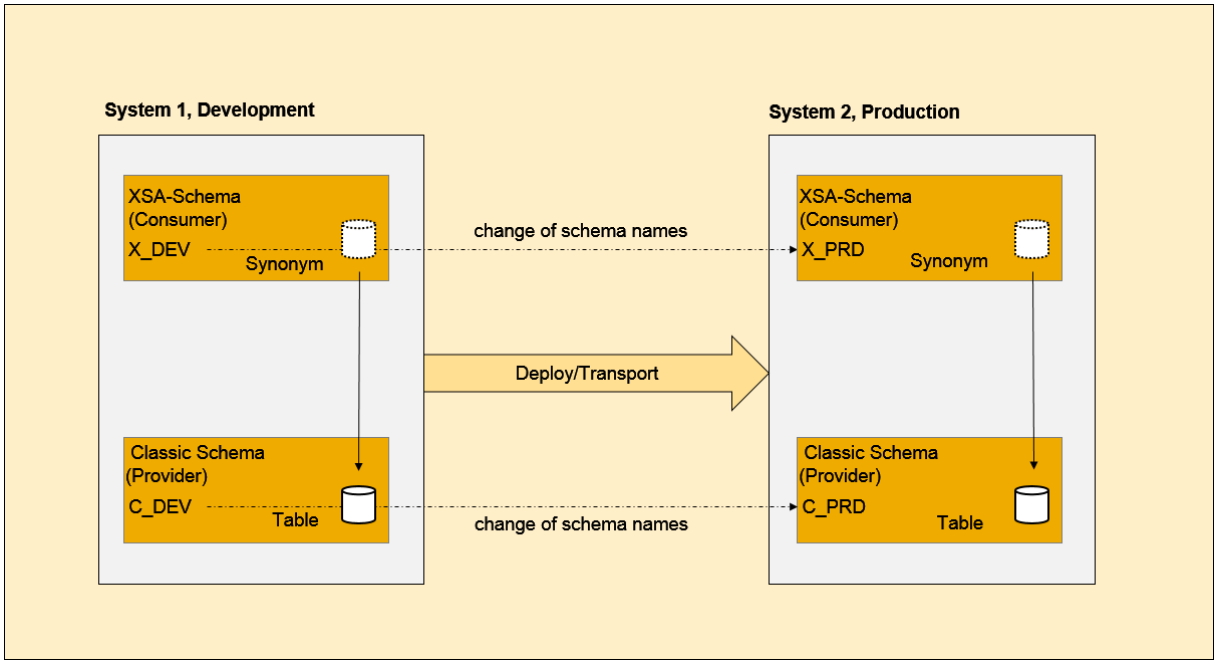
Synonyms in XS Advanced

In XS advanced database objects such as tables and views reside in an application-specific container, which is a sort of generated schema. For this reason, development has to be done in a schema-less way. The concept of isolated HDI containers makes it possible to deploy multiple containers into the same system, have multiple developers work on the same application independently from other developers, or even deploy applications and containers into a public Cloud environment, where binding between different containers (or services in general) should not be hard coded but done using a configuration mechanism during deployment.

Synonyms play a more important role in XS advanced and HDI than they do in the schemas used in XS classic. In XS advanced, using synonyms is the designated method to enable access to objects in other schemas, for example, a schema owned by another XS advanced application container.

The following diagram provides a visual overview of the most commonly used scenarios where synonyms in XS advanced can be effectively deployed to access objects in classic schemas outside of XS advanced, for example:

- Enabling access to tables provided by a classic (ERP) schema from an XS-advanced schema (an HDI container)
- Enabling access to tables provided by one XS-advanced schema from a different XS-advanced schema, for example, one schema containing data and different XS advanced Data Warehouse (DW) applications accessing the data
- Transporting schemas and the objects they contain to (and deploying them in) another environment



In the context of database synonyms, it is important to understand the role of the following users:

- Schema owner
  The user who owns the HDI container (or schema) where the synonym resides
- Object owner
  The user who creates the objects deployed within the container, for example, the synonym. The synonym's "target" objects belong to another user, for example, the user of the external schema.
- Application user
  The SAP HANA run-time user which runs the XS advanced applications in the HDI container

Related Information

Using Synonyms to Access External Schemas and Objects in XS Advanced [page 476]
Syntax Options in the hdbgrants File [page 497]

5.6.1.2 Users, Privileges, and Schemas

A short overview of which properties, users, and privileges are involved when using synonyms in XS advanced.

For security reasons, some database objects can be run in either `DEFINER` or `INVOKER` mode. The security mode is not defined in an access role; it is defined by the database object type, or in the case of a procedure, explicitly. The security mode is determined, in part, by the **type** of database object the application user calls, for example, a view, a synonym, or a procedure. More importantly for this XS advanced scenario, the security mode also determines which privileges need to be specified in the user role required to access the database object.

- Definer-Mode Access [page 485]
- Invoker-Mode Access [page 487]

> → Tip
>
> In the default DEFINER security mode, a database object (for example, a definer-mode procedure or a view) is executed with the privileges assigned to the object's owner. In the INVOKER security mode, a database object (for example, an invoker-mode procedure or a user-defined function (UDF)) is executed with the privileges assigned to the user who calls it. For more information about authorization checks on objects with dependencies, see the *Object Privileges* section in the *SAP HANA Security Guide*, which is listed in *Related Information* below.

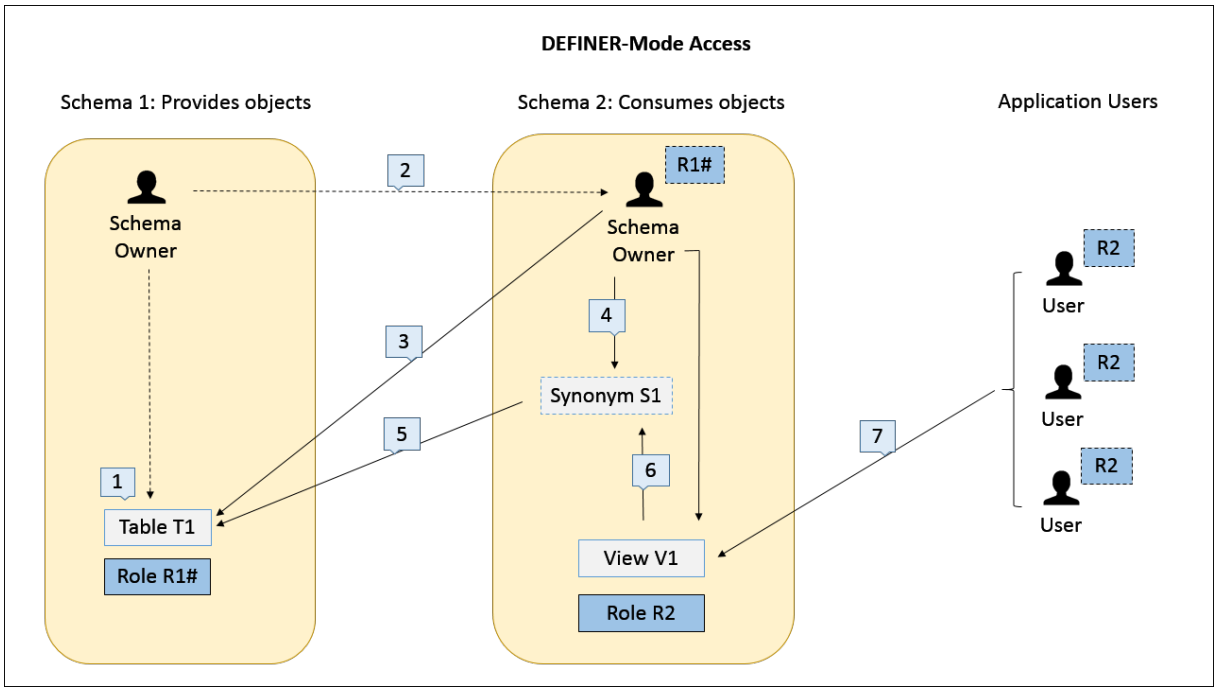
It is highly recommended to define object privileges in roles which can then be used to assign privileges for the target base object. This is particularly important for scenarios where the target base object is removed for some reason.

If a synonym's target base object (for example, a table or a view) is dropped and recreated, the corresponding object privilege has to be granted individually again to all referenced users. However, in most scenarios, the users consuming the target base object (indirectly via the synonym) are not known.

By using roles that include the object privilege, it is only necessary to reassign the object privilege to the respective role. All referencing users automatically receive the object privilege via their assigned roles.

Definer-Mode Access

The following diagram illustrates the users and privileges that are involved when setting up and using synonyms to enable "DEFINER"-mode access to database objects in an external schema (or another container) in XS advanced:



The various roles illustrated in the diagram are described in the following list:

- `Role R1#`
  - `SELECT` privilege on `Table T1` with `GRANT OPTION`
    Assigned to the owner of the schema that contains the consuming objects, for example, the synonym S1
- `Role R2`
  - `SELECT` privilege on `View V1`
    Assigned to the user of the application. This user is then able to access to the objects exposed by `Role R2`

> → Tip
>
> In the default "DEFINER" mode, a database object is executed with the privileges assigned to the object's owner; that is, the user who defines the object.

The following table provides more detailed information concerning the process of setting up and using synonyms in XS advanced:

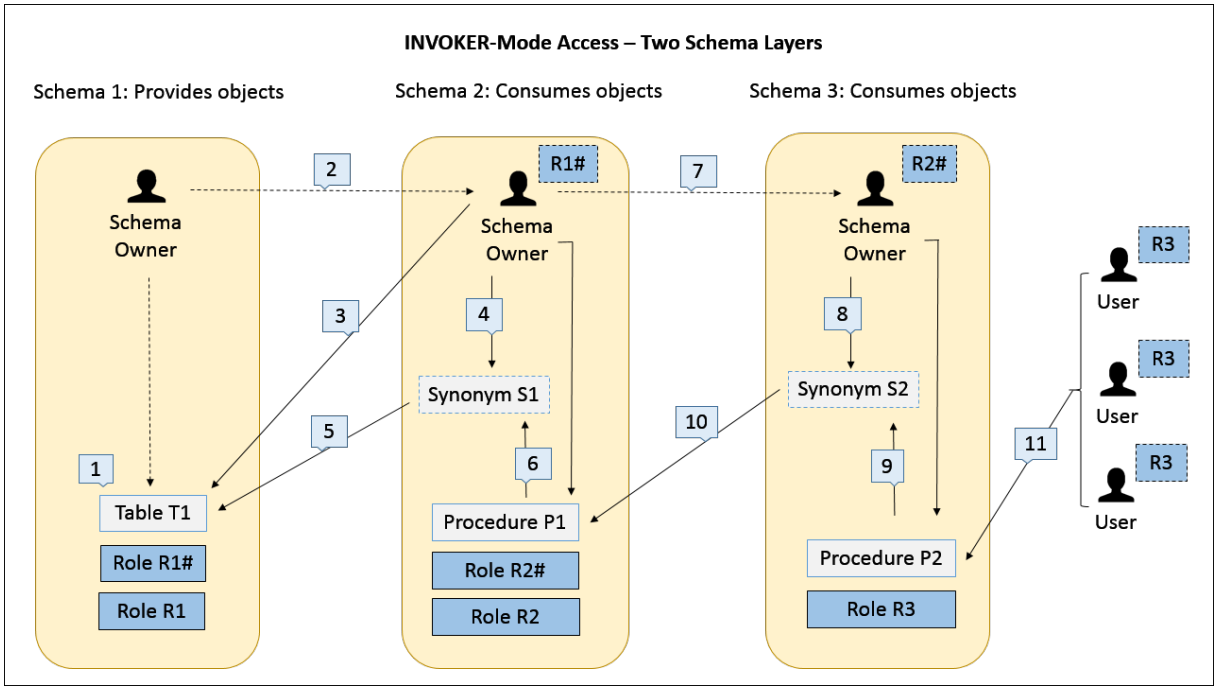
Synonym Users and Privileges for DEFINER-Mode Access

Step	Action	Comments
1	Create base target objects "`Table T1`" and role "`Role R1#`".	Synonyms can be created for other database objects, too, for example: tables, views, procedures, table functions, scalar functions, and sequences. `Role R1#` assigns the `SELECT` privilege on `Table T1` with `GRANT OPTION`
2	Grant privileges to "`Schema2`" owner.	Use `RoleR1#.hdbrole` to assign the required privileges. → Tip The "Schema Owner" in "`Schema2`" is the `"object_owner"` in the `hdbgrants` file.
3	Ensure the owner of "`Schema2`" has access to "`Table T1`"	The owner requires `SELECT` privilege on `Table T1` with `GRANT OPTION` which is defined in `Role R1#`.
4	Create "`Synonym S1`" (and role "`Role R2`").	When creating a synonym, the `SELECT` privilege is required for the DEFINER-mode access to the table, view, or sequence referenced as a base object. `Role R2` defines the privileges required by an application user to access the `View V1`.
5	Use "`Synonym S1`" to access "`Table T1`".	To access objects within an HDI-container, only private (i.e. schema-local) synonyms can be used.
6	Create a view to access "`Synonym S1`".	"`View V1`" references "`Synonym S1`".

Step	Action	Comments
7	Application users can access "`Table T1`" in "`Schema 1`" via "`View V1`" in "`Schema 2`"	Requires the `SELECT` privilege on "`View V1`"; this is defined in `Role R2`. `SELECT` privilege on `Table T1` is defined in `Role R1#`) → Tip User access to "`View V1`" is enabled using the technical user `application_user` (with the global `access_role`).

Invoker-Mode Access

The following diagram illustrates the users and privileges that are involved when using synonyms to enable "INVOKER"-mode access to database objects in an external schema (or another container) in XS advanced:



The various roles illustrated in the diagram for "INVOKER"-mode access are described in the following list:

- `Role R1`
  Assigned to the user of the application that consumes the underlying target objects
  - `SELECT` privilege on `Table T1`
- `Role R1#`
  Assigned to the owner of schema 2 which contains the consuming objects procedure P1 and synonym S1
  - `SELECT` privilege `with GRANT OPTION` on `Table T1`

- `Role R2`
  Assigned to the user of the application that consumes the underlying target objects
  - `EXECUTE` privilege on `Procedure P1`
  - The privileges defined in `Role R1` (`INVOKER` mode only)
- `Role R2#`
  Assigned to the owner of schema 3 which contains the consuming objects
  - `EXECUTE` privilege `with GRANT OPTION` on `Procedure P1`
  - The privileges defined in `Role R1#` (`INVOKER` mode only)
- `Role R3`
  Assigned to the user of the application that consumes the underlying target objects
  - `EXECUTE` privilege on `Procedure P2`
  - The privileges defined in `Role R2` (`INVOKER` mode only)
  - The privileges defined in `Role R1` (`INVOKER` mode only); this is included in Role R2

> → Tip
>
> In "INVOKER" mode, a database object (for example, an invoker-mode procedure or a user-defined function (UDF)) is executed with the privileges assigned to the user who starts it.

The following table provides more detailed information concerning the process of setting up and using multiple synonyms to enable access to INVOKER-mode database objects in XS advanced:

Synonym Users and Privileges for INVOKER-Mode Access

Step	Action	Comments

1	Create base target objects "`Table T1`" as well as "`Role R1`" and "`Role R1#`".	Synonyms can be created for other database objects, too, for example: tables, views, procedures, table functions, scalar functions, and sequences.
		`Role R1` assigns the `SELECT` privilege on `Table T1`
		`Role R1#` assigns the `SELECT` privilege on `Table T1` with `GRANT OPTION`

2	Assign access privileges to the owner of "`Schema2`".	Use `RoleR1#.hdbrole` to assign the required privileges. → Tip The "Schema Owner" in "`Schema2`" is the `"object_owner"` in the `hdbgrants` file.
3	Ensure the owner of "`Schema2`" has access to "`Table T1`"	`Role R1#` includes the `SELECT` privilege `WITH GRANT OPTION` on `Table T1`.

Step	Action	Comments
4	Create "`Synonym S1`", "`Role R2`", and "`Role R2#`".	"`Role R2`" defines the `EXECUTE` privilege on `Procedure P1` and includes a reference to the privileges defined in `Role R1` (`INVOKER` mode only) "`Role R2#`" defines the `EXECUTE` privilege `with GRANT OPTION` on `Procedure P1`and also includes a reference to the privileges defined in `Role R1#` (INVOKER mode only)
5	Use "`Synonym S1`" to access "`Table T1`".	When creating a synonym, the `SELECT` privilege is required for the table, view, or sequence referenced as a base object; for procedures and functions that are referenced as a base object, the `EXECUTE` privilege is required. **! Restriction** To access objects within an HDI-container, only private (schema-local) synonyms can be used.
6	Create an INVOKER-mode procedure "`Procedure P1`" to access "`Synonym S1`".	"`Procedure P1`" references "`Synonym S1`"
7	Assign access privileges to the owner of "`Schema 3`". **→ Tip** The "Schema Owner" in "`Schema3`" is the `"object_owner"` in the `hdbgrants` file; the user(s) in "`Schema3`" are the technical user `"application_user"`.	Use a database role object, for example, role `RoleR2.hdbrole`. **i Note** The role definition `RoleR2.hdbrole` must ensure that the owner of Schema 3 has the grant option and can assign privileges for access to "`Procedure P1`" in Schema 2.
8	Create "`Synonym S2`" and "`Role R3`".	`Synonym S2` provides access to `Procedure P1` which calls `Synonym S1`. `Role R3` defines the privileges required for `INVOKER` mode access to `Procedure P2`. `Role R3` must also include a reference to `Role R2`, which in turn includes `Role R1`.
9	Create an INVOKER-mode procedure "`Procedure P2`" to access "`Synonym S2`".	As explained in a previous step, a synonym cannot be used as the base object for another synonym.

Step	Action	Comments
10	Enable access to base object `Table T1` from Schema 3.	Requires the following privileges: • Role R1: SELECT ON T1 Role R1#: SELECT ON T1 WITH GRANT OPTION • Role R2: EXECUTE ON P1 + Role R1 Role R2#: EXECUTE ON P1 WITH GRANT OPTION + Role R1# • Role R3: EXECUTE ON P2 + Role R2 + Role R1
11	Assign access privileges to the application users.	Access privileges are defined in `Role R3`. → Tip In the `hdbgrants` file, the user(s) in "`Schema3`" are the technical user `"application_user"`.

Related Information

Enable Access to Objects in a Remote Classic Schema [page 476]
Database Synonyms in XS Advanced [page 482]
Syntax Options in the hdbgrants File [page 497]
SAP HANA Security Guide (Authorization, Object Privileges)

5.6.1.3 Granting Roles and Privileges for Use with Synonyms

Assigning roles and privileges to object owners for applications that use synonyms to access external objects.

You can use the `.hdbgrants` configuration file to assign privileges to the owner of the synonym object and the application users (consumers) of the synonym's target objects in the same way as you would with the SQL `grant` statement.

‹≡› Sample Code

`SNWD-table.hdbgrants`

```
{
  "granting-service [page 491]": {
   "object_owner": {
      "schema_roles": [
       {
        "roles" : "EPM_XXX::RoleR1"
       }
      ]
    },
    "application_user": {
```

```
      "schema_roles": [
       {
        "roles" : "EPM_XXX::RoleR2"
       }
      ]
    }
  }
}
```

→ Tip

For information about configuring a corresponding `.hdbrevokes` file that you can use to revoke granted permissions, see *Permissions for Objects in HDI Containers* in *Related Information* below. If both files exist, the `.hdbrevokes` file is processed before the `.hdbgrants` file. Since neither the `.hdbgrants` file nor the `.hdbrevokes` file is an HDI artifact, no HDI plug-in configuration is required to deploy them.

On startup, the HDI Deployer looks for `.hdbgrants` files and processes the contents in the following manner:

1. For each grantor in the file, the HDI Deployer checks for the existence of a bound service with the specified name (`"granting-service"` in the code sample above)
2. Connects to the database with the credentials of the named service
3. Grants the specified privileges to the grantees.

ⓘ Note

If the `"schema"` key is omitted for a privilege, then the `"schema"` property from the grantor service is used instead. If the `"name"` field in a `"global_object_privileges"` key of type `REMOTE SOURCE` is omitted, then the grantor's remote property is used instead.

In the `.hdbgrants` file, the top-level keys defines the **name** of the bound services which "**grant**" the specified privileges (the "grantors"), for example, `"granting-service"` in the code example above. The second-level keys define the users to whom the privileges must be granted (the "grantees"), for example, `"object_owner"` and `"application_user"`. The `"object_owner"` is used to specify the HDI container's object owner, and `"application_user"` defines the application users who are bound to the application modules, for example, the Node.js module. The third-level keys, for example, `"roles"`, are used to define the set of privileges to grant; the privileges are defined in a structure that is similar to that used in a role definition file (`myRole.hdbrole`).

The Granting Service

The first entry in the `hdbgrants` file (`“granting-service”` in the code example above) is the name of a service instance in your current Cloud Foundry space; this service instance executes the `GRANT` statement. The name specified can be either the name of an HDI container instance or the name of a user-provided service and must also be specified in the `resources` section of the multi-target application's development descriptor (`mta.yaml`).

Defining the Granting Service in the Application Development Descriptor

If the HDI container needs a granting-service, then in addition to the service itself, the multi-target application's development descriptor (`mta.yaml`) must include the information the HDI Deployer needs to be able to find the privilege-granting service. The following information is required:

1. `TARGET_CONTAINER`
   For the container specified in the `db` module, add a `TARGET_CONTAINER` property for the service that corresponds to the container.
2. `modules`
   In the `modules` section, add a new `requires` entry for the granting-service.
3. `resources`
   In the `resources` section, add a new entry for the granting service.

The following example of an `mta.yaml` file shows the modifications listed above which are required to define the granting service (if the corresponding container needs such a service):

Sample Code

Defining the Granting Service in the `mta.yaml` File

```
schema-version: '2.0'
ID: granting-service-example
version: 0.0.1
modules:
  - name: db
    type: hdb
    path: db
    requires:
      - name: hdi-container
        properties:                                    // #1.
          TARGET_CONTAINER: ~{hdi-container-service}   // #1.

      - name: granting-service                         // #2.
resources:
  - name: hdi-container
    type: com.sap.xs.hdi-container
    properties:
      hdi-container-service: ${service-name}
  - name: granting-service                             // #3.
    type: org.cloudfoundry.existing-service            // #3.
```

Permitted Types of Granting Services

The HDI Deployer supports the following types of granting-services:

- `hdi`
  An HDI container with access to the container's `GRANT` APIs. The corresponding service can be bound to the `db/` module application. The HDI Deployer recognizes the bound service by its `hdi_user` value in the credentials section and calls the container's API procedures to grant the privileges from the `.hdbgrants` file.
- `sql`
  A technical database user with `GRANT` privileges for the required object privileges, roles, system privileges, etc.
  In this scenario, a 'user-provided service' must be created in the same space as the HDI container.
  The service needs to be set up with the permissions of a specified database user who can connect to the database and grant the privileges specified in the `.hdbgrants` files during application deployment.
  You can use the XS advanced command-line interface to create a user-provided (granting) service, as illustrated in the following example, where `xs cups` is an abbreviation for `xs create-user-provided-service`:

  ```
  xs cups grantor-service -p '{ "host": "host.acme.com", "port": "30015",
  "certificate": "<myCertificate>", "user": "<TARGET_USER>",
  ```

```
"password": "Grant_123", "schema": "<TARGET_SCHEMA>", "tags": [ "hana" ] }'
```

- `"host"/"port"`
  Required for the connection to the database: `port` is the SQL port of the SAP HANA index server
- `"certificate"`
  If the database is configured to only accept secure connections, then the granting-service requires an SSL certificate that must be included in the user-provided service, for example, using the `"certificate":""` parameter.
- `"user"/"password"`
  Connection details for a database user that has grant permissions for the objects in the schema.
- `"schema"`
  The database schema that contains the objects to which access is to be granted.
- `"type"`
  The type of the grantor-service mechanism; valid values are: `"hdi"`, `"sql"`, or `"procedure"`. If the type is not specified, then the type is selected automatically based on the following rule: if the field `"hdi_user"` is present, then the type is set to `hdi`, otherwise, the type is set to `sql`.

- `procedure`
  If the technical database user does not have `GRANT` privileges for the required object privileges, roles, system privileges, etc., but only `EXECUTE` privileges on a stored procedure which can grant the privileges, then the following prerequisites apply:
  - At the database level, a `GRANT` procedure must exist (or be visible) in the schema which is used in the user-provided service.
  - The technical database user must have `EXECUTE` privileges on the `GRANT` procedure.
  - The name of the `GRANT` procedure must be specified in the user-provided service in the `"procedure"` field, for example:
    `"procedure": "GRANT"`.
  - The schema name of the `GRANT` procedure can be specified in the user-provided service in the `"procedure_schema"` field, for example:
    `"procedure_schema": "A_SCHEMA"`.
  - In this particular scenario, the user-provided service must contain a `"type"` field with the value `"procedure"`.
- `ignore`
  Grant privileges have already been assigned at the database-level, so the HDI Deployer ignores the content of the `.hdbgrants` file.

The Object Owner

The `“object_owner”` property defines the roles to be assigned to the container's **object** owner. The name of the role differs according to whether the target object is in an XS advanced schema (application container) or a classic SAP schema (for example, ERP).

Sample Code

`.hdbgrants` File for Role Assignment in a Classic Schema

```
"object_owner": {
   "schema_roles": [
    {
```

```
      "roles" : "EPM_XXX::RoleR1"
     }
    ]
},
```

If the synonym's target database object is in an XS advanced application's HDI container, the owner of the object is the schema owner. HDI container schema owners have a special name that ends with the string "#OO" (for example, `ContainerOwner#OO`), the corresponding role must also have the suffix #, for example, `RoleR1#`, as illustrated in the following example:

Sample Code

`.hdbgrants` File for Role Assignment in an HDI Container Schema

```
"object_owner": {
   "schema_roles": [
    {
     "roles" : "RoleR1#"
    }
   ]
},
```

If a role name ends with "#", for example, `MyRole#`, the role definition (specified in an `hdbrole` artifact) is allowed to include "`WITH GRANT OPTION`" privileges as well as references to other roles whose names end with "#". The HDI container object owner needs most privileges "WITH GRANT OPTION".

The Application User

The "`application_user`" property defines the roles to be assigned to the user of the XS advanced application that consumes the defined synonym and requires access to the corresponding schemas containing the target objects referenced by the synonym. The following example shows how to assign `application_user` to the privileges defined in the externally created role `"EPM_XXX::RoleR1"`.

Sample Code

`SNWD-table.hdbgrants`

```
"application_user": {
  "schema_roles": [
   {
    "roles" : "EPM_XXX::RoleR1"
   }
  ]
}
```

→ Tip

"`application_user`" does not refer to a real user; it refers to the technical user associated with the global "`access_role`" required for access to the corresponding run-time container. The "`access_role`" is assigned a set of default permissions for the run-time schema, for example: `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `EXECUTE`, `CREATE TEMPORARY TABLE`, and `SELECT CDS METADATA`.

Schema Roles

The `schema_roles` property is used to specify one or more schema-local `“roles”` to users by means of a grantor service. In XS advanced, access to the synonym's target object is enabled by assigning a local HDI container role that grants the container's object owner access to the target object in the foreign schema. This role must also have the `privileges_with_grant_option`, and for this reason the role name must end with the hash (#) character, for example, `RoleR1#`. Assigning XS advanced roles to an HDI schema owner requires some changes to the `.hdbgrants` file, as illustrated in the following example:

Sample Code

`SNWD-table.hdbgrants` for HDI Schema (Container) Roles

```
{
  "EPM_XXX-table-grantor": {
   "object_owner": {
      "schema_roles": [
       {
        "roles" : "EPM_XXX::RoleR1#"
       }
      ]
    },
    "application_user": {
      "schema_roles": [
       {
        "roles" : "EPM_XXX::RoleR1"
       }
      ]
    }
  }
}
```

Due to naming conventions in HDI, and the fact that the container role must provide the `GRANT` privileges, the name of the role-definition assigned to the HDI container's object owner must have the suffix "#", for example, `“MyRole#”`.

If a role name ends with the hash character (#), for example, `MyRole#`, the role definition can include “with-grant-option” privileges as well as references to other roles whose names also end with #. A role whose name ends with # can only be granted by means of the HDI APIs to a container’s object owner, for example, `<container>#OO`.

Note

For `schema_roles`, `schema_privileges`, and `object_privileges`, you can specify a fixed schema name by using the optional `schema` property in the privilege(s) defined in the `.hdbgrants` file.

The `schema` defined in `schema_roles`, `schema_privileges`, or `object_privileges` overrides the schema defined in the grantor service. If no fixed `schema` is defined in the privilege, then the schema defined in the grantor service is used instead. For more information, see *Syntax Options in the hdbgrants File* in *Related Information* below.

Roles

Use the `"roles"` property to specify the role (or roles) to be assigned to the object owner (one per HDI container) and application user by the grantor service, as illustrated in the following example:

Sample Code

`SNWD-table.hdbgrants`

```
{
  "EPM_XXX-table-grantor": {
   "object_owner": {
      "schema_roles": [
       {
        "roles" : ["EPM_XXX::RoleR1#"]
       }
      ]
    },
    "application_user": {
      "schema_roles": [
       {
        "roles" : ["EPM_XXX::RoleR1"]
       }
      ]
    }
  }
}
```

The attribute `"roles"` is used to specify the name of the role that defines the privileges to assign to users. The role name must be a non-empty string that contains only permitted characters.

In the `.hdbgrants` file, the following types of role names are possible:

- `<RoleName>`
  If a role name does **not** end with the hash character (#), for example, `MyRole`, then only privileges without grant options can be included in the role as well as references to any only other roles whose names also do not end with "#".
- `<RoleName>#`
  If a role name ends with the hash character (#), for example, `MyRole#`, its corresponding role definition can include "with-grant-option" privileges as well as references to other roles whose names also end with #. A role whose name ends with # can only be granted by means of the HDI APIs to a container's object owner, for example, `<container>#OO`.

The roles specified in the `hdbgrants` file must already exist as objects in the database catalog. The corresponding role-definition must define the various privileges required for access to database objects, for example: system, schema, object, and analytic privileges. In XS advanced, you can define a user role with an `.hdbrole` artifact; in XS classic, you can use alternative methods to define access privileges to database objects, for example, the SQL command `create role "EPM_XXX::external access`. In addition, you will also have to grant the `SELECT` privilege manually to each base objects referenced by the synonym.

⚠ Caution

To prevent unnecessary access to the schema via "definer-mode" objects (procedures, views), it is recommended to assign different roles to the object owner and the application user. The role for the object owner should provide `GRANT` options; the role for the application user should provide privileges for "invoker-mode" objects (for example, invoker-mode procedures).

Related Information

Enable Access to Objects in a Remote Classic Schema [page 476]
Syntax Options in the hdbgrants File [page 497]
Permissions for Objects in HDI-Container [page 274]

5.6.1.4 Syntax Options in the hdbgrants File

Assign the privileges required by users to access objects in the target schema.

The `.hdbgrants` configuration file enables you to assign privileges to the owner of the synonym object and the application users (consumers) of the synonym's target objects in the same way as you would with the SQL `grant` statement.

> **Note**
>
> For information about configuring a corresponding `.hdbrevokes` file that you can use to revoke granted privileges, see *Permissions for Objects in HDI Containers* in *Related Information* below. If both files exist, the `.hdbrevokes` file is processed before the `.hdbgrants` file. Since neither the `.hdbgrants` file nor the `.hdbrevokes` file is not an HDI artifact, no HDI plug-in configuration is required.

> **Sample Code**
>
> `external-access.hdbgrants`

```
{
  "external-access": {
    "object_owner": {
      "system_privileges" : [ "SYSTEM PRIVILEGE 1", "SYSTEM PRIVILEGE 2" ]
      "global_roles" : [
        {
          "roles" : [ "GLOBAL_ROLE_1", "GLOBAL_ROLE_2" ]
          "roles_with_admin_option" : [ "GLOBAL_ROLE_3", "GLOBAL_ROLE_4" ]
        }
      ],
      "schema_privileges" : [
        {
          "schema" : "<schema_name>"   // optional; override schema defined
in grantor service
          "privileges" : [ "INSERT", "UPDATE" ],
          "privileges_with_grant_option" : [ "SELECT" ]
        }
      ],
      "schema_roles" : [
        {
          "schema" : "<schema_name>"   // optional; override schema defined
in grantor service
          "roles" : [ "SCHEMA_ROLE_1", "SCHEMA_ROLE_2" ]
          "roles_with_admin_option" : [ "SCHEMA_ROLE_3", "SCHEMA_ROLE_4" ]
        }
      ],
      "object_privileges" : [
        {
          "schema" : "<schema_name>"   // optional; override schema defined
in grantor service
          "name": "AN_OBJECT",
          "privileges": [ "INSERT", "UPDATE" ],
```

```
          "privileges_with_grant_option" : [ "SELECT" ]
        }
      ],
      "global_object_privileges" : [
        {
          "name" : "A_REMOTE_SOURCE",
          "type" : "REMOTE SOURCE",
          "privileges" : [ "CREATE VIRTUAL TABLE" ],
          "privileges_with_grant_option" : [ "CREATE VIRTUAL PROCEDURE" ]
        }
      ]
    },
    "application_user": {
      "system_privileges": [...],
      "global_roles": [{...}],
      "schema_privileges": [{...}],
      "schema_roles": [{...}],
      "object_privileges": [{...}],
      "global_object_privileges": [{...}]
    }
  }
}
```

In the `hdbgrants` file, the first entry (`“external-access”` in the example above) is the name of a service instance in your current Cloud Foundry space, which executes the `GRANT` statement. This name can be either an HDI container instance or a user-provided service, which must be specified in the `resources` section of the application project's development-descriptor (`mta.yaml`).

> **i** Note
>
> For services created by the Instance Manager, you need to create a user-provided service with the credentials required for service access. This is because container instances created by the Instance Manager are not visible as "real" service instances; they are hidden behind the Instance-Manager service instance.

object_owner

Defines the roles and privileges to be assigned to the owner of the specified container's **object**. In the `“object_owner”` section, you can also set the following parameters:

- `system_privileges`
- `global_roles`

  > **i** Note
  >
  > `global_roles` supercedes `roles`, which is still supported for backward compatibility or compatibility with `.hdbrole`.

- `schema_privileges`
- `schema_roles`

  > **i** Note
  >
  > `schema_roles` supercedes `container_roles`, which is still supported for backward compatibility.

- `object_privileges`
- `global_object_privileges`

> **i Note**
>
> For `schema_roles`, `schema_privileges`, and `object_privileges`, you can specify a fixed schema name by using the optional `schema` property in the privilege(s) defined in the `.hdbgrants` file.

The `schema` defined in `schema_roles`, `schema_privileges`, or `object_privileges` overrides the schema defined in the grantor service. If no fixed `schema` is defined in the privilege, then the schema defined in the grantor service is used instead. For more information, see *Syntax Options in the hdbgrants File* in *Related Information* below.

The name of the role differs according to whether the target object is in an XS advanced schema (application container) or a classic SAP schema (for example, ERP). The names of the privileges can include `system`, `schema`, and `object` (and `global_oject`) privileges, as illustrated in the following example:

> **Sample Code**
>
> `.hdbgrants` File for Role Assignment in a Classic Schema

```
{
  "external_access": {
    "object_owner": {
      "system_privileges": [...],
      "global_roles": [{...}],
      "schema_privileges": [{...}],
      "schema_roles": [{...}],
      "object_privileges": [{...}],
      "global_object_privileges": [{...}]
    }
  }
}
```

system_privileges

You can use the `"system_privileges"` parameter to assign one or more system permissions to the object owner as illustrated in the following example:

> **Sample Code**
>
> System Privileges for the Object Owner in the `hdbgrants` File

```
{
  "external_access": {
    "object_owner" : {
      "system_privileges" : ["System Privilege 1, System Privilege 2],
      ...
    }
  }
}
```

global_roles

Use the `"global_roles"` property to specify the role (or roles) to be assigned to the object owner (one per HDI container) and application user by the grantor service, as illustrated in the following example:

> **i** Note
>
> `global_roles` supercedes `roles`, which is still supported for backward compatibility or compatibility with `.hdbrole`. Global roles are not supported when referencing an HDI container service; use a **container-local** role where possible, instead.

> Sample Code
>
> `SNWD-table.hdbgrants` for Global Roles

```
{
  "EPM_XXX-table-grantor": {
   "object_owner": {
      "global_roles": [
        {
          "roles" : ["Global_Role1", "Global_Role2"],
          "roles_with_admin_option" : ["Global_Role3", "Global_Role4"]
        }
      ]
    },
    "application_user": {
      "global_roles": [
        {
          "roles" : ["Global_Role1", "Global_Role2"],
          "roles_with_admin_option" : ["Global_Role3", "Global_Role4"]
        }
      ]
    },
  }
}
```

The property `"roles"` is used to specify the name of the role that defines the privileges to assign to users. The role name must be a non-empty string that contains only permitted characters.

In the `.hdbgrants` file, the following types of role names are possible:

- `<RoleName>`
  If a role name does **not** end with the hash character (#), for example, `MyRole`, then only privileges without grant options can be included in the role as well as references to any only other roles whose names also do not end with "#".
- `<RoleName>#`
  If a role name ends with the hash character (#), for example, `MyRole#`, it's corresponding role definition can include "with-grant-option" privileges as well as references to other roles whose names also end with #. A role whose name ends with # can only be granted by means of the HDI APIs to a container's object owner, for example, `<container>#OO`.

The roles specified in the `hdbgrants` file must already exist as objects in the database catalog. The corresponding role-definition must define the various privileges required for access to database objects, for example: system, schema, object, and analytic privileges. In XS advanced, you can define a user role with an `.hdbrole` artifact; in XS classic, you can use alternative methods to define access privileges to database objects, for example, the SQL command `create role "EPM_XXX::external access`. In addition, you will also have to grant the `SELECT` privilege manually to each base objects referenced by the synonym.

> ⚠ Caution
>
> To prevent unnecessary access to the schema via “definer-mode” objects (procedures, views), it is recommended to assign different roles to the object owner and the application user. The role for the object owner should provide `GRANT` options; the role for the application user should provide privileges for “invoker-mode” objects (for example, invoker-mode procedures). The roles themselves are defined in `.hdbrole` artifacts.

schema_privileges

You can use the `“schema_privileges”` parameter to assign permissions for the object owner in the target schema, for example, `INSERT` or `UPDATE` for `“privileges”`. You can also add the grant option to the system privilege, for example, `privileges_with_grant_option` on `SELECT`.

> → Tip
>
> For `schema_privileges`, you can specify a fixed schema name by using the optional `schema` property in the privilege(s) defined in the `.hdbgrants` file.

The `schema` defined in the `schema_privileges` property overrides the schema defined in the grantor service. If no `schema` is defined in `schema_privileges` in the `.hdbgrants` file, then the schema defined in the grantor **service** is used instead.

> ⁖ Sample Code
>
> Schema Privileges for the Object Owner in the `hdbgrants` File

```
{
  "external_access": {
    "object_owner" : {
      "schema_privileges" : [
        {
          "schema" : "<schema_name>"          // optional
          "privileges" : [ "INSERT", "UPDATE" ],
          "privileges_with_grant_option" : [ "SELECT" ]
        }
      ],
      ...
    }
  }
}
```

schema_roles

Use `“schema_roles”` to grant schema-local roles to users by means of a grantor service. Multiple roles can be specified in an array, for example, `“roles” : ["Schema_Role_1", "Schema_Role_1"]`. You can also specify additional roles that include administrator options, for example, `“roles_with_admin_option” : ["Schema_Role_3", "Schema_Role_4"]`.

Note

`schema_roles` supercedes `container_roles` and works for both normal schemas and HDI containers. The `container_roles` property is still supported for backward compatibility or compatibility with `.hdbrole`.

If the grantor service is bound to a normal database user, you can assign roles with or without administrator options, as illustrated in the following example:

Sample Code

`SNWD-table.hdbgrants` for HDI Schema (Container) Roles

```
{
  "EPM_XXX-table-grantor": {
   "object_owner": {
      "schema_roles": [
        {
          "schema" : "<schema_name>"          // optional
          "roles" : ["Schema_RoleR1", "Schema_RoleR2"],
          "roles_with_admin_option" : ["Schema_RoleR3", "Schema_RoleR4"]
        }
      ]
    }
  }
}
```

For `schema_roles`, you can specify a fixed schema name by using the optional `schema` property in the privilege(s) defined in the `.hdbgrants` file. The `schema` defined in the `schema_roles` property overrides the schema defined in the grantor service. If no `schema` is defined in `schema_roles` in the `.hdbgrants` file, then the schema defined in the grantor service is used instead.

Restriction

For container roles, the `roles-with-admin-option` is not supported.

If the grantor service is bound to an HDI container, then it is not possible to use the `“roles_with_admin_option”`. In the following example, the role assignment works for both normal database-user bindings and bindings to an HDI container, too:

Sample Code

`SNWD-table.hdbgrants` for HDI Schema (Container) Roles

```
{
  "EPM_XXX-table-grantor": {
   "object_owner": {
      "schema_roles": [
        {
          "roles" : ["Schema_RoleR1", "Schema_RoleR2"]
        }
      ]
    }
  }
}
```

> **i** Note
>
> Although the parameter `schema_roles` replaces `container_roles`, `container_roles` will continue to be supported for backward compatibility.

object_privileges

For each individual object, you can assign privileges to the object owner: either with the grant option (for example, `SELECT`) or **without** (for example, `INSERT`, `UPDATE`), as illustrated in the following example:

> **‹≡›** Sample Code
>
> Object Privileges in the `hdbgrants` File

```
{
  "external_access": {
    "object_owner" : {
      "object_privileges" : [
        {
          "schema" : "<schema_name>"          // optional
          "name": "AN_OBJECT",
          "privileges": [ "INSERT", "UPDATE" ],
          "privileges_with_grant_option" : [ "SELECT" ]
        }
      ],
    }
  }
}
```

> **→** Tip
>
> For `object_privileges`, it is possible to specify a fixed schema name by using the optional `schema` property in the privilege(s) defined in the `.hdbgrants` file.

The `schema` defined in the `object_privileges` property overrides the schema defined in the grantor service. If no fixed `schema` is defined in the object privilege, then the schema defined in the grantor service is used instead.

Object privileges are not supported when referencing an HDI container service. Instead, deploy a role with the desired object privileges to the referenced HDI container and then grant the deployed role in the `.hdbgrants` file using the `"schema_roles"` property.

global_object_privileges

For each individual object, you can assign global privileges to the object owner: either with the grant option (for example, `SELECT`) or **without** (for example, `INSERT`, `UPDATE`), as illustrated in the following example:

Sample Code

Global Object Privileges in the `hdbgrants` File

```
{
  "external_access": {
    "object_owner" :
      "global_object_privileges" : [
        {
          "name" : "A_REMOTE_SOURCE",
          "type" : "REMOTE SOURCE",
          "privileges" : [ "CREATE VIRTUAL TABLE" ],
          "privileges_with_grant_option" : [ "CREATE VIRTUAL PROCEDURE" ]
        }
      ]
    }
  }
}
```

application_user

Defines the roles and permissions to be assigned to the user of the XS advanced application that consumes the defined synonym and requires access to the corresponding schemas containing the target objects referenced by the synonyms. The following example shows how to assign `application_user` to the privileges defined in the externally created role `"EPM_XXX::RoleR1"`.

Tip

`“application_user”` does not refer to a real user; it refers to the technical user associated with the global “`access_role`” required for access to the corresponding run-time container. The “`access_role`” is assigned a set of default permissions for the run-time schema, for example: `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `EXECUTE`, `CREATE TEMPORARY TABLE`, and `SELECT CDS METADATA`.

- `system_privileges`
- `global_roles`

  ### Note

  `global_roles` supercedes `roles`, which is still supported for backward compatibility or compatibility with `.hdbrole`.
- `schema_privileges`
- `schema_roles`

  ### Note

  `schema_roles` supercedes `container_roles`, which is still supported for backward compatibility or compatibility with `.hdbrole`.

- `object_privileges`
- `global_object_privileges`

> Sample Code
>
> `.hdbgrants` File for Role Assignment in a Classic Schema

```
"application_user": {
  "system_privileges": [...],
  "global_roles": [{...}],
  "schema_privileges": [{...}],
  "schema_roles": [{...}],
  "object_privileges": [{...}],
  "global_object_privileges": [{...}]
},
```

Related Information

Using Synonyms to Access External Schemas and Objects in XS Advanced [page 476]
Database Synonyms in XS Advanced [page 482]
Roles (.hdbrole and .hdbroleconfig)
Permissions for Objects in HDI-Container [page 274]

5.6.2 Enable Access to Objects in Another HDI Container

Use a synonym to enable access to another HDI container.

Prerequisites

To complete the steps described in this task, the following prerequisites apply:

- You have access to the XS advanced run-time environment
- You have access to SAP Web IDE for SAP HANA
- You have access to the XS command-line interface client

Context

You can use a synonym in one XS advanced application to enable access to database objects in a different XS advanced application; that is, objects in an HDI container that belongs to another XS advanced application. You need to ensure access to the external HDI container and specify the type of privileges (SELECT, EXECUTE) required on the target object (table, view, etc.).

Procedure

1. Locate the target object in the external schema to which the XS advanced synonym will point.

   The target objects can be tables, views, functions, procedures, and so on; the target schema is another HDI container.
2. Create the XS advanced role definitions that enable access to the target database object.

   In this scenario, you create two roles:

   > **i** Note
   >
   > Both these roles must be created in the HDI schema that contains the target database object, for example, `Table_T1`.

   a. Create a role for the schema owner of the XS advanced application that contains the synonym.

      The role Role R1# (`Role_R1G.hdbrole`) defines the privileges required for external access to a specific target object (with grant option); the role must be assigned to the user who needs access to the schema where the target object `Table_T1` is located. In this case, the access role is assigned to the owner of the schema containing the synonym that points to the target table.

      > Sample Code
      >
      > Role Definition `Role_R1G.hdbrole` for HDI Schema Owner

      ```
      {
        "role": {
          "name": "Role_R1G#",
          "object_privileges": [
            {
              "name":"<Table_T1>",
              "type":"TABLE",
              "privileges_with_grant_option":[ "SELECT" ]
            }
          ]
        }
      }
      ```

      > → Tip
      >
      > The hash character (#) at the end of the role name signifies that the role is intended for assignment to a schema owner (for example, `<Container>#OO`) and that the role must include the "`with grant option`" privilege.

   b. Create a role for the user of the XS advanced application that contains the synonym.

      The role `Role_R1.hdbrole` defines the privileges required by the application user who needs access to the target object `Table_T1` by means of a synonym; the "`SELECT`" privilege is required for the target object (**without** grant option). The role must be assigned to the user of the XS advanced application which contains the synonym that needs access to the target object `Table_T1`.

      > **i** Note
      >
      > `Role_R1` is assigned to the XS advanced application user by including a reference to it in `RoleR2`, which you define in a later step. `Role_R2` is located in the same HDI schema as the synonym.

Sample Code

Role Definition `Role_R1.hdbrole` for XS Advanced Application User

```
{
  "role": {
    "name": "Role_R1",
    "object_privileges": [
      {
        "name":"<Table_T1>",
        "type":"TABLE",
        "privileges":[ "SELECT" ]
      }
    ]
  }
}
```

3. Grant access to the schema containing the synonym's target object (for example, `Table_T1`).

   The owner of the schema containing the synonym requires `SELECT` privileges on the target object to which the synonym points. Access for the schema owner can be enabled in user roles which are then referenced with the `"container_roles"` property in an `.hdbgrants` file, as illustrated in the following example:

Sample Code

Access-Granting Configuration File (`myApp/db/cfg/Synonym_S1-table.hdbgrants`)

```
{
  "EPM_XXX-table-grantor": {
     "object_owner": {
           "container_roles": [
               "Role_R1G#"
           ]
     },
     "application_user": {
           "container_roles": [
               "Role_R1"
           ]
     }
  }
}
```

Caution

To prevent unwanted schema access, for example, via views, it is recommended to assign different roles to the object owner and the application user.

4. Create the synonym(s).

   You can create the synonym design-time object in a subfolder of your XS advanced application's database module, for example in `/<MyApp>/db/src/synonyms/`. In this example, we name the synonym definition `Synonym_S1.hdbsynonym`, which contains multiple synonyms referencing tables (`Table_T1`, `Table_T2`, and `Table_T3`), as illustrated in the following example:

Sample Code

XS Advanced Synonym-Definition File (`/MyApp/db/src/synonyms/Synonym_S1.hdbsynonym`)

```
{
  "Table_T1": {}
```

```
}
```

> **Note**
>
> In this example, the synonym configuration is not included in the synonym file; it is moved to the synonym's corresponding configuration file `SynonymS1.hdbsynonymconfig`, as described in the following step.

5. Create a synonym configuration file.

   In XS advanced, the synonym configuration can also be defined in a `.hdbsynonymconfig` file, for example, `myApp/db/cfg/Synonym_S1.hdbsynonymconfig`.

   > **Note**
   >
   > The XS classic `.hdbsynonymtemplate` files can also be used; they are converted to the `.hdbsynonymconfig` file format required by XS advanced at deployment time.

6. Define details of the synonym configuration.

   Rather than defining a schema and a concrete schema name, you can use `“schema.configure”` to specify a path to a service name. The path expression is replaced at deployment time with the name of the schema of the referenced service. For example, in the following code example, `“schema.configure”` is replaced with schema used by the grantor service.

   > **Sample Code**
   >
   > Synonym Configuration File (`myApp/db/cfg/Synonym_S1.hdbsynonymconfig`)

```
{
  "Table_T1": {
    "target": {
      "object": "Table_T1",
      "schema.configure" : "<target-hdi-container>/schema"
    }
  },
  […]
}
```

7. Define the required HDI plug-in configuration for the synonym.

   Add the XS advanced application project's central HDI plugin-configuration file (`.hdbconfig`) to the database module's “`cfg/`” folder. You can copy the HDI plugin-configuration file from `myapp/db/src/.hdbconfig` to `myapp/db/cfg/.hdbconfig`.

   The `.hdiconfig` configuration file should include the following entries:

   > **Sample Code**
   >
   > HDI Plug-in Configuration File (`.hdiconfig`)

```
"hdbsynonym" : {
   "plugin_name" : "com.sap.hana.di.synonym",
   "plugin_version": "2.0.0.0"
},
"hdbsynonymconfig" : {
   "plugin_name" : "com.sap.hana.di.synonym.config",
   "plugin_version": "2.0.0.0"
```

```
}
```

→ Tip

The `"plugin_version": "2.0.0.0"` property is optional; from SAP HANA 2.0, the version of all plugins shipped with SAP HANA is the same as (and equal to) the SAP HANA version.

8. Update the XS advanced application's development descriptor (`mta.yaml`).

   Add references to "`<target-hdi-container>`" definition in the development descriptor for the application that needs to use the synonym to access the target objects, as illustrated in the example below.

‹≡› Sample Code

Application Development Descriptor (`MyApp/mta.yaml`)

```
modules:
  - name: db
    type: hdb
    path: db
    requires:
      - name: hdi-container
        properties:
          TARGET_CONTAINER: ~{hdi-container-service}
      - name: EPM_XXX-table-grantor
        group: SERVICE_REPLACEMENTS
        properties:
          key: EPM_log-table-grantor
          service: ~{EPM_Synonym_S1-table-grantor-service}
resources:
  - name: hdi-container
    type: com.sap.xs.hdi-container
    properties:
      hdi-container-service: ${service-name}

  - name: EPM_XXX-table-grantor
    type: org.cloudfoundry.existing-service
    properties:
      EPM_Synonym_S1-table-grantor-service: ${service-name}
    parameters:
      service-name: <HDI_Container_Name>-hdi-container
```

9. Consume the synonym you created.

   One way to consume synonyms from inside the XS advanced application's schema is to use a view (for example, `View_V1`) on the target table.

‹≡› Sample Code

SQL View `/myApp/db/src/View_V1.hdbview`

```
VIEW "View_V1" AS SELECT * FROM Table_T1
```

10. Create a role definition that enables access to the view consuming `Synonym_S1`.

→ Tip

The view is not **required** for the creation of a synonym; it is used here to test "consumption" of the synonym `Synonym_S1`.

The role definition (for example, `Role_R2.hdbrole`) must be assigned to the application users who need access to the view (in this example, `View_V1`). The XS advanced application user already has access to the view through the application container's `access_role`; the role `Role_R2.hdbrole` is required by users outside of the application.

Sample Code

Role Definition `myApp/db/src/Role_R2.hdbrole`

```
{
  "role": {
    "name": "Role_R2",
    "object_privileges": [
      {
        "name":"View_V1",
        "type":"VIEW",
        "privileges":[ "SELECT" ]
      }
    ]
  }
}
```

11. Build both XS advanced application projects: the application that contains the target table (`Table_T1`) and the application that contains `Synonym_S1` and `View_V1`.
12. Check the database catalogs to ensure the appropriate objects exist and are accessible with the synonyms you created.

Related Information

Using Synonyms to Access External Schemas and Objects in XS Advanced [page 476]
Database Synonyms in XS Advanced [page 482]
Syntax Options in the hdbgrants File [page 497]
Users, Privileges, and Schemas [page 484]

5.7 Create Virtual Tables

Virtual tables enable you to access objects in other databases without having to replicate in SAP HANA.

Prerequisites

You must already have created a remote source.

Context

The SAP Web IDE supports the creation of virtual tables using either a code editor or a built-in Wizard. The Wizard is designed to help those users who are not sure of the syntax required to create virtual tables in the code editor. After creating the virtual tables, you can use them as data source.

Procedure

1. Start SAP Web IDE in a Web browser.
2. If you want to create a new project for the virtual table, do the following:
   a. In SAP Web IDE, choose *File* > *New* > *Project from Template*.
   b. Choose the project template type.
   c. Type a name for the new project (for example, **myApp**) and choose *Next* to confirm.
   d. Specify details of the new project and choose *Next* to confirm.
   e. Create the new project; choose *Finish*.
3. Select the *SAP HANA Database Module* in which you want to create the virtual table.

   > → Tip
   >
   > To create a database module, right-click the root folder of your new project and, in the context menu, choose *New* > *SAP HANA Database Module*.

4. Browse to the *src* folder, right-click it and choose *New* > *Virtual Table*.
5. In the *Create Virtual Object* wizard, select a remote source and click *Next*.
6. Use the filter parameters to search for one or more remote objects you want to create a virtual table from and click *Next*.
7. Enter a name for the virtual object and click *Finish*.

   Use the naming format `namespace::virtual_table_name`.
8. Choose *Save* in the menu bar to save your calculation view.
9. Build an SAP HANA Database Module.

   The build process uses the design-time database artifacts to generate the corresponding actual objects in the database catalog.
   a. From the module context menu, choose *Build*.

5.8 Setting Up the Analytic Model

Modeling refers to an activity of refining or slicing data in database tables by creating analytic models or views to depict a business scenario.

The modeling process involves the simulation of entities, such as CUSTOMER, PRODUCT, and SALES, and relationships between them. You can use these related entities (analytic models) in analytics applications such as SAP BusinessObjects Explorer and Microsoft Office for reporting use cases.

Analytic models or views use various combinations of content data (that is, non-metadata) to model a business use case. Content data can be classified as follows:

- Attribute: Descriptive data, such as customer ID, city, and country.
- Measure: Quantifiable data, such as revenue, quantity sold and counters.

SAP Web IDE for SAP HANA offers data modelers and other user personas the modeling tools to work with SAP HANA content repository and SAP HANA database catalog objects.

Related Information

Create Calculation Views [page 512]
Preview Calculation View Output [page 518]
Defining Data Access Privileges [page 520]

5.8.1 Create Calculation Views

Use a graphical editor to create calculation views that depict a complex business scenario.

Context

Calculation views can be used to combine data from various sources. You can combine multiple transaction tables in a calculation view. You can also create calculation views to include layers of calculation logic.

Procedure

To create a calculation view:

1. Start SAP Web IDE in a Web browser.
2. To create a new project for the calculation view, do the following:
   a. In the SAP Web IDE, choose *File* > *New* > *Project from Template*.
   b. Choose the project template type.

      Currently, there is only one type of project template available, namely: *Multi-Target Application Project*. Select *Multi-Target Application Project* and choose *Next*.
   c. Type a name for the new MTA project (for example, **myApp** and choose *Next* to confirm).
   d. Specify details of the new MTA project and choose *Next* to confirm.
   e. Create the new MTA project; choose *Finish*.
3. Select the *SAP HANA Database Module* in which you want to create the calculation view.

> → Tip
>
> If you do not already have a database module, right-click the root folder of your new MTA project and, in the context menu, choose *New* > *SAP HANA Database Module*.

4. Navigate to the *src* folder, right-click and choose *New* > *Calculation View*.
5. Enter the details for the new calculation view.
   a. In the *Name* field, enter the name of the calculation view.
   b. From *Data Category*, select a value.
6. Choose *Create*.

   The calculation view editor is opened. Depending on the data category that you selected when you created the calculation view, the default view node can be an aggregation or a projection view node. If you are creating a calculation view with star join, the default view node is a star join view node.

   > i Note
   >
   > You can switch between the default view nodes. For example, from a default aggregation view node to a project view node or a star join view node. Right-click the default view node and select the required menu option.

7. Continue modeling the calculation view by selecting view nodes from the tool palette and adding them to the view editor.

   To switch between any of the inner aggregation or projection view nodes, right-click the view node and select the required menu option. The tool retains the same name after the switch operation.
8. Add data sources.

   To add data sources to the view node:

   a. Select a view node.
   b. In the editor toolbar, choose (Expand Details Panel).
   c. Choose (Add Data Source).
   d. In the *Find Data Sources* dialog box, select the type of the data source.
   e. Enter the name of the data source and select it from the list.

      You can add one or more data sources depending on the selected view node.
   f. Choose *Finish*.

   > i Note
   >
   > Supported data sources in view nodes in the current version.
   >
   > The *Find Data Sources* dialog box displays multiple object types in the search result. But, depending on the selected view node, you can only add activated (built) catalog tables, calculation views, SAP HANA CDS entities, SAP HANA CDS views, virtual tables, and table functions as data sources in the view nodes.
   >
   > You can also model calculation views using data sources from any tenant database available in the same SAP HANA instance.

9. (Optional) Add data sources from external services (HDI or non HDI containers).

You can use synonyms to access objects from user-defined schemas (non HDI) or from HDI containers. The tool automatically creates (or modifies the existing) `.hdbsynonym`, `.hdbgrants`, and `.hdbsynonymconfig` (which is created in `cfg` folder for objects in HDI), files that are necessary to consume the synonym.

> **Note**
>
> You can access data sources from external services only if you have created the necessary user-provided service and configured it in the `mta.yaml` file. For more information see, Consume Objects That are not Included in Your Development Project.

a. Select a view node.

b. Choose (Add Data Source).

The *Find Data Sources* dialog box appears.

c. In the *External Services* dropdown list, select the required external service.

d. Enter the name of the data source you want to access from the external service and select it from the search list.

You can add one or more data sources depending on the selected view node.

e. Choose *Create Synonym*.

f. (Optional) Modify the synonym name.

g. Define the object owner role and the application user role.

Object Type	Steps
Objects in HDI Containers	If the selected object is from a HDI container, 1. In the *Object Owner Role*, enter the name of the object owner role. 2. In the *Application User Role*, enter the name of the application user role. For more information on roles for HDI containers, see the topic, *Roles (.hdbrole)* in the *SAP HANA Developer Guide* (for XS Advanced). If the object for which you are creating the synonym is in a HDI container, select *Synonym for HDI Container*. This selection helps the tool identify that you are creating synonym for an object in the HDI container.
Objects in non HDI containers	If the selected object is from a HDI container, 1. In the *Object Owner Role*, enter the name of the object owner role. 2. In the *Application User Role*, enter the name of the application user role. For more information on roles for non HDI containers, see the topic, *Enable Access to Objects in a Remote Classic Schema* in the *SAP HANA Developer Guide* (for XS Advanced). **Note** If the object for which you are creating the synonym is in a non HDI container, ensure that you have not selected the *Synonym for HDI Container* checkbox.

h. Choose *Finish*.

The synonym is created, along with the `.hdbsynonym`, `.hdbgrants`, and `.hdbsynonymconfig` (which is created in `cfg` folder for objects in HDI) files in the same SAP HANA Database Module.

10. Define output columns.
    a. Select a view node.
    b. On the *Mapping* tab, select the column you want to add to the output.
    c. In the context menu, choose *Add To Output*.
    d. To add all columns in a data source to the output, from the context menu of the data source, select *Add To Output*.

    > **i Note**
    >
    > Using keep flag property. The *Keep Flag* property on attribute columns influence the result set. Use *Keep Flag* property, for example, to aggregate the measures by performing an SQL `GROUP BY` operation on them, even when they are not included in the query.
    >
    > 1. Select the view node.
    > 2. On the *Mapping* tab, select an output column.
    > 3. In the *Properties* section, set the value of *Keep Flag* property to *True*.

11. Define attributes and measures.

    To successfully activate a calculation view with data category as CUBE, you must specify at least one column as a measure.

    a. Select the *Semantics* view node.
    b. On the *Columns* tab, select a column value.
    c. In the *Type* dropdown list, select *Measure* or *Attribute*.

    If the data category is set to Cube, an additional aggregation column is available to specify the aggregation type for measures.

12. (Optional) Copy a calculation view.

    You can copy a calculation view and paste it using a different name within the same or different SAP HANA Database module. The tool automatically adjusts the calculation view ID in the view properties.

    a. In the Workspace view, right-click the required calculation view.
    b. Choose *Copy*.
    c. Right-click at the target folder and choose *Edit* > *Paste*.
    d. Provide a new name.
    e. Choose *OK*.

13. (Optional) View modeler objects in the outline pane.

    Use the outline pane in SAP Web IDE to obtain a quick overview of the modeler objects (view nodes and columns) in the calculation view.

    a. In the menu bar, choose *View* > *Outline*.

    The tool displays all modeler objects in the calculation view. You can also select an object in the outline pane, and navigate to the editor to identify where the object is used in the calculation view.

    b. In the right menu bar, choose (Outline) to show or hide the outline pane.

14. Choose *Save* to save your calculation view.
15. Build an SAP HANA Database Module.

The build process uses the design-time database artifacts to generate the corresponding runtime objects in the database catalog.

From the module context menu, choose *Build*.

Next Steps

After creating a calculation view, you can modify the output to your needs. The following table shows how you can modify the calculation view.

Working With View Nodes

Requirement	Task to Perform
Query data from two data sources and combine records from both the data sources, based on a join condition, or to obtain language-specific data.	Create Joins
Combine the results of two or more data sources.	Create Unions
Partition the data for a set of partition columns, and perform an order by SQL operation on the partitioned data.	Create Rank Nodes
Execute any of the available graph operations on the graph workspace.	Create Graph Nodes
Perform intersect set operations on two data sources.	Use Intersect Set Operation
Perform minus set operations on two data sources.	Use Minus Set Operation
Model calculation views with SAP HANA hierarchy functions.	Use Hierarchy Functions
Anonymize calculation view output.	Anonymize Data Using Calculation Views
Model table functions in calculation views with both tabular input parameters and scalar input parameters.	Model Table Functions as View Nodes
Filter the output of view nodes.	Filter Output of View Nodes.
Use a form-based editor to create virtual tables.	Create Virtual Tables

Working With Columns

Requirement	Task to perform
Count the number of distinct values for a set of attribute columns.	Create Counters
Create new output columns and calculate their values at runtime using an expression.	Create Calculated Columns
Assign semantic types to provide more meaning, and to attach information about the type of attributes and measures in calculation views.	Assign Semantics
Parameterize calculation views and execute them based on the values users provide at query runtime.	Create Input Parameters
Filter the results based on the values that users provide to attributes at runtime.	Assign Variables
Create level hierarchies to organize data in reporting tools.	Create Level Hierarchies

Requirement	Task to perform
Create parent-child hierarchies to organize data in reporting tools.	Create Parent-Child Hierarchies
Associate measures with currency codes and perform currency conversions.	Associate Measures with Currency
Associate measures with unit of measures and perform unit conversions.	Associate Measures with Unit of Measure
Define data masking for column values when modeling a calculation view.	Mask Column Values in Client Tools
Define default values for columns (both attributes and measures) when no value is provided during a SQL `INSERT` operation.	Handle Null Values in Columns
Group related measures and related attributes together in a folder.	Group Related Measures and Attributes

Working With Calculation View Properties

Requirement	Task to perform
Filter the view data either using a fixed client value or using a session client set for the user.	Filter Data for Specific Clients
Invalidate or remove data from the cache after specific time intervals.	Invalidate Cached Content
Discourage use of a calculation view.	Deprecate Calculation Views
Execute time travel queries on calculation views.	Enable Calculation Views for Time Travel Queries

Related Information

Working With View Nodes
Preview Calculation View Output [page 518]
Working With Attributes and Measures
Working With Calculation View Properties
Additional Functionality for Calculation Views
Defining Data Access Privileges [page 520]
Create Calculation Views with Star Joins
Create Calculation Views with Time Dimension
Example: Using Keep Flag

5.8.2 Preview Calculation View Output

After modeling calculation views based on your requirements, you can deploy them and preview their output from within the same tool.

Context

You can preview output data of calculation views in simple tabular format, or you can preview output data in graphical representations such as, bar graphs, area graphs, and pie charts. If your calculation view is defined with hierarchies, you can preview the output data in hierarchical representations.

You can export and download the output data to `.csv` files. You can also preview the SQL query that was executed to fetch the output data.

Procedure

1. Start SAP Web IDE in a Web browser.
2. In the *Workspace* view, select a calculation view to preview its output.
3. In the context menu, choose *Data Preview*.
4. Provide an input parameter or variable value.

   If you have defined any input parameters in the calculation view, provide the input parameter values.

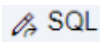
   a. Select an operator.
   b. Provide values for the *From* and *To* fields based on the selected operator.
   c. In the toolbar, choose (Open Content).

      In the *Raw Data* tab, the output data is displayed in tabular format.
5. (Optional) Apply filters.

   a. To apply filters on columns and preview the filtered output data, in the toolbar, choose (Add Filter).
   b. Choose *Add Filters*.
   c. Select columns and define the filter conditions.
   d. Choose *Apply*.
6. Export output data, if required.

   To export the raw data output to a `.csv` file,

   a. In the toolbar, choose (Download).
   b. In the *Delimiter* dropdown list, select a delimiter to use to separate the values in the `.csv` file.
   c. Choose *Download*.
7. View the SQL query for the calculation view, if required.

   a. To view the SQL query that the tool executed to fetch the data, in the toolbar, choose *SQL*.
   b. To view, edit, and execute the SQL query in the SQL editor, choose 

(Edit SQL Statement in SQL Console).

8. Preview the output of calculation views in graphical representations.

   You can preview the output of a calculation view as a bar graph, area graph, pie chart or table charts, and other graphical representations. These graphical representations allow more detailed analysis of output data.

   a. In the menu bar, choose *Analysis*.
   b. Configure the axis values by dragging and dropping the attributes and measures to the *Label Axis* and the *Value Axis*.

      The output data is displayed in a graphical representation. Select chart icons in the menu to view the output in a different graphical representation.

   c. In the toolbar, choose (Display Settings) to toggle legends and values.
   d. To view the query SQL query that was executed to fetch the data for the axis configurations, in the toolbar, choose *SQL*.
   e. To export the charts as `.png` files to your local system, in the toolbar, choose (Download) and select *Chart*.
   f. To export the raw output data as `.csv` files to your local system, in the toolbar, choose (Download) and select *Data*.

9. Preview output of hierarchies.

   SAP Database Explorer provides a *Hierarchies* tab to preview the output of calculation views that contain hierarchies. If you have defined any hierarchies in calculation views of type `Dimension` or `Cube`, you can preview output in hierarchical tree structures.

   > **i Note**
   >
   > The preview output of compound parent-child hierarchies is not supported in calculation views.

   a. In the menu bar, go to the *Hierarchies* tab.

      In the *Available Objects* pane, you can see all the hierarchies defined in the calculation view, including any hierarchies from shared dimensions. The measures defined in the calculation view are also displayed.

   b. Drag and drop a hierarchy to the *Selected Hierarchy* pane.

      > **! Restriction**
      >
      > You can preview output in hierarchical tree structure of only one hierarchy at a time. The preview of multiple hierarchies at the same time is not suported.

   c. (Optional) To preview the output with one or more measures from the calculation view, drag and drop the measure to the *Selected Measure(s)* pane.

      The output data is built in a hierarchical tree structure for the selected configuration. The root node is expanded, and the number of records displayed depends on the MDX console or SQL console settings.

      > **i Note**
      >
      > MDX queries do not support history enabled parameters, and do not obey any pruning settings defined for the calculation view. Thus, they are not part of the hierarchy data preview.

   d. If you are previewing output of a calculation view with data category `CUBE`, and if you want to view the MDX query that the tool executed to fetch the output data, in the toolbar, choose *MDX*.

e. If you are previewing output of a calculation view with data category `DIMENSION`, and if you want to view the SQL query that the tool executed on the BIMC tables to fetch the output data, in the toolbar, choose *SQL*.

f. To export the output data to a `.csv` file, in the toolbar, choose (Download).

10. Preview output of intermediate nodes.

    If you have activated the calculation view, you can also preview the output of any of its intermediate view nodes. This preview confirms that the output data that is passed to the higher view node levels.

    a. Switch to the *Development* perspective.
    b. In the *Workspace* view, select a calculation view and open it in the view editor.
    c. Right-click an intermediate view node and choose *Data Preview*.

Related Information

Create Calculation Views [page 512]

5.8.3 Defining Data Access Privileges

Use the analytic privilege editor in the SAP Web IDE to create analytic privileges.

You create analytic privileges to grant different users access to different portions of data in the same view based on their business role. Within the definition of an analytic privilege, the conditions that control which data users see is defined using SQL.

Standard object privileges (`SELECT`, `ALTER`, `DROP`, and more) implement coarse-grained authorization at object level only. Users either have access to an object, such as a table, view or procedure, or they don't. While this is sufficient, there are cases when access to data in an object depends on certain values or combinations of values. Analytic privileges are used in the SAP HANA database to provide such fine-grained control at row level of which data individual users can see within the same view.

Example

Sales data for all regions is contained within one calculation view. However, regional sales managers should only see the data for their region. In this case, an analytic privilege could be modeled so that they can all query the view, but only the data that each user is authorized to see is returned.

Related Information

Create Static SQL Analytic Privileges
Create Dynamic SQL Analytic Privileges

Create Analytic Privileges Using SQL Expressions
Structure of SQL-Based Analytic Privileges

6 Defining Web-based Data Access in XS Advanced

SAP HANA extended application services (SAP HANA XS) provide applications and application developers with access to the SAP HANA database using a consumption model that is exposed via HTTP.

In addition to providing application-specific consumption models, SAP HANA XS also host system services that are part of the SAP HANA database, for example: search services and a built-in Web server that provides access to static content stored in the SAP HANA repository.

The consumption model provided by SAP HANA XS focuses on server-side applications written in JavaScript and making use of a powerful set of specially developed API functions. However, you can use other methods to provide access to the data you want to expose in SAP HANA. For example, you can set up the Web-based data access for XS advanced applications using the following services:

- OData (v2 or v4)
  Use an ODATA service that exposes a data end point in SAP HANA
- XMLA
  Use the XML for Analysis (XMLA) interface to send a Multi-dimensional Expressions (MDX) query
- Information Access Protocol (InA)
  Provide access to data for the purposes of analysis with queries that use the Information Access (InA) protocol

Related Information

Maintaining OData Services in XS Advanced [page 522]
Data Access with XMLA in SAP HANA XS [page 610]
Data Access with InA in XS Advanced [page 618]

6.1 Maintaining OData Services in XS Advanced

Define OData services for your Java and JavaScript applications in XS advanced.

In XS advanced, you can create OData services that use either OData version 2.0 or OData version 4.0. OData version 2.0 is the basis for the XSODATA service infrastructure. Among other things, OData version 4.0 extends the query filters available, for example, `$select` and `$expand`.

> **i** Note
>
> The XSODATA infrastructure for OData services based on OData version 2 will no longer be extended or improved. For new OData services, it is recommended to move to OData version 4 in combination with

CDS or, alternatively, switch to the new SAP Cloud application programming model mentioned below. In addition, XSODATA will not support SAP HANA Cloud services, as described in SAP note: 3013788.

The OData version your XS advanced applications can consume is determined by the programming model you use to develop the application, for example, XS advanced or SAP Cloud application programming model, as described in the following list:

- SAP Cloud application programming model
  XS advanced supports the SAP Cloud application programming model, which is now the recommended model for new applications that make use of OData services. This programming model provides both Java and JavaScript APIs that enable the easy use and integration of either OData v 2.0 or v 4.0 services.
- XS advanced application programming model:
  - Java applications
    In XS advanced, CDS-OData is currently available for use with Java and OData versions 2 and 4.
  - JavaScript & Node.js
    For JavaScript and Node.js applications in XS advanced, you can define an XS OData service using OData version 2.0, for example, `myOdataV2Service.xsodata`. However, since there are no plans to extend or improve the underlying XSOData infrastructure, and for the sake of future compatibility, it is recommended to use the new SAP Cloud application programming model for your OData services in this scenario.

    > **i Note**
    >
    > Although it is possible to consume an OData v4 service, it is not recommended since it would require adding a Java module to your XS advanced Multi-Target JavaScript Application (MTA) and then routing the OData requests to the Java component. For this reason, it is recommended to use the SAP Cloud programming model.

OData Versions and SAP Application Programming Models

Application Programming Model	Java	JavaScript/Node.js
SAP Cloud Programming Model *	OData v2.0, v4.0	OData v4.0
SAP HANA XS advanced	OData v2.0, v4.0	OData v2.0

> **→ Tip**
>
> The SAP Cloud application programming model is the recommended development model for new OData projects. Applications developed using the SAP Cloud application programming model can be deployed both to the SAP Cloud platform and the XS advanced run-time platform.

Related Information

Defining OData v2 Services for XS Advanced JavaScript Applications [page 524]
Defining OData v4 Services for XS Advanced Java Applications [page 580]
Working with the SAP Cloud Application Programming Model [page 108]
SAP Note 2691905

SAP Note 3013788

6.1.1 Defining OData v2 Services for XS Advanced JavaScript Applications

Create the OData service definitions consumed by your SAP HANA XS advanced JavaScript application.

An OData v2 service definition for your XS advanced JavaScript application is described in an “xsodata” artifact, for example, `myOdataService.xsodata`; the definition for your OData version 2 service should be placed in the `js/src/odata/services/` folder of your JavaScript project module. The following example shows two OData services (`srv1.xsodata` and `srvN.xsodata`) consumed by a JavaScript application.

> ⚠ Caution
>
> The XSODATA infrastructure for OData services based on OData version 2 will no longer be extended or improved. If you are developing a new OData service, it is recommended to move to OData version 4 in combination with CDS.

Example

Sample Code

```
JavaScript-AppName
|- db/
|  |- package.json
|  |- src/
|     |- .hdiconfig
|     \- mytable.hdbdd
|
|- web/
|  |- xs-app.json
|  |- package.json
|  \- resources/
|- js/
|  |- start.js
|  |- package.json
|  \- src/
|     \- odata/                  # XSOData v2 service resources
|        |- resources/           # XSOData v2 service resources
|        \- services/            # XSOData v2 services
|           | - srv1.xsodata     # XSOData v2 service definition
|           \ - srvN.xsodata     # XSOData v2 service definition
|- security/
|  \- xs-security.json
\- mta.yaml
```

Exposing a Database Table

You can expose the database table `db/src/mytable.hdcds` as `'My Entity Set'` as follows:

1. Create the project structure (as shown in the example above) in your local system.
2. Create the OData service definition file `myservice.xsodata`.
3. Expose the table as EntitySet in the `xsodata` file:

   Sample Code

   ```
   service {
   "mytable" as "MyEntitySet";
   }
   ```

4. Deploy your application with the client deployment tool.
5. Call your service from the client UI, for example, a Web browser:
   You can call your OData service with: the following URI
   `http://<myServer>:<port>/odata/services/<myService>.xsodata`
   You can requests the metadata of the OData service using the following URI
   `http://<myServer>:<port>/odata/services/<myService>.xsodata/$metadata`

! Restriction

OData supports both the ATOM and JSON formats in request and response payloads. However, the Node.js module only support JSON. To specify JSON as the payload format, you can either add the following system-query option to the request "`$format=json`" or add the following request header "`Accept:json`".

Related Information

Define the Data an OData Service Exposes (XS Advanced) [page 526]
Create an OData Service Definition (XS Advanced) [page 526]

6.1.1.1 Define the Data an OData Service Exposes (XS Advanced)

An OData service exposes data stored in database tables or views as OData collections for analysis and display by client applications. However, first of all, you need to ensure that the tables and views to expose as an OData collection actually exist.

Context

To define the data to expose using an OData service, you must perform the following two steps:

Procedure

1. Create a simple database table to expose with an OData service.
2. Create a simple database view to expose with an OData service. This step is optional; you can expose tables directly. In addition, you can create a calculation view as a modeling view.

6.1.1.2 Create an OData Service Definition (XS Advanced)

The OData service definition is a configuration file you use to specify which data (for example, views or tables) is exposed as an OData collection for analysis and display by client applications.

Context

An OData service for SAP HANA XS Advanced is defined in a text file with the file extension `.xsodata`, for example, `OdataSrvDef.xsodata`. It must contain at least the entry service {}, which would generate an operational OData service with an empty service catalog and an empty metadata file.

Procedure

1. Create the file that will contain your OData service definition.

   An OData service definition is described in an `xsodata` artifact, for example, `myOdataService.xsodata`; the service definition should be placed in the `services/` folder of your OData project structure.
2. Define the OData service.

   The OData service definition uses the OData Service Definition Language (OSDL), which includes a list of keywords that you specify in the OData service-definition file to enable important features.

The following example shows a simple OData service definition exposing a simple table:

```
service {
        "sample.odata::table" as "MyTable";
}
```

This service definition exposes a table defined in the file `sample.odata:table.hdbtable` and creates an EntitySet for this entity named `MyTable`. The specification of an alias is optional. If omitted, the default name of the EntitySet is the name of the repository object file, in this example, `table`.

3. Save the OData service definition.

Related Information

Define the Data an OData Service Exposes

6.1.1.2.1 OData Service Definitions (XS Advanced)

The OData service definition is the mechanism you use to define what data to expose with OData, how, and to whom. Data exposed as an OData collection is available for analysis and display by client applications, for example, a browser that uses functions provided by an OData client library running on the client system.

To expose information by means of OData to applications using SAP HANA XS, you must define database views that provide the data with the required granularity. Then you create an OData service definition, which is a file you use to specify which database views or tables are exposed as OData collections.

> **i Note**
>
> SAP HANA XS supports OData version 2.0, which you can use to send OData queries (for example, using the HTTP `GET` method). Language encoding is restricted to UTF-8.

An OData service for SAP HANA XS is defined in a text file with the file suffix `.xsodata`, for example, `OdataSrvDef.xsodata`. The file must contain at least the entry `service {}`, which would generate a completely operational OData service with an empty service catalog and an empty metadata file. However, usually you use the service definition to expose objects in the database catalog, for example: tables, SQL views, or calculation rules.

In the OData service-definition file, you can use the following ways to name the SAP HANA objects you want to expose by OData:

> **i Note**
>
> The syntax to use in the OData service-definition file to reference objects is dictated by the object type, for example, database catalog (runtime).

- **Database objects**
  Expose an object using the object's database catalog (runtime) name. The support for database objects is mainly intended for existing or replicated objects that do not have a repository design-time representation.

The following example shows how to include a reference to a table in an OData service definition using the table's *runtime* name.

```
service {
    "myTable" as "MyTable";
}
```

By default, all entity sets and associations in an OData service are writeable, that is they can be modified with a CREATE, UPDATE, or DELETE requests. However, you can prevent the execution of a modification request by setting the appropriate keyword (*create*, *update*, or *delete*) with the `forbidden` option in the OData service definition. The following example of an OData service definition for SAP HANA XS shows how to prevent any modification to the table `myTable` that is exposed by the OData service. Any attempt to make a modification to the indicated table using a CREATE, UPDATE, or DELETE request results in the HTTP response status `403 FORBIDDEN`.

```
service {
     “sap.test::myTable”
          create forbidden
          update forbidden
          delete forbidden;
}
```

For CREATE requests, for example, to add a new entry to a table exposed by an OData service, you must specify an explicit key (not a generated key); the key must be included in the payload of the CREATE request. For UPDATE and DELETE requests, you do not need to specify the key explicitly (and if you do, it will be ignored); the key is already known, since it is essential to specify which entry in the table must be modified with the UPDATE or DELETE request.

> **i** Note
>
> Without any support for IN/OUT table parameters in SQLScript, it is not possible to use a sequence to create an entry in a table exposed by an OData service. However, you can use XS JavaScript exits to update a table with a generated value.

6.1.1.2.1.1 OData Service-Definition Type Mapping

During the activation of the OData service definition, SQL types defined in the service definition are mapped to EDM types according to a mapping table.

For example, the SQL type "`Time`" is mapped to the EDM type "`EDM.Time`"; the SQL type "`Decimal`" is mapped to the EDM type "`EDM.Decimal`"; the SQL types "`Real`" and "`Float`" are mapped to the EDM type "`EDM.Single`".

> **i** Note
>
> The OData implementation in SAP HANA Extended Application Services (SAP HANA XS) does not support all SQL types.

In the following example, the SQL types of columns in a table are mapped to the EDM types in the properties of an entity type.

```
{name = "ID"; sqlType = INTEGER; nullable = false;}, {name = "RefereeID";
sqlType = NVARCHAR; length = 10; nullable = true;}
```

```
<Property Name="ID" Type="Edm.Int32" Nullable="false"/> <Property
Name="RefereeID" Type="Edm.String" MaxLength="10" Nullable="true"/>
```

Related Information

OData Service Definition: SQL-EDM Type Mapping (XS Advanced) [page 575]
OData Service Definitions

6.1.1.2.1.2 OData Service-Definition Features

The OData service definition provides a list of keywords that you use in the OData service-definition file to enable important features. For example, the following list illustrates the most-commonly used features used in an OData service-definition and, where appropriate, indicates the keyword to use to enable the feature:

- **Aggregation**
  The results of aggregations on columns change dynamically, depending on the grouping conditions. As a result, aggregation cannot be done in SQL views; it needs to be specified in the OData service definition itself. Depending on the type of object you want to expose with OData, the columns to aggregate and the function used must be specified explicitly (**explicit aggregation**) or derived from metadata in the database (**derived aggregation**). Note that aggregated columns cannot be used in combination with the `$filter` query parameter, and aggregation is only possible with generated keys.
- **Association**
  Define associations between entities to express relationships between entities. With associations it is possible to reflect foreign key constraints on database tables, hierarchies and other relations between database objects.
- **Key Specification**
  The OData specification requires an EntityType to denote a set of properties forming a unique key. In SAP HANA, only tables can have a unique key, the primary key. All other (mostly view) objects require you to specify a key for the entity. The OData service definition language (OSDL) enables you to do this by denoting a set of existing columns or by generating a **local key**. Bear in mind that local keys are transient; they exist only for the duration of the current session and cannot be dereferenced.

  > **i** Note
  >
  > OSDL is the language used to define a service definition; the language includes a list of keywords that you use in the OData service-definition file to enable the required features.

- **Parameter Entity Sets**
  You can use a special parameter entity set to enter input parameters for SAP HANA calculation views and analytic views. During activation of the entity set, the specified parameters are retrieved from the metadata of the calculation (or analytical) view and exposed as a new **EntitySet** with the name suffix "`Parameters`", for example "`CalcViewParameters`".
- **Projection**
  If the object you want to expose with an OData service has more columns than you actually want to expose, you can use SQL views to restrict the number of selected columns in the `SELECT`. However, for those cases

where SQL views are not appropriate, you can use the **with** or **without** keywords in the OData service definition to **include** or **exclude** a list of columns.

Related Information

OData Service-Definition Examples

6.1.1.3 Enable SAP OData Annotations

Add annotations to the OData v2`$metadata` document.

Context

Version two of the OData protocol allows you to add annotations to the `$metadata` document. The purpose of the annotations is to add information and hints to the EDM elements; the additional information can be used by the service clients (for example, an SAPUI5 application) to more clearly represent the service entities.

> **i** Note
>
> SAP defines its own specific set of OData annotations. However, not all of the SAP OData annotations are supported in XS OData for XS advanced model. For more information, see *SAP OData Annotations* in *Related Links* below.

Procedure

1. Open the OData service description, for example, `myODataService.xsodata`.
2. Enable OData annotations.

   Use the `annotations {}` keyword to enable OData annotations, as illustrated in the following example:

   > Sample Code
   >
   > Enabling OData Annotations

```
service ... {
    ...
}
annotations {
   enable OData4SAP;
}
```

> **! Restriction**
>
> The annotation configuration must be specified immediately after the "`service`" element.

If annotations are enabled for an XS OData service in XS advanced, the following XML name space is added for the `Edmx` element of the `$metadata` document:

> **Sample Code**
>
> ```
> xmlns:sap="http://www.sap.com/Protocols/SAPData"
> ```

Related Information

SAP OData Annotations [page 531]
SAP Annotations for OData Version 2.0

6.1.1.3.1 SAP OData Annotations

The OData v2 protocol allows the use of annotations in the metadata document.

The second version of the OData protocol allows you to add annotations to the metadata document; the purpose of the annotations is to add information and hints to the EDM elements. This structural metadata makes it easy to understand a service, and human-readable documentation can be directly embedded into the metadata document, helping developers consume an OData service. The added information can be used by the service clients (for example, and SAPUI5 application) to better represent the service entities. The annotations are divided into annotation elements and annotation attributes, represented as XML elements and XML attributes respectively.

Located at the OData service root, the service document lists the top-level resources that are available. The metadata document, on the other hand, describes the structure of all resources in the OData service; the metadata document is located at the address `$metadata` relative to the OData service root.

The OData protocol defines the concept of the annotations without adding any details concerning the particular names and values, which can or should be used for them. In addition, SAP defines its own specific set of annotations.

> **i Note**
>
> Not all of the SAP OData annotations are supported in XS OData for XS advanced model. For more information, see *SAP Annotations for OData Version 2.0* in *Related Links* below.

By default, the annotations are **not** serialized in the `$metadata` document of an XS OData service. For each of the annotations a default value is defined according to the information provided in *SAP Annotations for OData Version 2.0*. It is important to remember that an annotation is added to the `metadata` document only if the annotation's value differs from the default value.

To enable the annotations for a particular XS OData service, the following configuration has to be specified in the service document (the respective `.xsodata` service-definition file), as illustrated in the following example:

Sample Code

Enabling OData Annotations

```
service ... {
    ...
}
annotations {
   enable OData4SAP;
}
```

Restriction

The annotation configuration must be specified immediately after the "`service`" element.

If annotations are enabled for an XS OData service in XS advanced, the following XML name space is added for the `Edmx` element of the `metadata` document:

Sample Code

```
xmlns:sap="http://www.sap.com/Protocols/SAPData"
```

Related Information

SAP Annotations for OData Version 2.0
Enable SAP OData Annotations [page 530]

6.1.1.3.1.1 OData Entity Set Annotations

A list of the attributes that can be used as annotations in the entity **set** element (`edm:EntitySet`).

For XS OData services used in XS advanced, entity sets can be annotated with the following attributes:

- `sap:addressable` [page 533]
  Direct access to the specified entity set is permitted
- `sap:creatable` [page 533]
  New entities can be created in the specified entity set
- `sap:updatable` [page 533]
  Existing entities in the specified entity set can be updated
- `sap:deletable` [page 534]
  Existing entities can be removed from the specified entity set

sap:addressable

Permit direct access to the specified entity set.

Default Value

“true”

Additional Supported Values

“false” if the entity set represents either a calculation view or input parameters for a calculation view.

sap:creatable

Allow the creation of new entities in the specified entity set.

Default Value

“true”

Additional Supported Values

The following table shows the additional values that are supported for the attribute `sap:creatable` in the `edm:EntitySet` element and the corresponding requirements.

Additional Supported Values

Parameter Value	Requirement
“false”	The "`create forbidden`" setting is defined for the entity set in the OData service definition (`.xsodata`) file
	The entity set represents a database view, for example, a table or a calculation view.
	The entity set represents input parameters for a calculation view.
	“`aggregation`” is defined for the entity set, for example, using the “`aggregates always`” expression in the OData service definition (`.xsodata`) file.

sap:updatable

Allow changes and updates to existing entities in the specified entity set.

Default Value

“true”

Additional Supported Values

The following table shows the additional values that are supported for the attribute `sap:updatable` in the `edm:EntitySet` element and the corresponding requirements.

Additional Supported Values

Parameter Value	Requirement
“false”	The "`update forbidden`" setting is defined for the entity set in the OData service definition (`.xsodata`) file
	The entity set represents a database view, for example, a table or a calculation view.
	The entity set represents input parameters for a calculation view.
	“`aggregation`” is defined for the entity set, for example, using the “`aggregates always`” expression in the OData service definition (`.xsodata`) file.
	A generated key is defined for the entity set.

sap:deletable

Allow the deletion of entities from the specified entity set.

Default Value

“true”

Additional Supported Values

The following table shows the additional values that are supported for the attribute `sap:deletable` in the `edm:EntitySet` element and the corresponding requirements.

Additional Supported Values

Parameter Value	Requirement
“false”	The "`delete forbidden`" setting is defined for the entity set in the OData service definition (`.xsodata`) file
	The entity set represents a database view, for example, a table or a calculation view.
	The entity set represents input parameters for a calculation view.
	“`aggregation`” is defined for the entity set, for example, using the “`aggregates always`” expression in the OData service definition (`.xsodata`) file.
	A generated key is defined for the entity set.

Related Information

SAP OData Annotations [page 531]
Enable SAP OData Annotations [page 530]
SAP Annotations for OData Version 2.0

6.1.1.3.1.2 OData Entity-Type Annotations

A list of the attributes that can be used as annotations in the entity **type** element (`edm:EntityType`).

sap:semantics

Describes the semantic of the entity type.

Default Value

Undefined.

Additional Supported Values

The following values are supported, in addition to the default value (if defined):

- `"aggregate"` value in one of the following cases:
  - Where aggregation is defined for the entity set of the entity type using `"aggregates"` expression in `.xsodata` file
  - Where the entity type represents a calculation view, which has a measure attribute
- `"parameters"` value if the entity type represents input parameters for a calculation view

Related Information

OData Entity Set Annotations [page 532]
Enable SAP OData Annotations [page 530]
SAP Annotations for OData Version 2.0

6.1.1.3.1.3 OData Annotations for Entity-Type Properties

A list of the supported annotations for properties of entity types.

- `sap:semantics` [page 536]
- `sap:parameter` [page 537]

- `sap:label` [page 537]
- `sap:filterable` [page 538]
- `sap:display-format` [page 538]
- `sap:aggregation-role` [page 538]
- `sap:unit` [page 539]
- `sap:filter-restriction` [page 539]

sap:semantics

The semantic of the entity-type property.

Default Value

Undefined; there is no defined default value.

Additional Supported Values

The annotation is added only for the properties representing calculated attributes or input parameters in a calculation view. The annotation value corresponds to the semantic type of a calculated attribute or an input parameter. The semantic type of an **input parameter** is specified as a value of "`type`" attribute of "`valueDomain`" element, defined in a calculation view definition file (for example, `myCalcView.hdbcalculationview`), as illustrated in the following example:

Sample Code

```
<variable id="parameterID" parameter="true">
    <variableProperties datatype="NVARCHAR" length="10" mandatory="true" >
        <valueDomain type="Currency"/>
        ...
    </variableProperties>
</variable>
```

Semantic type of a **calculated attribute** is specified as a value of a "`semanticType`" attribute of "`calculatedAttribute`" element, defined in a calculation view definition file (for example, `myCalcView.hdbcalculationview`), as illustrated in the following example:

Sample Code

```
<calculatedAttribute id="attributeID" semanticType="currencyCode">
    ...
</calculatedAttribute>
```

The following table shows how the semantic type values of calculated attributes and input parameters are mapped to the values of the "`sap:semantics`" annotation:

! Restriction

Only those values listed in the following table are supported.

Mapping Between Attribute Values and Annotation

Calculated Attribute Semantic Type	Input Parameter Semantic Type	sap:semantics value
currencyCode	Currency	currency-code
unitOfMeasure	UnitOfMeasure	unit-of-measure
date.businessDateFrom	-	dtstart
date.businessDateTo	-	dtend

sap:parameter

A property is annotated with this annotation, if it represents a parameter.

Default Value

Undefined; there is no defined default value.

Additional Supported Values

The annotation is added only to the properties representing input parameters of a calculation view. The annotation value corresponds to the value of the "`mandatory`" attribute in the input parameter definition, defined in a calculation view definition file (`myCalcView.hdbcalculationview`).

Sample Code

```
<variable id="inputParameterId" parameter="true">
    <variableProperties datatype="INTEGER" mandatory="true">
</variable>
```

The following table shows how the value of the "`mandatory`" attribute value is mapped to the value of the "`sap:parameter`" annotation:

Mapping Between Attribute Values and Annotation

"mandatory" Attribute Value	sap:parameter Value
true	mandatory
false	optional

sap:label

A short, human-readable text that is suitable for use in labels and captions in a user interface (UI).

Default Value

Undefined; there is no defined default value.

Additional Supported Values

The annotation is added only for the properties representing calculated attributes or input parameters of a calculation view. The annotation value corresponds to the "`defaultDescription`" attribute in the definition of an input parameter or a calculated attribute in the calculation view definition (`myCalcView.hdbcalculationview`) file.

Sample Code

```
<variable id="inputParameterId" parameter="true">
    <descriptions defaultDescription="Input parameter label"/>
</variable>
<calculatedAttribute id="calcAttributeId">
    <descriptions defaultDescription="Calculated attribute label"/>
</calculatedAttribute>
```

sap:filterable

Indicates if the property can be used with the "`$filter`" system query option.

Default Value

"`true`"

Additional Supported Values

The annotation is added with the "`false`" value if the property satisfies one of the following conditions:

- The property represents a generated key
- The property represents a measure attribute of a calculation view
- The property is used in the aggregation, defined as the "`aggregates always`" expression in the XS OData service-definition (`.xsodata`) file.

sap:display-format

The format used to display the property.

Default Value

Undefined; there is no defined default value.

Additional Supported Values

The "`Date`" value can be used if the `SQL DATE` type is used for the property on the database side.

sap:aggregation-role

The aggregation role of the property.

Default Value

Undefined; there is no defined default value.

Additional Supported Values

The "`measure`" value can be used if the property represents a measure attribute of a calculation view or is used in the aggregation that is defined in the "`aggregates always`" expression in the XS advanced OData service-definition (`.xsodata`) file.

sap:unit

The name of a property in the context of the entity type containing the currency code or unit of measure for a numeric value of the current property.

Default Value

Undefined; there is no defined default value.

Additional Supported Values

The annotation is added only for the properties representing calculated attributes of a calculation view. The annotation value corresponds to the value of "`attributeName`" attribute of "`unitCurrencyAttribute`" element of the calculated attribute definition in `myCalcView.hdbcalculationview` file.

> ℹ Note
>
> In the following example, the annotation value will be "`currencyCodeAttribute`".

> ‹≡› Sample Code

```
<calculatedAttribute id="attributeId">
    <unitCurrencyAttribute attributeName="currencyCodeAttribute"/>
</calculatedAttribute>
```

sap:filter-restriction

Describes filter restrictions for the property, if any exist.

Default Value

Undefined; there is no defined default value.

Additional Supported Values

The annotation is added only for the properties representing input parameters of a calculation view. The annotation value corresponds to the values of "`multiLine`" and "`type`" attributes of the "`selection`" element of the input parameter definition in the calculation view definition file (for example, `myCalcView.hdbcalculationview`.

Sample Code

```
<variable id="parameterId" parameter="true">
    <variableProperties datatype="INTEGER">
        <selection multiLine="false" type="SingleValue"/>
    </variableProperties>
</variable>
```

The following table below shows the mapping between the "`multiLine`" and "`type`" attribute values and the value of the "`sap:filter-restriction`" annotation.

! Restriction

Only those values specified in the following table are supported.

Mapping Between Attribute Values and Annotation

multiLine	type	sap:filter-restriction
false	SingleValue	single-value
true	SingleValue	multi-value
false	Interval	interval

Related Information

Enable SAP OData Annotations [page 530]
OData Entity Set Annotations [page 532]
OData Entity-Type Annotations [page 535]
SAP Annotations for OData Version 2.0

6.1.1.3.1.4 OData Navigation Property Annotations

A list of the annotations supported for use with relations between entities.

Overview

- `sap:creatable` [page 541]
- `sap:filterable` [page 541]

sap:creatable

Indicates if new related entities can be created.

Default Value

"`true`"

Additional Supported Values

The annotation value is always "`false`" because neither "`deep insert`" (`POST` request payload containing data for both parent and related entity) nor `POST` request for the `.../EntitySet(key)/navPropertyName` URL is supported in XS OData for XS advanced.

sap:filterable

Indicates if the property can be used with the "`$filter`" system query option.

Default Value

"`true`"

Additional Supported Values

The annotation value is always "`false`" because navigation properties cannot be used in a `$filter` system query option in XS OData for XS advanced.

Related Information

OData Entity Set Annotations [page 532]
OData Entity-Type Annotations [page 535]
OData Annotations for Entity-Type Properties [page 535]
OData Association Set Annotations [page 542]
SAP Annotations for OData Version 2.0

6.1.1.3.1.5 OData Association Set Annotations

A list of the annotations supported for use with relations between association sets.

Overview

- `sap:creatable` [page 542]
- `sap:updatable` [page 542]
- `sap:deletable` [page 542]

sap:creatable

Allow the creation of relations represented by the association set.

Default Value

“true”

Additional Supported Values

“`false`” if the association set connects entity sets for calculation view results and calculation view input parameters.

sap:updatable

Allow changes and updates to relations represented by the association set.

Default Value

“true”

Additional Supported Values

“`false`” if the association set connects entity sets for calculation view results and calculation view input parameters.

sap:deletable

Allow the deletion of relations represented by the association set.

Default Value

“true”

Additional Supported Values

"`false`" if the association set connects entity sets for calculation view results and calculation view input parameters.

Related Information

OData Entity Set Annotations [page 532]
OData Entity-Type Annotations [page 535]
OData Annotations for Entity-Type Properties [page 535]
OData Navigation Property Annotations [page 540]
SAP Annotations for OData Version 2.0

6.1.1.4 OData Service-Definition Examples (XS Advanced)

The OData service definition describes how data exposed in an end point can be accessed by clients using the OData protocol.

Each of the examples listed below is explained in a separate section. The examples show how to use the OData Service Definition Language (OSDL) in the OData service-definition file to generate an operational OData service that enables clients to use SAP HANA XS to access the OData end point you set up.

- Empty Service
- Namespace Definition
- Object Exposure
- Property Projection
- Key Specification
- Associations
- Aggregation
- Parameter Entity Sets
- Nullable Properties

> **! Restriction**
>
> OData supports both the ATOM and JSON formats in request and response payloads. However, the Node.js module only support JSON. To specify JSON as the payload format, you can either add the following system-query option to the request "`$format=json`" or add the following request header "`Accept:json`".

Related Information

OData Empty Service [page 544]

OData Namespace Definition [page 545]
OData Object Exposure [page 545]
OData Property Projection [page 546]
OData Key Specification [page 547]
OData Associations [page 549]
OData Aggregation [page 553]
OData Parameter Entity Sets [page 554]
OData ETag Support [page 556]
OData Nullable Properties [page 557]
OData Configurable Cache Settings [page 557]
Custom Exits for OData Write Requests

6.1.1.4.1 OData Empty Service

An OData service for SAP HANA XS is defined by a text file containing at least the following line:

Service definition `sample.odata:empty.xsodata`

```
service {}
```

A service file with the minimal content generates an empty, completely operational OData service with an empty service catalog and an empty metadata file:

> **i** Note
>
> - Examples and graphics are provided for illustration purposes only; some URLs may differ from the ones shown.

`http://<myHANAServer>:<port>/odata/services/<myService>.xsodata`

```
{
  "d" : {
    "EntitySets" : []
  }
}
```

`http://<myHANAServer>:<port>/odata/services/<myService>.xsodata/$metadata`

```
<edmx:Edmx Version="1.0"
xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx">
   <edmx:DataServices m:DataServiceVersion="2.0"
   xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">
      <Schema Namespace="sample.odata.empty"
      xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
      xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
      xmlns="http://schemas.microsoft.com/ado/2007/05/edm">
         <EntityContainer Name="empty" m:IsDefaultEntityContainer="true"/>
      </Schema>
   </edmx:DataServices>
</edmx:Edmx>
```

An empty service metadata document consists of one `Schema` containing an empty `EntityContainer`. The name of the `EntityContainer` is the name of the `.xsodata` file, in this example "`empty`".

6.1.1.4.2 OData Namespace Definition

Every `.xsodata` file must define it's own namespace by using the `namespace` keyword:

Service definition `sample.odata:namespace.xsodata`

```
service namespace "my.namespace" {}
```

The resulting service metadata document has the specified schema namespace:

> **i** Note
>
> Examples and graphics are provided for illustration purposes only; some URLs may differ from the ones shown.

`http://<myHANAServer>:<port>/odata/services/<myService>.xsodata/$metadata`

```
<edmx:Edmx Version="1.0"
xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx">
   <edmx:DataServices m:DataServiceVersion="2.0"
   x[illegible]/ado/2007/08/dataservices/metadata">
      <Schema Namespace="my.namespace"
      [illegible]com/ado/2007/08/dataservices"
        xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
        xmlns="http://schemas.microsoft.com/ado/2007/05/edm">
          <EntityContainer Name="namespace" m:IsDefaultEntityContainer="true"/>
      </Schema>
   </edmx:DataServices>
</edmx:Edmx>
```

6.1.1.4.3 OData Object Exposure

In the examples provided to illustrate object exposure, the following definition of a table applies:

Table definition `sample.odata:table.hdbtable`

```
COLUMN TABLE "sample.odata::table" (
    "ID" INTEGER,
    "Text" NVARCHAR(1000),
    "Time" TIMESTAMP,
    PRIMARY KEY ("ID")
);
```

Database Objects

Similar to the exposure of an object by using the repository design-time name is the exposure by the database name:

Service definition `sample.odata:db.xsodata`

```
service {
   "sample.odata::table" as "MyTable";
}
```

The service exposes the same table by using the database catalog name of the object and the name of the schema where the table is created in.

6.1.1.4.4 OData Property Projection

If the object you want to expose with an OData service has more columns than you actually want to expose, you can use SQL views to restrict the number of selected columns in the `SELECT`.

Nevertheless, SQL views are sometimes not appropriate, for example with calculation views, and for these cases we provide the possibility to restrict the properties in the OData service definition in two ways. By providing an including or an excluding list of columns.

Including Properties

You can specify the columns of an object that have to be exposed in the OData service by using the with keyword. Key fields of tables must not be omitted.

Service definition `sample.odata:with.xsodata`

```
service {
    "sample.odata::table" as "MyTable" with ("ID","Text");
}
```

The resulting `EntityType` then contains only the properties derived from the specified columns:

> **i** Note
>
> Examples and graphics are provided for illustration purposes only; some URLs may differ from the ones shown.

`http://<myHANAServer>:<port>/odata/services/<myService>.xsodata/$metadata`

```
<EntityType Name="MyTableType">
   <Key>
      <PropertyRef Name="ID"/>
   </Key>
   <Property Name="ID" Type="Edm.Int32" Nullable="false"/>
   <Property Name="Text" Type="Edm.String" Nullable="true" MaxLength="1000"/>
</EntityType>
```

Excluding Properties

The opposite of the `with` keyword is the `without` keyword, which enables you to specify which columns you do NOT want to expose in the OData service:

Service definition `sample.odata:without.xsodata`

```
 service {
    "sample.odata::table" as "MyTable" without ("Text","Time");
}
```

The generated `EntityType` then does NOT contain the properties derived from the specified columns:

`http://<myHANAServer>:<port>/odata/services/<myService>.xsodata/$metadata`

```
<EntityType Name="MyTableType">
   <Key>
      <PropertyRef Name="ID"/>
   </Key>
   <Property Name="ID" Type="Edm.Int32" Nullable="false"/>
</EntityType>
```

6.1.1.4.5 OData Key Specification

The OData specification requires an `EntityType` to denote a set properties forming a unique key. In HANA only tables may have a unique key, the primary key. For all other (mostly view) objects you need to specify a key for the entity.

In OSDL, you can specify a key for an entity/object by denoting a set of existing columns or by generating a key.

> **i** Note
>
> Key attributes are not evaluated.

For the examples illustrating key specification, we use the following SQL view, which selects all data from the specified table.

View definition `sample.odata:view.hdbview`

```
{
    VIEW "sample.odata::view" as select * from "sample.odata::table"
 }
```

Existing Key Properties

If the object has set of columns that may form a unique key, you can specify them as key for the entity. These key properties are always selected from the database, no matter if they are omitted in the `$select` query

option. Therefore explicit keys are not suitable for calculation views and analytic views as the selection has an impact on the result.

Service definition `sample.odata:explicitkeys.xsodata/$metadata`

```
 service {
     "sample.odata::view" as "MyView" key ("ID","Text");
}
```

The metadata document for the exposure of the view above is almost equal to the metadata document for repository objects. Only the key is different and consists now of two columns:

> **i Note**
>
> Examples and graphics are provided for illustration purposes only; some URLs may differ from the ones shown.

`http://<myHANAServer>:<port>/odata/services/<myService>.xsodata/$metadata`

```
<edmx:Edmx Version="1.0" xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx">
   <edmx:DataServices m:DataServiceVersion="2.0"
   xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">
      <Schema Namespace="sample.odata.explicitkeys"
      xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
      xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
      xmlns="http://schemas.microsoft.com/ado/2007/05/edm">
         <EntityType Name="MyViewType">
            <Key>
               <PropertyRef Name="ID"/>
               <PropertyRef Name="Text"/>
            </Key>
            <Property Name="ID" Type="Edm.Int32" Nullable="false"/>
            <Property Name="Text" Type="Edm.String" Nullable="true" MaxLength="1000"/>
            <Property Name="Time" Type="Edm.DateTime" Nullable="true"/>
         </EntityType>
         <EntityContainer Name="explicitkeys" m:IsDefaultEntityContainer="true">
            <EntitySet Name="MyView"
            EntityType="sample.odata.explicitkeys.MyViewType"/>
         </EntityContainer>
      </Schema>
   </edmx:DataServices>
</edmx:Edmx>
```

> **⚠ Caution**
>
> The OData infrastructure cannot check whether your specified keys are unique, so be careful when choosing keys.

Generated Local Key

For objects that do not have a unique key in their results, for example, calculation views or aggregated tables, you can generate a locally valid key. This key value numbers the results starting with `1` and is not meant for

dereferencing the entity; you cannot use this key to retrieve the entity. The key is valid only for the duration of the current session and is used only to satisfy OData's need for a unique ID in the results. The property type of a generated local key is `Edm.String` and cannot be changed.

Service definition `sample.odata:generatedkeys.xsodata`

```
service {
    "sample.odata::view" as "MyView" key generate local "GenID";
}
```

`http://<myHANAServer>:<port>/odata/services/<myService>.xsodata/$metadata`

```
<EntityType Name="MyViewType">
  <Key>
    <PropertyRef Name="GenID"/>
  </Key>
  <Property Name="GenID" Type="Edm.String" Nullable="false" MaxLength="2147483647"/>
  <Property Name="ID" Type="Edm.Int32" Nullable="false"/>
  <Property Name="Text" Type="Edm.String" Nullable="true" MaxLength="1000"/>
  <Property Name="Time" Type="Edm.DateTime" Nullable="true"/>
</EntityType>
<EntityContainer Name="generatedkeys" m:IsDefaultEntityContainer="true">
  <EntitySet Name="MyView" EntityType="sample.odata.generatedkeys.MyViewType"/>
</EntityContainer>
```

As a consequence of the transient nature of generated local keys, it is not possible to define navigation properties on these entities or use them in filter or order by conditions.

6.1.1.4.6 OData Associations

You can define associations between entities to express relationships between entities. With associations it is possible to reflect foreign key constraints on database tables, hierarchies and other relations between database objects. OSDL supports simple associations, where the information about the relationship is stored in one of the participating entities, and complex associations, where the relationship information is stored in a separate association table.

Associations themselves are freestanding. On top of them you can specify which of the entities participating in the relationship can navigate over the association to the other entity by creating `NavigationProperty` objects.

For the examples used to illustrate OData associations, we use the tables `customer` and `order`:

Table definition: `sample.odata:customer.hdbtable`

```
COLUMN TABLE "sample.odata::customer" (
    "ID" INTEGER NOT NULL,
    "OrderID" INTEGER,
    PRIMARY KEY ("ID")
);
```

Table definition: `sample.odata:order.hdbtable`

```
COLUMN TABLE "sample.odata::order" (
```

```
    "ID" INTEGER NOT NULL,
    "CustomerID" INTEGER,
    PRIMARY KEY ("ID")
);
```

There is one relationship `order.CustomerID` to `customer.ID`.

Simple Associations

The definition of an association requires you to specify a name, which references two exposed entities and whose columns keep the relationship information. To distinguish the ends of the association, you must use the keywords `principal` and `dependent`. In addition, it is necessary to denote the multiplicity for each end of the association.

Service definition: `sample.odata:assocsimple.xsodata`

```
service {
    "sample.odata::customer" as "Customers";
    "sample.odata::order" as "Orders";
    association "Customer_Orders" with referential constraint principal
"Customers"("ID") multiplicity "1" dependent "Orders"("CustomerID") multiplicity
"*";
}
```

The association in the example above with the name `Customer_Orders` defines a relationship between the table `customer`, identified by its `EntitySet` name `Customers`, on the `principal` end, and the table `order`, identified by its entity set name `Orders`, on the `dependent` end. Involved columns of both tables are denoted in braces ({}) after the name of the corresponding entity set. The `multiplicity` keyword on each end of the association specifies their cardinality - in this example, one-to-many.

The `with referential constraint` syntax ensures that the referential constraint check is enforced at design time, for example, when you activate the service definition in the SAP HANA repository. The referential constraint information appears in the metadata document.

> **i** Note
>
> SAP strongly recommends that you use the `with referential constraint` syntax.

The number of columns involved in the relationship must be equal for both ends of the association, and their order in the list is important. The order specifies which column in one table is compared to which column in the other table. In this simple example, the column `customer.ID` is compared to `order.CustomerID` in the generated table join.

As a result of the generation of the service definition above, an `AssociationSet` named `Customer_Orders` and an `Association` with name `Customer_OrdersType` are generated:

`http://<myHANAServer>:<port>/odata/services/<myService>.xsodata/$metadata`

```
<EntityType Name="CustomersType">
   <Key>
      <PropertyRef Name="ID"/>
   </Key>
   <Property Name="ID" Type="Edm.Int32" Nullable="false"/>
   <Property Name="RecruitID" Type="Edm.String" Nullable="true" MaxLength="1"/>
</EntityType>
<EntityType Name="OrdersType">
   <Key>
      <PropertyRef Name="ID"/>
   </Key>
   <Property Name="ID" Type="Edm.Int32" Nullable="false"/>
   <Property Name="CustomerID" Type="Edm.Int32" Nullable="false"/>
</EntityType>
<Association Name="Customer_OrdersType">
   <End Type="sample.odata.assocsimple.CustomersType" Role="CustomersPrincipal" Multiplicity="1"/>
   <End Type="sample.odata.assocsimple.OrdersType" Role="OrdersDependent" Multiplicity="*"/>
   <ReferentialConstraint>
      <Principal Role="CustomersPrincipal">
         <PropertyRef Name="ID"/>
      </Principal>
      <Dependent Role="OrdersDependent">
         <PropertyRef Name="CustomerID"/>
      </Dependent>
   </ReferentialConstraint>
</Association>
<EntityContainer Name="assocsimple" m:IsDefaultEntityContainer="true">
   <EntitySet Name="Customers" EntityType="sample.odata.assocsimple.CustomersType"/>
   <EntitySet Name="Orders" EntityType="sample.odata.assocsimple.OrdersType"/>
   <AssociationSet Name="Customer_Orders"
   Association="sample.odata.assocsimple.Customer_OrdersType">
      <End Role="CustomersPrincipal" EntitySet="Customers"/>
      <End Role="OrdersDependent" EntitySet="Orders"/>
   </AssociationSet>
</EntityContainer>
```

The second association is similar to the first one and is shown in the following listing:

```
 association "Customer_Recruit" with referential constraint principal
"Customers"("ID") multiplicity "1" dependent "Customers"("RecruitID")
multiplicity "*";
```

Complex Associations

For the following example of a complex association, an additional table named `knows` is introduced that contains a relationship between customers.

Table definition: `sample.odata:knows.hdbtable`

```
COLUMN TABLE "sample.odata::knows" (
    "KnowingCustomerID" INTEGER NOT NULL,
    "KnowCustomerID" INTEGER NOT NULL,
    PRIMARY KEY ("KnowingCustomerID","KnowCustomerID")
);
```

Relationships that are stored in association tables such as `knows` can be similarly defined as simple associations. Use the keyword `over` to specify the additional table and any required columns.

Service definition: `sample.odata:assoccomplex.xsodata`

```
service {
    "sample.odata::customer" as "Customers";
    "sample.odata::order" as "Orders";
    association "Customer_Orders"
        principal "Customers"("ID") multiplicity "*"
        dependent "Customers"("ID") multiplicity "*"
        over "sample.odata::knows" principal ("KnowingCustomerID") dependent
("KnownCustomerID");
}
```

With the keywords `principal` and `dependent` after `over` you can specify which columns from the association table are joined with the `principal` respectively dependent columns of the related entities. The number of columns must be equal in pairs, and their order in the list is important.

The generated `Association` in the metadata document is similar to the one created for a simple association except that the `ReferentialConstraint` is missing:

`http://<myHANAServer>:<port>/odata/services/<myService>.xsodata/$metadata`

```
<Association Name="Customer_OrdersType">
  <End Type="sample.odata.assoccomplex.CustomersType" Role="CustomersPrincipal" Multiplicity="*"/>
  <End Type="sample.odata.assoccomplex.CustomersType" Role="CustomersDependent" Multiplicity="*"/>
</Association>
<EntityContainer Name="assoccomplex" m:IsDefaultEntityContainer="true">
  <EntitySet Name="Customers" EntityType="sample.odata.assoccomplex.CustomersType"/>
  <EntitySet Name="Orders" EntityType="sample.odata.assoccomplex.OrdersType"/>
  <AssociationSet Name="Customer_Orders"
  Association="sample.odata.assoccomplex.Customer_OrdersType">
    <End Role="CustomersPrincipal" EntitySet="Customers"/>
    <End Role="CustomersDependent" EntitySet="Customers"/>
  </AssociationSet>
</EntityContainer>
```

Navigation Properties

By only defining an association, it is not possible to navigate from one entity to another. Associations need to be bound to entities by a `NavigationProperty`. You can create them by using the keyword `navigates`:

Service definition: `sample.odata:assocnav.xsodata`

```
service {
    "sample.odata::customer" as "Customers" navigates ("Customer_Orders" as
"HisOrders");
    "sample.odata::order" as "Orders";
    association "Customer_Orders" principal "Customers"("ID") multiplicity "1"
dependent "Orders"("CustomerID") multiplicity "*";
}
```

The example above says that it is possible to navigate from `Customers` over the association `Customer_Order` via the `NavigationProperty` named "`HisOrders`".

The right association end is determined automatically by the entity set name. But if both ends are bound to the same entity, it is necessary to specify the starting end for the navigation. This is done by specifying either `from principal` or `from dependent` which refer to the `principal` and `dependent` ends in the association.

Service definition: `sample.odata:assocnavself.xsodata`

```
service {
    "sample.odata::customer" as "Customers"
        navigates ("Customer_Orders" as "HisOrders","Customer_Recruit" as
"Recruit" from principal);
    "sample.odata::order" as "Orders";
    association "Customer_Orders" principal "Customers"("ID") multiplicity "1"
dependent "Orders"("CustomerID") multiplicity "*";
        association "Customer_Recruit" principal "Customers"("ID") multiplicity
"1" dependent "Customers"("RecruitID") multiplicity "*";
}
```

In both cases a `NavigationProperty` is added to the `EntityType`.

`http://<myHANAServer>:<port>/odata/services/<myService>.xsodata/$metadata`

```
<EntityType Name="CustomersType">
  <Key>
    <PropertyRef Name="ID"/>
  </Key>
  <Property Name="ID" Type="Edm.Int32" Nullable="false"/>
  <Property Name="RecruitID" Type="Edm.String" Nullable="true" MaxLength="1"/>
  <NavigationProperty Name="HisOrders"
  Relationship="sample.odata.assocnavself.Customer_OrdersType" FromRole="CustomersPrincipal"
  ToRole="OrdersDependent"/>
  <NavigationProperty Name="Recruit"
  Relationship="sample.odata.assocnavself.Customer_RecruitType" FromRole="CustomersPrincipal"
  ToRole="CustomersDependent"/>
</EntityType>
```

6.1.1.4.7 OData Aggregation

The results of aggregations on columns change dynamically depending on the grouping conditions. This means that aggregation cannot be performed in SQL views; it needs to be specified in the OData service definition itself. Depending on the type of object to expose, you need to explicitly specify the columns to aggregate and the function to use or derived them from metadata in the database.

In general, aggregations do not have consequences for the metadata document. It just effects the semantics of the concerning properties during runtime. The grouping condition for the aggregation contain all selected non-aggregated properties. Furthermore, aggregated columns cannot be used in `$filter`, and aggregation is only possible with generated keys.

Derived Aggregation

The simplest way to define aggregations of columns in an object is to derive this information from metadata in the database. The only objects with this information are calculation views and analytic views. For all other

object types, for example, tables and SQL views, the activation will not work. To cause the service to use derived information, you must specify the keywords *aggregates always*, as illustrated in the following example:

```
service {
    "sample.odata::calc" as "CalcView"
        keys generate local "ID"
        aggregates always;
}
```

Explicit Aggregation

The example for the explicit aggregation is based on the following table definition: `sample.odata:revenues.hdbtable`

```
COLUMN TABLE "sample.odata::revenues" (
    "Month" INTEGER NOT NULL,
    "Year" INTEGER NOT NULL,
    "Amount" INTEGER,
    PRIMARY KEY ("Month","Year")
);
```

You can aggregate the columns of objects (without metadata) that are necessary for the derivation of aggregation by explicitly denoting the column names and the functions to use, as illustrated in the following example of a service definition: `sample.odata:aggrexpl.xsodata`

```
service {
    "sample.odata::revenues" as "Revenues"
        keys generate local "ID"
        aggregates always (SUM of "Amount");
}
```

The results of the entity set `Revenues` always contain the aggregated value of the column `Amount`. To extract the aggregated revenue amount per year, add `$select=Year,Amount` to your requested URI.

6.1.1.4.8 OData Parameter Entity Sets

SAP HANA calculation views can interpret input parameters. For OData, these parameters can be entered by using a special parameter *entity set*.

Parameter entity sets can be generated for calculation views by adding *parameters via entity* to the entity, as illustrated in the following service-definition example:

```
service {
    "sample.odata::calc" as "CalcView"
        keys generate local "ID"
        parameters via entity;
}
```

During loading of the service, parameters specified in `sample.odata/calc.calculationview` are retrieved from the metadata of the calculation view and exposed as a new `EntitySet` named after the entity set name and the suffix `Parameters`, for example, `CalcViewParameters`. A `NavigationProperty` named `Results` is generated to retrieve the results from the parameterized call.

The name of the generated parameter entity set and the navigation property can be customized, as illustrated in the following service-definition example:

```
service {
    "sample.odata::calc" as "CalcView"
        keys generate local "ID"
        parameters via entity "CVParams" results property "Execute";
}
```

With the definition above, the name of the parameter entity set is `CVParams`, and the name of the `NavigationProperty` for the results is `Execute`.

Navigating to Entities via Parameters

In an OData service definition, you can enable navigation between an entity and a parameterized entity. This feature is particularly useful if you need to have access to individual entries in a parameterized entity set, for example, a calculation view with parameters. If you need to access individual entries in an entity set that has parameters, you must expose the parameters as keys. If you do not need to have access to individual entries in an entity set, you can use the `key generate local` option to generate a pseudo key.

To enable navigation between an entity and a parameterized entity, you must perform the following steps:

1. Specify the parameters as part of the key of the target entity
2. Define the association between the entities

Enabling navigation between an entity and a parameterized entity is only possible if the parameters are part of the entity-type key in the OData service definition file. To make the parameters part of the key of the target entity, use the `via key` syntax, as illustrated in the following example:

```
service {
   "sap.test::calcview" key ("theKeyColumns") parameters via key and entity;
}
```

You also have to define an **association** between the source and target entities, for example, with additional entries introduced by the `via parameters` keyword, as illustrated in the following example:

```
service {
   "sap.test::table" as "Tab" navigates ("avp" as "ViewNav");
   "sap.test::calcview" as "View" key ("theKeyColumns") parameters via key and
entity;

   association via parameters "avp"
     principal "Tab"("paramValue") multiplicity "*"
     dependent "View"("parameter") multiplicity "*";
}
```

> **i** Note
>
> The order of the property list of the dependent end is crucial.

The parameters you define in the dependent end of the association **must** be the first properties in the list. In addition, the parameters specified **must** be given in the same order as they are specified in the view, as illustrated in the following example:

```
association via parameters "avp"
```

```
    principal "Tab"("col1", "col2", "col3") multiplicity "*"
    dependent "View"("parameter1", "parameter2", "colA") multiplicity "*";
```

In the example immediately above, the principal "`Tab`" has three columns that contain the information that is required to navigate to the dependent "`View`" in the association.

- "`col1`"
  The value of "`col1`" should be set for "`parameter1`"
- "`col2`"
  The value of "`col2`" should be set for "`parameter2`"
- "`col3`"
  The parameter "`col3`" contains additional information that is not passed as an input parameter, but as part of a `WHERE` condition.

The generated SQL statement would look like the following:

```
select ... from "sap.test::calcview"(placeholder."$$parameter1$$"=>?,
placeholder."$$parameter2$$"=>?)
                   where "colA"=?
```

> **Note**
>
> This navigation property cannot be used in combination with the OData query options `$expand`, `$filter` and `$orderby`.

6.1.1.4.9 OData ETag Support

This feature allows a service to define the fields that are to be included in the concurrency check.

You can now use entity tags (ETags) for optimistic concurrency control. If you choose to use this feature, then you must enable it per entity in the `.xsodata` file. Enabling this feature per entity allows for the concurrency control to be applied to multiple fields. The following code example provides information about how to do this.

> **Sample Code**
>
> ```
> service
> { entity "sap.test.odata.db.views::Etag" as "EtagAll"
>   key ("KEY_00") concurrencytoken;
>  entity "sap.test.odata.db.views::Etag" as "EtagNvarchar"
>   key ("KEY_00") concurrencytoken ("NVARCHAR_01","INTEGER_03");
> }
> ```

If you specify `concurrencytoken` only, then all properties, except the key properties, are used to calculate the ETag value. If you provide specific properties, then only those properties are used for the calculation.

> **Note**
>
> You **cannot** specify `concurrencytoken` on aggregated properties that use the `AVG` (average) aggregation method.

6.1.1.4.10 OData Nullable Properties

You can create a service to enable nullable properties in OData.

During the 'Create' phase, the XSODATA layer generates all entity properties automatically. Since the properties are not nullable, consumers of the code are forced to pass dummy values into them. However, OData supports `$filter` and `$orderby` conditions on the `null` value. This means that it is now possible to treat `null` as a value, if you enable it. You can enable this behavior for the entire service only, not per entity.

The following code example provides information about how you can do this.

Sample Code

```
service {
 …
 }
settings {
  support null;
  content cache-control "no-store";
  metadata cache-control "max-age=86401,must-revalidate";
  hints
      "NO_CALC_VIEW_UNFOLDING";
  limits
      max_records = 10,
      max_expanded_records = 30;
}
```

If you enable support for `null`, then `$filter` requests, such as `$filter=NVARCHAR_01 eq null`, are possible. Otherwise `null` is rejected with an exception. If you do not enable the `null` support, then the default behavior applies.

Note

`null` values are “ignored” in comparisons: "ignored" in the sens that if you compare a column with `null` and the columns contain per definition no `null` values, no record passes the filter.

Related Information

OData Configurable Cache Settings [page 557]
OData Hints for SQL Select Statements [page 559]
OData Entity Limits [page 560]

6.1.1.4.11 OData Configurable Cache Settings

You can create a service to configure the cache settings for the `$metadata` request to optimize performance.

When calling OData services, the services make repeated requests for the `$metadata` document. Since changes to the underlying entity definitions occurs rarely, SAP has enabled the option to configure caching for

these `$metadata` documents. By configuring the cache, you can avoid many redundant queries to process the metadata.

The following code example provides information about how you can do this.

Sample Code

```
service {
 ...
}
settings {
  support null;
  content cache-control "no-store";
  metadata cache-control "max-age=86401,must-revalidate";
  hints
      "NO_CALC_VIEW_UNFOLDING";
  limits
      max_records = 10,
      max_expanded_records = 30;
}
```

content cache-control

You can use the `content cache-control` parameter to set the HTTP header "value" that is used for cache control in the data responses, for example:

```
content cache-control "no-store";
```

The value you specify must be enclosed in double quotes (for example, "`<value>`"), and multiple parameters must be separated by a comma.

→ Tip

You can include any value supported by the HTTP specification for cache-control. `"no-store"` indicates that the cache should not store any details of the client request or server response.

metadata cache-control

You can use the `metadata cache-control` parameter to set the header HTTP "value" that is used for the cache control in the metadata response, for example:

```
metadata cache-control "max-age=86401,must-revalidate";
```

The value you specify for `metadata cache-control` must be enclosed in double quotes (for example, "`<value>`"), and multiple elements must be separated by a comma, as illustrated in the example above.

→ Tip

You can include any value supported by the HTTP specification for cache-control. In the example, above, `"must-revalidate"` indicates that the cache must verify the status of resources (fresh or stale) and not

use any stale resource whose validity has expired; `"max-age"` specifies the amount of time a resource is considered fresh (and still usable).

Related Information

OData Nullable Properties [page 557]
OData Hints for SQL Select Statements [page 559]
OData Entity Limits [page 560]

6.1.1.4.12 OData Hints for SQL Select Statements

Provide specific hints for SQL select statements in your OData service.

You can use the `hints` property to add elements to the SQL `SELECT` statement the OData service uses to query data.

⚠ Caution

Bear in mind that setting the wrong hint can result in unexpected data results. And if you remove the (default) `NO_CALC_VIEW_UNFOLDING` hint explicitly (for example, to improve performance), it is important to check the results of the resulting calculation views for accuracy and correctness.

Sample Code

OData Service Hints Settings

```
service {
 …
}
settings {
  support null;
  content cache-control "no-store";
  metadata cache-control "max-age=86401,must-revalidate";
  hints
      "NO_CALC_VIEW_UNFOLDING", "ABC";
  limits
      max_records = 10,
      max_expanded_records = 30;
}
```

If no `hints` are specified, the default hint '`NO_CALC_VIEW_UNFOLDING`' is still added to the SQL `SELECT` statements used in the service query, for example, `select * from with hint (NO_CALC_VIEW_UNFOLDING)`. This is the default behavior and is required for backward compatibility.

If SQL hints are specified, for example, '`hints null`' or any other value such as '`hints ABC`', then the SQL parameter '`NO_CALC_VIEW_UNFOLDING`' is **not** added to the SQL `SELECT` statement automatically. For '`hints null`', no hint statement is included in the SQL `SELECT` statement used by the OData service.

→ Tip

It is possible to add the hint `"NO_CALC_VIEW_UNFOLDING"` manually, and multiple hints can be added by separating each hint with a comma. For example, the setting '`hints "NO_CALC_VIEW_UNFOLDING",`

"ABC"' produces the SQL statement 'select * from with hint (NO_CALC_VIEW_UNFOLDING, ABC)'.

Related Information

OData Nullable Properties [page 557]
OData Configurable Cache Settings [page 557]
OData Entity Limits [page 560]

6.1.1.4.13 OData Entity Limits

Restrict the amount of entities that an OData service can load from the database.

You can use the `limits` property in `settings` to restrict the amount of entities loaded from the database by an OData service. The `limits` setting is useful if you need to avoid problems with memory shortages and want to reduce the size of the data included in the response to an OData service query.

Sample Code

OData Service Hints Settings

```
service {
 …
}
settings {
  support null;
  content cache-control "no-store";
  metadata cache-control "max-age=86401,must-revalidate";
  hints
      "NO_CALC_VIEW_UNFOLDING";
  limits
      max_records = 10,
      max_expanded_records = 30;
}
```

max_records

You can use the property `max_records` to restrict the number of entities addressed by the resource path. If there are more records selected than allowed with `max_records` then an error (400 Bad request) is returned to the client. To avoid the problem of bad requests, the client can use `$top` and `$skip` to select less than the number of entities specified in `max_records`.

Note

The default value for `max_records` is “1000”. If this value is **not** suitable for your application, add the appropriate setting to the application's xsodata service file, for example, `max_records = 25;`.

max_expanded_records

You can use `max_expanded_records` to restrict the number of entries of a single expanded navigation property. If there are more records selected than allowed with `max_expanded_records`, then an error (400 Bad request) is returned to the client. With OData V2 its not possible to apply `$top` or `$skip` to an expanded navigation property, which means that the client cannot avoid this error alone solely for the expanded navigation property.

However, if a feed is returned then reducing the entries in the feed with `$top` and `$skip` or `$filter` might help to reduce the number of entries collected with expanded navigation properties. Bear in mind that expanded records are counted per navigation property and level, so for `max_expanded_records = 100`, there may be 1 root entity with 100 expanded items or, alternatively, 2 root entities one of which has 25 and the other 75 expanded items.

> **i** Note
>
> The default value for `max_expanded_records` is “10000”. If this value is **not** suitable for your application, add the appropriate setting to the applications xsodata service file, for example, `max_expanded_records = 250;`.

Related Information

OData Nullable Properties [page 557]
OData Configurable Cache Settings [page 557]
OData Hints for SQL Select Statements [page 559]

6.1.1.4.14 Custom Exits for OData Requests

Define validation checks or modification operations during an XS OData write request.

The xsodata library supports custom exits to enable the application to define JavaScript functions which are called while processing the OData request. These exits can be implemented either as a JavaScript function or as a stored procedure. You can define exits for creating, updating, and deleting entity sets as well as for the database commit operation (for example, pre-commit and post-commit). It is not possible to define an exit for simple read requests.

> **⥱** Sample Code
>
> Exit definition in the xsodata Service-Definition file

```
service {
  "my_name_space::customers" as "Customers"
     keys("CustomerID")
     navigates("CustomerToProduct" as "Product" from principal)
     create forbidden
     update using "sap.abc.xyz:myExits.xsjslib::updateEntity"
       events( before "sap.abc.xyz:myLoggingInfrastructure.xsjslib::log" )
     delete forbidden;
```

```
}
```

Exit Types

The following type of write exits are supported for OData write requests in SAP HANA XS:

- Validation Exit:
  Validation exits allow the application to validate data either before or after the data is changed in the database. Any error returned from the application exit function stops any further processing of the request and a rollback is performed. For batch requests, the processing of further requests in the same change set is stopped.
- Modification Exit
  Modification exits enable you to define custom logic that can be used to create, update, or delete an entry in an entity set. If defined, a modification exit is is executed instead of the generic actions provided by the OData infrastructure. You use the `using` keyword to register the exit.

XS OData Validation Exits

Write Request	Code Example	Explanation
Create	`create events (before "<custom exit 1>", after "<custom exit 2>", ...)`	The xsodata library calls the exit custom exit 1 before creating the entity, and the exit custom exit 2 after creating the entity
	`create events (precommit "<custom exit 1>", postcommit "<custom exit 2>", ...)`	The xsodata library calls the exit custom exit 1 before committing the data, and the exit custom exit 2 after committing
	`create forbidden`	Prohibits any create operation on the entity set
Update	`update events (before "<custom exit 1>", after "<custom exit 2>", ...)`	The xsodata library calls the exit custom exit 1 before creating the entity, and the exit custom exit 2 after updating the entity
	`update events (precommit "<custom exit 1>", postcommit "<custom exit 2>", ...)`	The xsodata library calls the exit custom exit 1 before committing the data, and the exit custom exit 2 after committing
	`update forbidden`	Prohibits any update operation on the entity set
Delete	`delete events (before "<custom exit 1>", after "<custom exit 2>", ...)`	The xsodata library calls the exit custom exit 1 before creating the entity, and the exit custom exit 2 after deleting the entity
	`delete events (precommit "<custom exit 1>", postcommit "<custom exit 2>", ...)`	The xsodata library calls the exit custom exit 1 before committing the data, and the exit custom exit 2 after committing

Write Request	Code Example	Explanation
	`delete forbidden`	Prohibits the deletion of the entity

The following table lists the types of modification exits you can define with XS Odata:

XS OData Modification Exits

Write Request	Code Example	Explanation
Create	`create using "<custom exit>"`	The xsodata library calls the exit `<custom exit>` instead of creating the entity directly
Update	`update using "<custom exit>"`	The xsodata library calls the exit `<custom exit>` instead of updating the entity directly
Delete	`delete using "<custom exit>"`	The xsodata library calls the exit `<custom exit>` instead of deleting the entity directly

XS Odata also allows you to combine both validation exits and modification exits in the same request, as illustrated in the following example:

Sample Code

```
create using "<custom exit>" events(before "<custom exit 1>", after "<custom exit 2>", ...)
```

Exit Execution Order for XS OData Requests

For **single** OData requests, the following execution order applies for the custom exit:

Exit Execution Order for Single OData Requests

Odata Request Element	Action	Exit Execution Order
Request #	Process the request	1. `before` 2. `using` (or implicit update happens) 3. `after` 4. `precommit` 5. Data is committed to the database 6. `postcommit`

The following table shows the execution order applied to custom exits for **batch** OData requests:

Note

This example assumes that there are two (2) requests and the batch request also includes a change set with 2 requests.

Exit Execution Order for OData Batch Requests

OData Batch-Request Element	Action	Exit Execution Order
Request 1	Process the request	Single OData request execution order
Change set	Process request 1 in the change set	1. `before` 2. `using` 3. `after`
	Process request 2 in the change set	1. `before` 2. `using` 3. `after`
	Process the commit operation	1. `precommit` of the first request 2. `precommit` of the second request 3. Commit the requested data 4. `postcommit` of the first request 5. `postcommit` of the second request
Request 2	Process request 2	Single OData request execution order

Exit Implementation

You can choose to perform the exit in any of the following ways:

- JavaScript Function [page 564]
- Stored procedure [page 566]

XS Odata Custom Exit as an XS JavaScript Function

The JavaScript function needs to be implemented within an XS JavaScript library (`*.xsjslib` artifact). There is no need to use the export mechanism of Node.js; a plain function is all you need, as illustrated in the following example:.

Sample Code

```
//myLoggingInfrastructure.xsjslib

function my_create_before_exit(param) {
}
```

The input parameter param is an object with the following content:

1. `connection`
   The SQL connection used in the OData request
2. `beforeTableName`

The name of a temporary table with the single entry **before** the operation (UPDATE and DELETE events only)

3. `afterTableName`
   The name of a temporary table with the single entry **after** the operation (CREATE and UPDATE events only)

The content of the temporary tables can be fetched using the available SQL connection as illustrated in the following example:

Sample Code

```
var stmt = 'select * from "'+ param.afterTableName + '"',
    xStmt = param.connection.prepareStatement( stmt ).executeQuery(),
    data = xStmt._rows[0]; // {col1:val1, col2:val2, ...}
```

For the custom exit function, there are two possibilities to handle an error:

1. Throw an error
   Use the function `throw "error message";`.

   Note

   The response has the status code “`500`”, which cannot be modified.

2. Return an object
   Return an object with the following structure:

   Sample Code

   ```
   return{
      HTTP_STATUS_CODE: code, // e.g. 400, 500, etc.
      ERROR_MESSAGE: 'error message',
      DETAILS: 'more details'
   };
   ```

   Note

   This method enables you to add more details and control the status code of the response.

Within the xsodata files, exit functions are referenced with a colon (:) separated string, for example: `<directory>:<file>::function` .

Sample Code

```
sap.abc.xyz:myLoggingInfrastructure.xsjslib::my_create_before_exit
```

- `sap.abc.xyz`
  References a directory
- `myLoggingInfrastructure.xsjslib`
  References a XS JavaScript library (`xsjslib` file)
- `my_create_before_exit`
  References the exit function

The following example shows an XS JavaScript startup configuration with two root folders `/_SYS_REPO` and `/my_content`:

Sample Code

```
//Start xsjs listener
xsjs({
   compress: false,
   rootDirs: [
      path.join(__dirname, '_SYS_REPO')
      path.join(__dirname, 'my_content')
   ],
   ...}).listen(...)
```

In the example above, the file `myLoggingInfrastructure.xsjslib` must be stored in one of the following locations:

- `<app directory>/_SYS_REPO/sap/abc/xyz/myLoggingInfrastructure.xsjslib`
- `<app directory>/my_content/sap/abc/xyz/myLoggingInfrastructure.xsjslib`

XS Odata Custom Exit as an SQLScript Stored Procedure

If you register a custom exit for an OData write request in the form of an SQLScript procedure, the signature of the registered script must follow specific rules, for example, depending on whether it is registered for *entity* or *link* write operations and depending on the operation itself. The signature must also have table-typed parameters for both input and output:

- Entity Write Operations
- Link Write Operations

For **entity** write operations, the methods registered for the CREATE operation are passed a table containing the new entry that must be inserted into the target table; the UPDATE operation receives the entity both before and after the modification; the DELETE operation receives the entry that must be deleted. The table type of the parameters (specified with the *EntityType* keyword in the table below) corresponds to the types of the exposed entity.

Entity Write Operations

Script Type	Create	Update	Delete
before, after, precommit, using	IN new EntityType, OUT error ErrorType	IN new EntityType, IN old EntityType, OUT error ErrorType	IN old EntityType, OUT error ErrorType
postcommit	IN new EntityType	IN new EntityType, IN old EntityType	IN old EntityType

For **link** write operations, all exits that are executed before the commit operation take two table-typed input parameters and one table-typed output parameter. The first parameter must correspond to the structure of the entity type at the **principal** end of the association; the second parameter must correspond to the **dependent** entity type.

Link Write Operations

Script Type	Create, Update, Delete
before, after, precommit, using	IN principal PrincipalEntityType, IN dependent DependentEntityType, OUT error ErrorType
postcommit	IN principal PrincipalEntityType, IN dependent DependentEntityType

> **Note**
>
> Parameter types (`IN`, `OUT`) are checked during activation; the data types of table type columns are **not** checked.

The `OUT` parameter enables you to return error information. The first row in the OUT table is then serialized as `inner error` in the error message. If no error occurs, the OUT table must remain empty. The structure of the table type `ErrorType` is not restricted. Any columns with special names `HTTP_STATUS_CODE` and `ERROR_MESSAGE` are mapped to common information in the OData error response. Content of columns with other names are serialized into the `inner error` part of the error message that allows the return of custom error information.

Error Message Content

Column Name	Type	Value Range	Error Response Information
HTTP_STATUS_CODE	INTEGER	400-417 (default: 400)	The HTTP response status code
ERROR_MESSAGE	NVARCHAR		The error message (`<message>`)

> **Note**
>
> If the SQLScript procedure throws an exception or writes an error messages to the `OUT` parameter table, the OData write operation is aborted. If more than one error message is added, only the content of the first row is returned in the resulting error message. Any scripts registered for the `postcommit` event must not have an `OUT` parameter as the write operation cannot be aborted at such a late stage, even in the event of an error.

The following example illustrates a typical error-type table type, which is defined in a design-time file that must have the `.hdbtabletype` file suffix, for example `error.hdbtabletype`:

```
"sample.odata::error" AS TABLE (
    "HTTP_STATUS_CODE" INTEGER,
    "ERROR_MESSAGE" NVARCHAR(100),
    "DETAIL"  NVARCHAR(100)
)
```

The following example shows how information is extracted from the error table if an error occurs during the execution of a create procedure for an OData write operation:

```
create procedure "sample.odata::createmethod"(IN new "sample.odata::table", OUT
error "sample.odata::error")
language sqlscript
sql security invoker as
     id INT;
begin
     select ID into id from :new;
     if :id < 1000 then
          error = select 400 as http_status_code,
          'invalid ID' error_message,
          'value must be >= 1000' detail from dummy;
     else
          insert into "sample.odata::table" values (:id);
     end if;
end;
```

Related Information

Tutorial: Creating a Validation Exit with SQLScript [page 568]
Tutorial: Creating a Modification Exit with XS JavaScript [page 571]

6.1.1.4.15 Tutorial: Creating a Validation Exit with SQLScript

Use SQLScript to create a custom validation exit which runs server-side verification and data-consistency checks for an OData **update** operation.

Prerequisites

To perform this task, you need the following objects:

- A table to expose, for example, `sample.odata::table.hdbtable`
- An error type, for example, `sample.odata::error.hdbtabletype`

Context

In this tutorial, you see how to register an SQL script for an OData update operation; the script verifies, before the execution of the update operation, that the updated value is larger than the previous one. In the example shown in this tutorial, you define the table to be updated and a table type for the error output parameter of the exit procedure.

Procedure

1. Create a table definition file using `.hdbtable` syntax.

   The table to expose is defined in `sample.odata:table.hdbtable`, which should look like the following example:

   ```
   COLUMN TABLE "table" (
       "ID" INTEGER NOT NULL,
       PRIMARY KEY ("ID")
   );
   ```

2. Create a table type for the error output parameter of the exit procedure.

   The error type file `sample.odata:error.hdbtabletype` should look like the following example:

   ```
   "sample.odata::error" AS TABLE (
       "HTTP_STATUS_CODE" INTEGER,
       "ERROR_MESSAGE" NVARCHAR(100),
   ```

```
    "DETAIL"  NVARCHAR(100)
)
```

3. Create a procedure that runs before the UPDATE event.

   The procedure script for the `before UPDATE` event must have two table input parameters and one output parameter, for example:

   - `IN new "sample.odata::table"`
   - `IN old "sample.odata::table"`
   - `OUT error "sample.data::error"`

   The procedure `sample.odata:beforeupdate.hdbprocedure` would look like the following example:

```
procedure "sample.odata::beforeupdate"
     (IN new "sample.odata::table", IN old "sample.odata::table", OUT error
"sample.odata::error")
language sqlscript
sql security invoker as
     idnew INT;
     idold INT;
begin
     select ID into idnew from :new;
     select ID into idold from :old;
if :idnew <= :idold then
error = select 400 as http_status_code,
     'invalid ID' error_message,
     'the new value must be larger than the previous' detail from dummy;
end if;
end;
```

4. Register the procedure to be executed at the `before` event.

   You use the `update events (before "...")` keywords to register the procedure, as illustrated in the following example of an OData service file:

```
service {
     "sample.odata::table"
          update events (before "sample.odata::beforeupdate");
}
```

6.1.1.4.16 Tutorial: Creating a Validation Script with XS JavaScript

Use server-side JavaScript to write a script that you can register as a validation exit for an OData update operation for an entity.

Prerequisites

To perform this task, you need the following objects:

- A table to expose for the OData create operation, for example, `sample.odata::table.hdbtable`

Context

SAP HANA XS enables you to register custom code that handles the OData write operation. In this tutorial, you see how to use server-side JavaScript (XSJS) to write a script which you register as a validation exit for OData UPDATE operations for an entity. The script you register can verify the data to insert, continue to perform the intended operation, or return an error. To register an XS JavaScript function as an OData validation exit, perform the following steps:

Procedure

1. Create a table definition file, for example, using the `.hdbtable` syntax.

   The table you create in this step is used in the XS JavaScript function you create later in the tutorial. The table to expose is defined in `sample.odata:table.hdbtable`, which should look like the following example:

   Sample Code

   ```
   COLUMN TABLE "table" (
          "ID" INTEGER NOT NULL,
          PRIMARY KEY ("ID")
   );
   ```

2. Create the XS JavaScript function that you want to register for OData modification events.

   The function you register has one parameter, which can have the properties listed in the following table:

   - `connection`
   - `beforeTableName`
   - `afterTableName`

   ! Restriction

   The XS JavaScript function that you register for OData validation events must be created in the form of an XSJS library, for example, with the file extension `.xsjslib`; the XS JavaScript function you register for OData validation events cannot be an `.xsjs` file.

   The XS JavaScript function `jsexit.xsjslib` should look something like the code illustrated in the following example:

   Sample Code

   XS JavaScript OData Validation Exit `jsexit.xsjslib`

   Sample Code

   ```
   function update_verify (param) {
     var after = param.afterTableName,
       stmt = 'select * from "'+ after + '"',
       pStmt = param.connection.prepareStatement( stmt ),
       xStmt = pStmt.executeQuery(),
       cols = xStmt._columnNames,// [col1, col2, ...]
   ```

```
    data = xStmt._rows[0];// {col1:val1, col2:val2, ...}
  pStmt.close();
  if (data['ID']>50) {
    console.log('** Err');

    return { HTTP_STATUS_CODE: 400, ERROR_MESSAGE: 'ID is beyond the
range', DETAILS: 'ID can not be greater than 50'};
  } else {
     console.log('** Continue to update');
  }
}
```

3. Bind the XS JavaScript function to the entity specified in the OData service definition.

   To bind the XS JavaScript function to a specified entity, use the syntax: `<Package.Path>:<file>.<suffix>::<XSJS_FunctionName>`, as illustrated in the following example:

   Sample Code

   OData Service Definition

```
service {
    "sample.odata::table" as "Table"
    update events (before "sap.test:jsexit.xsjslib::update_verify");
}
```

Related Information

Tutorial: Creating a Validation Exit with SQLScript [page 568]
Tutorial: Creating a Modification Exit with XS JavaScript [page 571]
Defining OData v2 Services for XS Advanced JavaScript Applications [page 524]

6.1.1.4.17 Tutorial: Creating a Modification Exit with XS JavaScript

You can use server-side JavaScript to write a script which you register as a modification exit for an OData **update** operation for an entity.

Prerequisites

To perform this task, bear in mind the following prerequisites:

- A table to expose for the OData create operation, for example, `sample.odata::table.hdbtable`

Context

SAP HANA XS enables you to register custom code that handles the OData write operation. In this tutorial, you see how to use server-side JavaScript (XSJS) to write a script which you register as a modification exit for OData `UPDATE` operations for an entity. The script you register verifies the data to insert, and throws a defined error string in the event of an error, for example, `Could not update table; check access permissions`.

To register an XS JavaScript function as an OData modification exit, perform the following steps:

Procedure

1. Create a table definition file, for example, using the `.hdbtable` syntax.

   The table you create in this step is used in the XS JavaScript function you create later in the tutorial. The table to expose is defined in `sample.odata:table.hdbtable`, which should look like the following example:

   ```
   COLUMN TABLE "table" (
       "ID" INTEGER NOT NULL,
       PRIMARY KEY ("ID")
   );
   ```

2. Create the XS JavaScript function that you want to register for OData modification events.

   > **i** Note
   >
   > The XS JavaScript function that you want to register for OData modification events must be created in the form of an XSJS library, for example, with the file extension `.xsjslib`; the XS JavaScript function cannot be an `.xsjs` file.

   The function you register has one parameter, which can have the properties listed in the following table:

Property	Type	Description
connection	Connection	The SQL connection used in the OData request
beforeTableName	String	The name of a temporary table with the single entry before the operation (UPDATE and DELETE events only)
afterTableName	String	The name of a temporary table with the single entry after the operation (CREATE and UPDATE events only)

   The XS JavaScript function `jsexit.xsjslib` could look like the following example:

   ```
   function update_instead(param) {
      $.trace.debug(“entered function”);
      let before = param.beforeTableName;
      let after = param.afterTableName;
      let pStmt = param.connection.prepareStatement('select * from ”' + after
   +'"');
      // ...
      if (ok) {
          // update
   ```

```
   } else {
        throw “an error occurred; check access privileges”
   }
}
```

3. Bind the XS JavaScript function to the entity specified in the OData service definition.

   To bind the XS JavaScript function to a specified entity, use the syntax: `<Package.Path>:<file>.<suffix>::<XSJS_FunctionName>` as illustrated in the following example:

```
service {
   "sample.odata::table" as "Table" update using
"sap.test:jsexit.xsjslib::update_instead";
}
```

6.1.1.5 OData Service Definition Language Syntax (XS Advanced)

The OData Service Definition Language (OSDL) provides a set of keywords that enable you to set up an ODATA service definition file that specifies what data to expose, in what way, and to whom.

The following list shows the syntax of the OData Service Definition Language (OSDL) in an EBNF-like format; conditions that apply for usage are listed after the table.

```
definition              :=service [annotations]
service                 :='service' [namespace] body
namespace               :='namespace' quotedstring
quotedstring            :=quote string quote
string                  :=UTF8
quote                   :='"'
body                    :='{' content '}'
content                 :=entry [content]
entry                   :=( entity | association ) ';'
entity                  :=object [entityset] [with] [keys] [navigates]
[aggregates] [parameters] [modification]
object                  :=['entity'] ( repoobject | catalogobject )
repoobject              :=quote repopackage '/' reponame '.' repoextension quote
repopackage             :=string
reponame                :=string
repoextension           :=string
catalogobject           :=catalogobjectschema '.' catalogobjectname
catalogobjectschema     :=quotedstring
catalogobjectname       :=quotedstring
entityset               :='as' entitysetname
entitysetname           :=quotedstring
with                    :=( 'with' | 'without' ) propertylist
propertylist            :='(' columnlist ')'
columnlist              :=columnname [',' columnlist]
columnname              :=quotedstring
keys                    :='keys' ( keylist | keygenerated )
keylist                 :=propertylist
keygenerated            :='generate' ( keygenlocal )
keygenlocal             :='local' columnname
navigates               :='navigates' '(' navlist ')'
navlist                 :=naventry [',' navlist]
naventry                :=assocname 'as' navpropname [fromend]
assocname               :=quotedstring
navpropname             :=quotedstring
fromend                 :='from' ( 'principal' | 'dependent' )
aggregates              :='aggregates' 'always' [aggregatestuple]
```

```
aggregatestuple         :='(' aggregateslist ')'
aggregateslist          :=aggregate [',' aggregateslist]
aggregate               :=aggregatefunction 'of' columnname
aggregatefunction       :=( 'SUM' | 'AVG' | 'MIN' | 'MAX' )
parameters              :='parameters' 'via' [parameterskeyand] 'entity'
[parameterentitysetname] [parametersresultsprop]
parameterskeyand        :='key' 'and'
parameterentitysetname :=quotedstring
parametersresultsprop  :='results' 'property' quotedstring
modification            :=[create] [update] [delete]
create                  :='create' modificationspec
update                  :='update' modificationspec
delete                  :='delete' modificationspec
modificationspec        :=( modificationaction [events] | events | 'forbidden' )
modificationaction      :='using' action
action                  :=quotedstring
events                  :='events' '(' eventlist ')'
eventlist               :=eventtype action [',' eventlist]
eventtype               :=( 'before' | 'after' | 'precommit' | 'postcommit' )
association             :=associationdef [assocrefconstraint] principalend
dependentend [( assoctable | storage | modification )]
associationdef          :='association' assocname
assocrefconstraint      :='with' 'referential' 'constraint'
principalend            :='principal' end
dependentend            :='dependent' end
end                     :=endref multiplicity
endref                  :=endtype [joinpropertieslist]
endtype                 :=entitysetname
joinpropertieslist      :='(' joinproperties ')'
joinproperties          :=columnlist
multiplicity            :='multiplicity' quote multiplicityvalue quote
multiplicityvalue       :=( '1' | '0..1' | '1..*' | '*' )
assoctable              :='over' repoobject overprincipalend overdependentend
[modification]
overprincipalend        :='principal' overend
overdependentend        :='dependent' overend
overend                 :=propertylist
storage                 :=( nostorage | storageend [modification] )
nostorage               :='no' 'storage'
storageend              :='storage' 'on' ( 'principal' | 'dependent' )
annotations             :='annotations' annotationsbody
annotationsbody         :='{' annotationscontent '}'
annotationscontent      :=annotationconfig [annotationscontent]
annotationconfig        :='enable' annotation
annotation              :='OData4SAP'
```

> **i** Note
>
> Support for OData annotations is currently not available in SAP HANA XS Advanced.

Conditions

The following conditions apply when using the listed keywords:

1. If the `namespace` is not specified, the schema namespace in the EDMX metadata document will be the repository package of the service definition file concatenated with the repository object name. E.g. if the repository design time name of the `.xsodata` file is `sap.hana.xs.doc/hello.xsodata` the namespace will implicitly be `sap.hana.xs.doc.hello`.
2. `keyslist` must not be specified for objects of type 'table'. They must only be applied to objects referring a view type. `keygenerated` in turn, can be applied to table objects.

3. If the `entityset` is not specified in an entity, the EntitySet for this object is named after the repository object name or the `catalogobjectname`. For example, if `object` is "`sap.hana.xs.doc/odata_docu`", then the `entitysetname` is implicitly set to `odata_docu`, which then can also be referenced in associations.
4. The `fromend` in a `naventry` must be specified if the `endtype` is the same for both the `principalend` and the `dependentend` of an association.
5. The number of `joinproperties` in the `principalend` must be the same as in the `dependentend`.
6. Ordering in the `joinproperties` of `ends` is relevant. The first `columnname` in the `joinproperties` of the `principalend` is compared with the first `columnname` of the `dependentend`, the second with the second, and so on.
7. The `overprincipalend` corresponds to the `principalend`. The number of properties in the `joinproperties` and the `overproperties` must be the same and their ordering is relevant. The same statement is true for the dependent end.
8. `aggregates` can only be applied in combination with `keygenerated`.
9. If `aggregatestuple` is omitted, the aggregation functions are derived from the database. This is only possible for calculation views and analytic views.
10. Specifying `parameters` is only possible for calculation views and analytic views.
11. The default `parameterentitysetname` is the `entitysetname` of the entity concatenated with the suffix "`Parameters`".
12. If the `parametersresultsprop` is omitted, the navigation property from the parameter entity set to the entity is called "`Results`".
13. Support for OData `annotations` is currently under development. For more information about the SAP-specific metadata annotations that become available with the `enable OData4SAP` statement in an `.xsodata` file, see the Related Links below. Note that not all annotations allowed by OData are supported by SAP HANA XS.

Related Information

SAP Annotations for OData
Open Data Protocol

6.1.1.6 OData Service Definition: SQL-EDM Type Mapping (XS Advanced)

During the activation of the OData service definition, the SAP HANA SQL types are mapped to the required OData EDM types according to the rules specified in a mapping table.

The following mapping table lists how SAP HANA SQL types are mapped to OData EDM types during the activation of an OData service definition.

> **i Note**
>
> The OData implementation in SAP HANA XS supports only those SQL types listed in the following table.

SAP HANA SQL to OData EDM Type Mapping

SAP HANA SQL Type	OData EDM Type
Time	Edm.Time
Date	Edm.DateTime
SecondDate	Edm.DateTime
LongDate	Edm.DateTime
Timestamp	Edm.DateTime
TinyInt	Edm.Byte
SmallInt	Edm.Int16
Integer	Edm.Int32
BigInt	Edm.Int64
SmallDecimal	Edm.Decimal
Decimal	Edm.Decimal
Real	Edm.Single
Float	Edm.Single
Double	Edm.Double
Varchar	Edm.String
NVarchar	Edm.String
Char	Edm.String
NChar	Edm.String
Binary	Edm.Binary
Varbinary	Edm.Binary

Example SQL Type Mapping

The following examples shows how SAP HANA SQL types (name, integer, NVarchar) of columns in a table are mapped to the OData EDM types in the properties of an entity type.

SAP HANA SQL:

```
{name = "ID"; sqlType = INTEGER; nullable = false;},
{name = "RefereeID"; sqlType = NVARCHAR; length = 10; nullable = true;}
```

The following example illustrates how the SAP HANA SQL types illustrated in the previous example are mapped to EDM types:

```
<Property Name="ID" Type="Edm.Int32" Nullable="false"/>
<Property Name="RefereeID" Type="Edm.String" Nullable="true" MaxLength="10"/>
```

6.1.1.7 OData Batch Requests (XS Advanced)

The OData standard allows the collection of multiple individual HTTP requests into one single batched HTTP request.

Clients using a defined OData service to consume exposed data can collect multiple, individual HTTP requests, for example, retrieve, create, update and delete (GET, POST, PUT, DELETE), in a single "batch" and send the batched request to the OData service as a single HTTP request. You can compile the batch request manually (by creating the individual requests in the batch document by hand) or automatically, for example, with an AJAX call that adds requests to a queue and loops through the queues to build the batch request. In both cases, the OData standard specifies the syntax required for the header and body elements of a valid batch request document.

SAP HANA XS supports the OData `$batch` feature out-of-the-box; there is nothing to configure in SAP HANA XS to use $batch to perform operations in SAP HANA using an OData service. To understand how the $batch feature works, you need to look at the following phases of the operation:

- Batch Request
- Batch Response

A batch request is split into two parts: the request header and the request body. The body of a batch request consists of a list of operations in a specific order where each operation either retrieves data (for example, using the HTTP `GET` command) or requests a change. A change request involves one or more insert, update or delete operations using the `POST`, `PUT`, or `DELETE` commands.

> **i** Note
>
> A change request must not contain either a **retrieve** request or any nested change requests.

The batch request must contain a `Content-Type` header specifying the value "`multipart/mixed`" and a boundary ID `boundary=batch_#`; the batch boundary ID is then used to indicate the start of each batch request, as illustrated in the following example.

```
POST /service/$batch HTTP/1.1
Host: host
Content-Type: multipart/mixed;boundary=batch_8219-6895         // Define batch ID
--batch_8219-6895                                               // Batch 1 start
  Content-Type: multipart/mixed; boundary=changeset_a4e3-a738 // Define
changeset ID
  --changeset_a4e3-a738                                         // Changeset 1
start
    Content-Type: application/http
    Content-Transfer-Encoding: binary
    [PUT...]

  --changeset_a4e3-a738                                         // Changeset 2
start
    Content-Type: application/http
    Content-Transfer-Encoding: binary
    [POST...]
  --changeset_a4e3-a738--                                      // Changeset (all)
end
--batch_8219-6895                                               // Batch part 2
start
  Content-Type: application/http
  Content-Transfer-Encoding:binary
  [GET...]
--batch_8219-6895--                                             // Batch (all) end
```

Within the batch request, changeset is defined by another boundary ID (for example, `boundary=changeset_123`), which is then used to indicate the start and end of the change requests. The batch request must be closed, too.

> **i** Note
>
> In the following example of a simple OData batch request, some content has been removed to emphasize the structure and layout.

```
POST http://localhost:8002/sap/sample/odata/syntax.xsodata/$batch HTTP/1.1
Host: localhost:8002
Connection: keep-alive
Content-Length: 471
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/30.0.1599.101 Safari/537.36
Cache-Control: no-cache
Content-Type: multipart/mixed; boundary=batch_123
Accept: */*
Accept-Encoding: identity
Accept-Language: en-US,en;q=0.8
x-sap-request-language: en-US
--batch_123
Content-Type:multipart/mixed;boundary=changeset_456
Content-Transfer-Encoding:binary
--changeset_456
Content-Type:application/http
Content-Transfer-Encoding:binary
POST BatchSample HTTP/1.1
Content-Type:application/json
Content-Length:11
{"ID" : 14}
--changeset_456
Content-Type:application/http
Content-Transfer-Encoding:binary
POST BatchSample HTTP/1.1
Content-Type:application/json
Content-Length:11
Accept: application/json
{"ID" : 15}
--changeset_456--
--batch_123--
```

The batch response includes a response for each of the retrieve or change operations included in the corresponding batch request. The order of the responses in the response body must match the order of requests in the batch request. In the context of the batch response, the following is true:

- The response to a retrieve request is always formatted in the same way regardless of whether it is sent individually or as part of batch.
- The body of the collected response to a set of change-requests is one of the following:
  - A response for **all** the successfully processed change requests within the change set, in the correct order and formatted exactly as it would have appeared outside of a batch
  - A single response indicating the failure of the entire change set

The following example shows the form and syntax of the OData batch response to the request illustrated above.

```
HTTP/1.1 202 Accepted
content-type: multipart/mixed; boundary=0CDF14D90919CC8B4A32BD0E0B330DA10
content-length: 2029
content-language: en-US
cache-control: no-cache
expires: Thu, 01 Jan 1970 00:00:00 GMT
```

```
--0CDF14D90919CC8B4A32BD0E0B330DA10
Content-Type: multipart/form-data; boundary=0CDF14D90919CC8B4A32BD0E0B330DA11
Content-Length:      1843
--0CDF14D90919CC8B4A32BD0E0B330DA11
Content-Type: application/http
Content-Length: 1118
content-transfer-encoding: binary
HTTP/1.1 201 Created
Content-Type: application/atom+xml;charset=utf-8
location: http://localhost:8002/sap/sample/odata/syntax.xsodata/BatchSample(14)/
Content-Length: 943
<?xml version="1.0" encoding="utf-8"
standalone="yes"?><entry xml:base="http://localhost:8002/sap/sample/
odata/syntax.xsodata/" xmlns:d="http://schemas.microsoft.com/ado/
2007/08/dataservices" xmlns:m="http://schemas.microsoft.com/ado/2007/08/
dataservices/metadata" xmlns="http://www.w3.org/2005/Atom"><id>http://
localhost:8002/sap/sample/odata/syntax.xsodata/BatchSample(14)</id><title
type="text"></title><author><name /></author><link rel="edit"
title="BatchSample" href="BatchSample(14)"/><link rel="http://
schemas.microsoft.com/ado/2007/08/dataservices/related/Ref" type="application/
atom+xml;type=entry" title="Ref" href="BatchSample(14)/Ref"></
link><category term="sap.sample.odata.syntax.BatchSampleType"
scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" /><content
type="application/xml"><m:properties><d:ID m:type="Edm.Int32">14</d:ID><d:SELFID
m:type="Edm.Int32" m:null="true"></d:SELFID></m:properties></content></entry>
--0CDF14D90919CC8B4A32BD0E0B330DA11
Content-Type: application/http
Content-Length: 427
content-transfer-encoding: binary
HTTP/1.1 201 Created
Content-Type: application/json
location: http://localhost:8002/sap/sample/odata/syntax.xsodata/BatchSample(15)
Content-Length: 271
{"d":{"__metadata": {"uri":"http://localhost:8002/sap/sample/odata/
syntax.xsodata/
BatchSample(15)","type":"sap.sample.odata.syntax.BatchSampleType"},"ID":15,"SELFI
D":null,"Ref":{"__deferred":{"uri":"http://localhost:8002/sap/sample/odata/
syntax.xsodata/BatchSample(15)/Ref"}}}}
--0CDF14D90919CC8B4A32BD0E0B330DA11--
--0CDF14D90919CC8B4A32BD0E0B330DA10--
```

OData Batch Requests in SAPUI5 Applications

If you are developing a UI client using SAPUI5, you can make use of the ODataModel tools to ensure that the data requests generated by the various UI controls bound to an OData service are collected and sent in batches. The SAPUI5 ODataModel toolset includes a large selection of tools you can use to configure the use of the OData batch feature, for example:

- `setUseBatch`
  Enable or disable batch processing for all requests (read and change)
- `addBatchChangeOperations`
  Appends the change operations to the end of the batch stack, which is sent with the `submitBatch` function
- `addBatchReadOperations`
  Appends the read operations to the end of the batch stack, which is sent with the `submitBatch` function
- `submitBatch`
  Submits the collected changes in the batch which were collected via `addBatchReadOperations` or `addBatchChangeOperations`.

Related Information

Open Data Protocol
SAPUI5 ODataModel Reference

6.1.2 Defining OData v4 Services for XS Advanced Java Applications

Set up an OData (v4) service for use by a Java application in an XS advanced.

An OData v4 service for use with an XS advanced Java application is described in a design-time OData artifact and referenced in the corresponding CDS document (by means of an `@OData.publish` annotation) that exposes the specified tables or entities in the annotated context. The definition of your OData service should be placed in the `java/odata/v4/` folder of your Java application project module. The following example shows the project structure for an XS advanced Java application which consumes an OData version 4.0 service.

Example

Sample Code

```
Java-AppName
|- db/
|   |- package.json
|   |- src/
|      |- .hdiconfig
|      \- myDataModel.hdbcds       # @OData v4 annotations
|- web/
|   |- xs-app.json                 # Route for Java Odata v4 module
|   |- package.json
|   \- resources/
|- java/                           # Enable DB/OData v4 support
|   |- package.json
|   |- src/
|   \- odata/v4/                   # OData v4 service resources
|- security/
|   \- xs-security.json            # OData v4 user authorization scopes
\- mta.yaml                        # Requires Java/OData v4 module
```

Related Information

Maintaining OData Services in XS Advanced [page 522]
Tutorial: Create OData v4 Services for XS Advanced Java Applications [page 581]

6.1.2.1 Tutorial: Create OData v4 Services for XS Advanced Java Applications

Step-by-step instructions for creating OData v4 services in an XS advanced Java application.

Context

To create an OData v4 service from an annotated CDS context using tools available in SAP Web IDE for SAP HANA, perform the following steps:

Procedure

1. Create an MTA project.

   For more information about using SAP Web IDE for SAP HANA to set up a Multi-Target Application (MTA), see *Related Links* below.
2. Add an HDB module to the newly created MTA project.

   In SAP Web IDE for SAP HANA, right-click the root folder of the XS advanced MTA project and choose *New* > *SAP HANA Database Module*.
3. Add the required CDS files to the HDB module of your Java MTA.

   You can add CDS files to your MTA's HDB module in either of the following ways:
   - Import existing CDS artifacts from the local file system
   - Create new CDS artifacts manually, for example, using the CDS Graphical Editor.
     In SAP Web IDE for SAP HANA, expand the XS advanced application's DB module, right-click the `src/` folder and choose *New* > *CDS Artifact*. For more information about creating a CDS file, see *Related Links* below.
4. Annotate the required CDS contexts with the annotation "`@OData.publish : true`".

   This step is required to indicate which CDS context must be exposed as an OData v4 service.

   > **Note**
   >
   > You can publish only a child context. If your CDS file contains both a parent as well as a child context, only the child context can be published. For more information, see *Related Links* below.

   > **Sample Code**
   >
   > CDS Context exposed as an OData v4 Service

```
namespace "your"."namespace";
@OData.publish : true
context ContextName{
  entity MyEntity{
    key myID : Integer
    };
```

```
}
```

5. Create a Java module in the same MTA.

   In SAP Web IDE for SAP HANA, right-click the root folder of the XS advanced MTA project and choose *New* > *Java Module*. While creating the new Java module, make sure you enable the following options:

   - *Enable SAP HANA Database support*
     The corresponding SAP HANA database module must already exist for the Java application.
   - *Enable OData support*

   > **Note**
   >
   > This requires a Java-module template that supports OData V4, for example, *OData V4 Service* or *Web Application with OData V4 Support*.

6. Write extensions within the Java module by using the API provided (required only for modify operations).

   You can use the extensions framework to enhance functionality of your OData service to support data modification operations, for example, CREATE, UPDATE, and DELETE. For more information on the Java-based OData extension framework, see *Related Links* below.

   > **Note**
   >
   > At this point you have a working MTA that exposes an OData v4 service from the annotated CDS contexts defined in the database module. A productive application may require an application router, which routes requests to the Java module, and a security configuration, which performs authentication and authorization checks during execution of a query. For configuring security and application router, follow the below steps.
   >
   > The following steps show you how to create an application router in the HTML 5 module and configure the `xs-security.json` file. For more information about configuring application security, see *Related Links* below.

7. Add an HTML5 module to your MTA project.

   In SAP Web IDE for SAP HANA, right-click the root folder of the XS advanced MTA project and choose *New* > *Basic HTML5 Module*.

8. Configure the **mta.yaml** file to expose your Java module.

   To expose your OData service, perform the steps indicated in the following procedure:

   a. Configure the `mta.yaml` development descriptor file to expose the URL for the Java module.

      Initially, the relevant section of the `mta.yaml` file would look similar to the code sample below; you need to verify that the Java module **provides** its URL for consumption by other modules. With this change, the configuration would look like the following example:

      > **Sample Code**

      ```
      - name: MyJavaModule
        type: java
        path: MyJavaModule
        requires:
          - name: hdi-container
            properties:
      ```

```
        JBP_CONFIG_RESOURCE_CONFIGURATION: [tomcat/webapps/ROOT/
META-INF/context.xml:{service_name_for_defaultDB" : "~{hdi-container-
name}"}]'
  provides:
    - name: MyJavaModule_api
      properties:
        service_url: ${default-url}
```

b. Configure the development descriptor (`mta.yaml`) such that the HTML5 module **requires** the above URL exposed from the Java module. Change the configuration of HTML5 module to match the following example:

Sample Code

```
- name: MyHTML5Module
  type: html5
  path: MyHTML5Module
  requires:
   - name: MyJavaModule_api
     group: destinations
     properties:
      name: MyJavaModule_be
      url: ~{service_url}
```

c. Configure the application descriptor (`xs-app.json`) located in the `MyHTML5Module` directory to add a route to the Java module under the source "`/java/odata/v4`". You also need to specify the authentication type as **xsuaa**, as illustrated in the following example:

Sample Code

```
{
    "welcomeFile: "index.html",
    "authenticationMethod: "route",
    "routes": [{
        "source": "/java/odata/v4",
        "authenticationType": "xsuaa",
        "destination": "MyJavaModule_be"
            }]
}
```

Note

The `"destination":` value `"MyJavaModule_be"` must match the value specified in the application's development descriptor (`mta.yaml`)

9. Create a security descriptor (**xs-security.json**) file at the project level.

   For more information about creating the security descriptor (`xs-security.json`), see *Related Links* below. If you do not need to create any more scopes than those that are required by default (for example, due to security constraints in the `web.xml`), you can use the following example as a template.

Tip

The example below defines a single authorization scope (`ODATASERVICEUSER`) and a role template (`GenericODataAccessRole`) that has this scope.

Sample Code

```
{
"xsappname": "YourAppName ",
"scopes": [{
    "name": "$XSAPPNAME.ODATASERVICEUSER",
    "description": "Enter"
    }],
"role-templates": [{
    "name": "GenericODataAccessRole",
    "scope-references": [

"$XSAPPNAME.ODATASERVICEUSER"]
    }]
}
```

a. Add a dependency to the User Account and Authentication (UAA) service in the HTML5 (Web) module and the JAVA module of the development descriptor (`mta.yaml`), as illustrated in the following example:

Sample Code

```
- name: MyJavaModule
  type: java
  path: MyJavaModule
  requires:
    - name: MY_uaa
    - name: hdi-container
        properties:
          JBP_CONFIG_RESOURCE_CONFIGURATION:
          '[tomcat/webapps/ROOT/META-INF/context.xml:
          {"service_name_for_DefaultDB" : "~{hdi-container-name}"}]'
  provides:
    - name: MyJavaModule_api
      properties:
        service_url: ${default-url}
- name: MyHtml5Module
  type: html5
  path: MyHtml5Module
  requires:
    - name: MY_uaa
    - name: MyJavaModule_api
      group: destinations
      properties:
        name: MyJavaModule_be
        url:  ~{service_url}
       forwardAuthToken:true
```

Remember to add `MY_uaa` to the resource segment, too, as illustrated in the following example:

Sample Code

```
resources:
  - name: hdi-container
    properties:
      hdi-container-name: ${service-name}
      type: com.sap.xs.hdi-container
  - name: MY_uaa
    type: com.sap.xs.uaa
    parameters:
      path: ./xs-security.json
```

10. Create a new User Account and Authentication (UAA) service instance using the XS command-line interface.

    a. Log in to your XS advanced system from the command line interface using the `xs login` command.
    b. Download the security descriptor you created (`xs-security.json`) to your local file system.
    c. In the command shell, change directory to the directory where the `xs-security.json` file is located.
    d. Run the following command:

    ```
    xs create-service xsuaa devuser MY_uaa –c ./xs-security.json
    ```

11. Build the MTA Java application.

    You can now deploy your application and test the OData service. However, you need to ensure that users (for example, developers) have the authorization scopes required to access the application. To set up the required authorization, perform the following steps:

    a. Build the Multi-Target Application (MTA).
    b. Request the administrator to create a role for the `GenericODataAccessRole`. This is the role for which a role-template was defined in the `xs-security.json` security descriptor.

       If you have the required administration authorization, you can create a new role, which must then be added to the developer user's role collection. For more information about roles and role collections, see *Related Links*.

12. Run the Java module and the HTML5 module.

    This generates a `<Base URL>` for your application, which you can find the at the top right corner of the console in SAP Web IDE for SAP HANA.

    Check the availability of the OData service you created; you can access the OData service using the URLs listed in the following table:

    Sample OData service URLs

Scenario	Sample URL
Single context with namespace:	`<Base URL for the HTML5 module>`/java/odata/v4/`<namespace>`._.`<context>`
Single context without namespace:	`<Base URL for the HTML5 module>`/java/odata/v4/`<context>`
Child context with namespace:	`<Base URL for the HTML5 module>`/java/odata/v4/`<namespace>`._.`<OuterContext>`.`<InnerContext>`
Child context without namespace:	`<Base URL for the HTML5 module>`/java/odata/v4/`<OuterContext>`.`<InnerContext>`

    You can clean the cache of your application using the `clearCache` tool, for example, with either of the following URLs:

    - With namespace: `<Base URL for the HTML5 module>`/java/odata/clearCache/`<namespace>`._.`<context>`
    - Without namespace: `<Base URL for the HTML5 module>`/java/odata/clearCache/`<context>`

> **i Note**
>
> Cache cleaning is required only when the database module of an XS advanced application is updated and deployed individually; that is, without building and deploying the complete MTA to which the database module belongs. If you are deploying a complete, updated MTA, cache cleaning is not required. To perform a clean-cache operation, you need the authorization scope *ODATASERVICEADMIN*.

Related Information

SAP HANA Administration Guide (Building Roles for XS Advanced Applications)
Create the Security Descriptor for an XS Advanced Application [page 995]
Setting Up Application Projects [page 1087]
CDS Contexts in XS Advanced [page 339]
Configure the XS Advanced Application Router [page 913]
Using the OData Extension Framework [page 598]
Publishing CDS Contexts as OData Services [page 586]

6.1.2.1.1 Publishing CDS Contexts as OData Services

A CDS context defined in a CDS document can be published as an OData service.

It is possible to publish a CDS context as an OData V4 service by annotating the CDS context in the CDS document with the annotation `@OData.publish: true`.

> **! Restriction**
>
> The use of the `@OData.publish` annotation in a CDS document has the following restrictions.

- It is only possible to publish a CDS context as an OData v4 service using the `@OData.publish` annotation. If any other CDS artifact is extended with the `@OData.publish` annotation, for example, an entity or a view, the annotation is ignored and the CDS artifact is not considered for creating an OData service.
- An annotated context that includes a nested context cannot be published as an OData service.

  > **→ Tip**
  >
  > You can annotate a nested context for exposure as an OData service. However, the nested context must not, itself, contain any nested (child) contexts.

- If the annotated context includes a navigation to an external entity or view (outside the annotated context), no OData service can be created.
  To navigate from one context to another, you must create a view of the target entity and create a navigation to that view in the annotated context. However, it is allowed to include a reference to a complex type, inferred type, and scalar type from the annotated context to another context. In that scenario, every OData service creates the type inside its service and creates an internal reference.

- Direct access to the artifacts defined in an annotated nested context is possible, for example, using the following syntax in the query URL:
  - With namespace: `<Base URL>/java/odata/v4/<namespace>._.<OuterContext>.<InnerContext>`
  - Without namespace: `<Base URL>/java/odata/v4/<OuterContext>.<InnerContext>`

Contexts as OData v4 Services

All top-level contexts defined in a CDS document (or belonging to a specified name space) and appearing under the annotation `@OData.publish` are published as OData v4 services. There is no restriction on the number of CDS contexts that can be annotated as `OData.publish : true`.

Sample Code

Publish a CDS Context (and Contents) as an OData v4 Service

```
namespace acme.com;
@OData.publish : true
context ContextA {
  MyEntity1 {
    key ID : Integer;
  };
  MyEntity2 {
    key ID : Integer;
    type Address1
  };
};
```

Note

An annotated context that includes a nested (sub) context cannot be published as an OData service. However, you **can** annotate the nested context itself for exposure as an OData service. However, the nested context must not contain any nested contexts.

Subcontexts as OData v4 Services

In the following example, the CDS artifacts defined in the subcontexts `SubContextA1` and `SubContextA2` are exposed as OData v4 services.

Sample Code

Publish CDS Subcontexts an OData v4 Services

```
namespace acme.com;
context ContextA {
  @OData.publish : true
  context ContextA1 {
    MyEntity1 {
      key ID : Integer;
    };
  };
```

```
  @OData.publish : true
  context ContextA2 {
    MyEntity2 {
      key ID : Integer;
      type Address1
    };
  };
};
```

Related Information

CDS Contexts in XS Advanced [page 339]
Tutorial: Create OData v4 Services for XS Advanced Java Applications [page 581]

6.1.2.1.2 Supported CDS-OData v4 Features in the XS Advanced Java Stack

This document lists the OData v4 and CDS features supported by the XS advanced Java framework.

The information in this topic covers the following areas:

- Supported OData v4 Features [page 589]
- Supported CDS Features [page 592]

Code Syntax

Example CDS Document (`Sample.hdbcds`) with OData Service Enabled

```
@OData.publish : true
context Sample {
  type custid : Integer;
  entity Customer{
    key CustomerID : custid;
    key Type : String(20);
    CustomerName : String(100);
    CustAddress  : Address;
    SalesOrders : association[0..*] to SalesOrderHeader {CustomerID,Type};
  };

  type Address{
    Street : String(100);
    Area : String(100);
    City : String(30);
    State : String(30);
    Country : String(100);
  };

  entity SalesOrderHeader{
    key SalesOrderId : String(36);
    Note : hana.VARCHAR(300);
    CurrencyCode : String(5);
    NetAmount : Decimal(10,2);
    TaxAmount : Integer;
    OrderDate : LocalDate;
    CallTime : LocalTime;
```

```
    DeliveryETA : UTCDateTime;
    CashOnDelivery : Boolean;
    CustomerID : customerid;
    Type : CustomerType;
    ToCustomer : association[1] to Customer {CustomerID,Type};
    LineItems : association[0..*] to SalesOrderLineItem {SalesOrderID};
  };

  entity SalesOrderLineItem{
    key SalesOrderID : String(36);
    key Position : Integer;
    ProductId : Integer;
    Note : String(255);
    Quantity : Integer;
    SalesOrder : association to SalesOrderHeader;
  };
};
```

Supported OData Features

The following table lists the OData version 4.0 features supported by the XS advanced Java framework:

Supported OData Features

Feature	Supported Scenario	Sample Query
Read	Read an entity set	`/Customer`
	Read a particular entity	`/Customer(CustomerID=1,Type='Enterprise')`
	Filter operation on entity set	`/Customer?$filter=(Name eq 'Rowan Fairbrother')`
Navigation	One to many	`/Customer(CustomerID=1,Type='Enterprise')/SalesOrders`
	One to one	`/SalesOrderHeader(1)/ToCustomer`
	Navigation with filter	`/Customer(CustomerID=1,Type='Enterprise')/SalesOrders?$filter=NetAmount gt 5`
	Cross service referencing	
`$expand`	Expand a subset	`/Customer?$expand=SaleOrders`
	Multi-level expand	`/Customer?$expand=(SalesOrders?$expand=(LineItems))`

Feature	Supported Scenario	Sample Query
	Expand more than one subset	`/SalesOrderHeader?$expand=LineItems,ToCustomer`
	Multi-level expand with filter operations	`/Customer?$expand=(SalesOrders?$expand=(LineItems?$filter=Position eq 1))`
	Query operation with multi-level expand	`/SalesOrderHeader?$expand=LineItems($select=quantity)&$select=deliveryeta`
`$top & $skip`	Request a particular page of items (`$top`) by excluding (`$skip`) selected items. Number of results can be restricted under expanded entites.	`/Customer?$top=10&$skip=5` `/Customer?$expand=SalesOrders($top=2,$orderby=NetAmount desc)` Expands SalesOrders for a customer only with the top 2 highest netamount
Pagination	You must treat the URL of the *Next* link as "opaque". You cannot append system query options to the URL of a *Next* link. OData services may use the reserved system query option `$skiptoken` when building next links.	N/A
`$batch`	Read and Query options	
	Change sets	
`$apply`	`Aggregate`	`/SalesOrderHeader?$apply=aggregate(NetAmount with sum as total)`
	`GroupBy`	`/SalesOrderHeader?$apply=groupby(CustomerID)`
	`Filter`	`SalesOrderHeader?$apply=filter(NetAmount add TaxAmount lt 10)`
	`Compute`	`/SalesOrderHeader?$apply=compute(NetAmount add TaxAmount as TaxInclusive)`

Feature	Supported Scenario	Sample Query
$count	As a resource path segment **! Restriction** Not supported with navigation	`/SalesOrderHeaders/$count`
	As a system query option **! Restriction** Works only at the parent level	`/SalesOrderHeaders?$count=true`
$filter	Lambda operators: **! Restriction** • Works only at the parent level • Only comparison operators are supported Comparison operators: **! Restriction** The `'has not'` operator is not supported. Logical operators: **! Restriction** The `'not'` operator is not supported. Query functions: **! Restriction** Only string functions supported. In string functions, `'indexof'` and `'substring''` are not supported.	`/SalesOrderHeaders?$filter=LineItems/any(d:d/Quantity gt 200)`

OData v4 Analytics

In OData version 4, the `Aggregate` option enables data aggregation in OData without compromising general OData principles; it avoids the creation of any superfluous Edm data types that are required for showing aggregated data. Without aggregation, you must create Edm types, each of which has the structure of the desired view. This is not a development model that is scalable or maintainable, and it would require a lot of effort to set up and implement.

You can combine options with the `$apply` query, as illustrated in the following examples:

- `compute/aggregate`
  `SalesOrderHeader?$apply=compute(NetAmount add TaxAmount as TaxInclusive)/aggregate(TaxInclusive with sum as Totality)`
- `filter/aggregate`
  `SalesOrderHeader?$apply=filter(NetAmount add TaxAmount gt 10)/aggregate(TaxAmount with sum as Tottax)`
- `filter/aggregate/ compute`
  `SalesOrderHeader?$apply=filter(NetAmount add TaxAmount gt 10)/aggregate(TaxAmount with sum as Tottax)/compute(Tottax mul 3 as TriTottax)`
- `filter/groupby(aggregate)`
  `SalesOrderHeader?$apply=filter(NetAmount add TaxAmount gt 10)/groupby((CustomerID),aggregate(TaxAmount with average as AvgTx))`
- `groupby(aggregate)/filter`
  `SalesOrderHeader?$apply=groupby((CustomerID),aggregate(NetAmount with sum as total)/filter(total gt 20000))`
- `filter/compute`
  `SalesOrderHeader?$apply=filter(NetAmount add TaxAmount gt 10)/compute(TaxAmount mul 2 as DT)&$expand=LineItems`

Supported CDS Features

The following table lists the CDS features supported by the XS advanced Java framework in combination with OData v4 :

Supported CDS Features

Feature	Comments
Entities	Entities without a key are not supported
Views	Views without keys are not supported
Data types:	• Primitive • Scalar • Inferred • Complex • Complex types within complex types • Inferred types within complex types

Feature	Comments
UI annotations	• `Hidden` • `Identification` • `LineItem` • `Mandatory` • `Masked` • `MultiLineText` • `Optional` • `ReadOnly` • `StatusInfo` • `Chart` • `Contact` • `FieldGroup` • `DataPoint` • `SelectionFields` • `HeaderInfo`
Additional metadata annotations	• `Consumption.semanticObject` • `Consumption.filter` • `ObjectModel.semanticKey` • `ObjectModel.virtualElement` • `ObjectModel.createEnabled` • `ObjectModel.updateEnabled` • `ObjectModel.deleteEnabled` • `Semantics.quantity.unitOfMeasure` • `Semantics.amount.currencyCode` • `Semantics.address`

Note

To use UI annotations, you need to add `using sap.common::UI;` in your CDS files. In addition, it is necessary to add a `package.json` to the XS advanced application's HDB module or modify the existing one. The `package.json` file in the HDB module must specify the additional dependency to the "`@sap/ui-annotations`", as illustrated in the following example:

Sample Code

Using UI Annotations

```
{
 "name": "deploy",
 "dependencies":
  {
   "@sap/hdi-deploy": "3.1.2"
   "@sap/ui-annotations": "2.0.3"
```

```
  },
 "scripts":
   {
    "start": "node node_modules/@sap/hdi-deploy/deploy.js"
   }
}
```

Related Information

Defining OData v4 Services for XS Advanced Java Applications [page 580]
Tutorial: Create OData v4 Services for XS Advanced Java Applications [page 581]
CDS to OData v4 Datatype Mapping [page 594]
Streaming Data for OData Queries [page 605]
Enable Server Driven Pagination [page 606]

6.1.2.1.2.1 CDS to OData v4 Datatype Mapping

This document lists how the XS advanced Java framework interprets the CDS artifacts defined in CDS documents (`.hdbcds`) and converts them to OData metadata.

The information in this topic covers the following areas:

- CDS-OData v4 Datatype Mapping [page 594]
- CDS Artifact to OData v4 Datatype Mapping [page 595]

CDS Data Types as OData v4 Data Types

The following table illustrates how CDS data types are mapped to OData V 4.0 data types:

Datatypes Mapping

CDS Datatype	OData Edm Datatype
String (n)	Edm.String
LargeString	Edm.Stream
Binary (n)	Edm.Binary
LargeBinary	Edm.Stream
Boolean	Edm.Boolean
Integer	Edm.Int64

CDS Datatype	OData Edm Datatype
Integer64	Edm.Int64
Decimal(p,s)	Edm.Decimal
DecimalFloat	Edm.Decimal
BinaryFloat	Edm.Double
LocalDate	Edm.Date
LocalTime	Edm.TimeOfDay
UTCDateTime	Edm.DateTimeOffset
UTCTimeStamp	Edm.DateTimeOffset

CDS Artifact as OData v4 Datatypes

The following table illustrates how CDS artifacts, for example, entities or views, are mapped to OData V 4.0 data types:

Artifacts Mapping

CDS Artifacts	OData Type
Context	Schema
Entity (with keys)	Entity type
Entity (without keys)	**Not exposed**; can be exposed as functions
Views (with keys)	Entity type
Views (without keys)	**Not exposed**; can be exposed as functions
Column properties	Edm properties
Complex type	Complex type
Scalar type	Type definition
Association (0...1)	Navigation type=<entity type> Nullable="true"
Association (1...1)	Navigation type=<entity type> Nullable="false"
Association (m...*)	Navigation type=<collection>
Association to complex type	

CDS Artifacts	OData Type
Calculated fields	
Enumeration	
Constant	
Default values(Not supported by CDS)	

Related Information

Tutorial: Create OData v4 Services for XS Advanced Java Applications [page 581]
Defining OData v4 Services for XS Advanced Java Applications [page 580]

6.1.2.1.3 Using Cross Service Referencing

This document explains you how to use cross service navigation and referencing.

Cross service navigation enables inter communication of services. With this, navigation properties of entities of one service can reach entities of another service in a service group.

Sample Code

Context: product_sales:

```
using product_category;
@OData.publish : true
context product_sales
{
  entity products
    {
    key id : Integer;
    name : String(50);
    category : Association to product_category.category;
    };
};
```

Context: product_category

```
using product_details;
@OData.publish : true
context product_category
{
  entity category
    {
    key id : Integer;
    name : String(50);
    description : product_details.description;
    };
};
```

Context: product_details

```
@OData.publish : true
context product_details
{
type description
    {
    short_description : String(40);
    long_description : String(100);
    };
};
```

Navigation: product_sales → product_category

Referencing: product_category → product_details

Now, when each of the mentioned contexts is published as OData service, their metadata would look like:

Sample Code

Context: **product_sales**

```
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx"
Version="4.0">
 <edmx:Reference Uri="../../product_category/$metadata">
  <edmx:Include Namespace="product_category"/>
 </edmx:Reference>
 <edmx:DataServices>
  <Schema xmlns="http://docs.oasis-open.org/odata/ns/edm"
Namespace="product_sales">
   <EntityType Name="products">
    <Key>
     <PropertyRef Name="id"/>
    </Key>
    <Property Name="id" Type="Edm.Int32" Nullable="false"/>
    <Property Name="name" Type="Edm.String" MaxLength="50"/>
    <NavigationProperty Name="category" Type="product_category.category"/>
   </EntityType>
      <EntityContainer Name="B0235B494D214C34B40169193F1F00A5">
        <EntitySet Name="products" EntityType="product_sales.products">
         <NavigationPropertyBinding Path="category"
Target="product_category.B0235B494D214C34B40169193F1F00A5/category"/>
        </EntitySet>
      </EntityContainer>
 </Schema>
 </edmx:DataServices>
</edmx:Edmx>
```

Context: **product_category**

Sample Code

```
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx"
Version="4.0">
 <edmx:Reference Uri="../../product_details/$metadata">
  <edmx:Include Namespace="product_details"/>
 </edmx:Reference>
  <edmx:DataServices>
    <Schema xmlns="http://docs.oasis-open.org/odata/ns/edm"
Namespace="product_category">
     <EntityType Name="category">
      <Key>
       <PropertyRef Name="id"/>
      </Key>
```

```
      <Property Name="id" Type="Edm.Int32" Nullable="false"/>
      <Property Name="name" Type="Edm.String" MaxLength="50"/>
      <Property Name="description" Type="product_details.description"/>
     </EntityType>
      <EntityContainer Name="B0235B494D214C34B40169193F1F00A5">
       <EntitySet Name="category" EntityType="product_category.category"/>
      </EntityContainer>
    </Schema>
  </edmx:DataServices>
 </edmx:Edmx>
```

Context: **product_details**

```
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx"
Version="4.0">
 <edmx:DataServices>
  <Schema xmlns="http://docs.oasis-open.org/odata/ns/edm"
Namespace="product_details">
    <ComplexType Name="description">
    <Property Name="short_description" Type="Edm.String" MaxLength="40"/>
    <Property Name="long_description" Type="Edm.String" MaxLength="100"/>
    </ComplexType>
    <EntityContainer Name="B0235B494D214C34B40169193F1F00A5"/>
  </Schema>
 </edmx:DataServices>
</edmx:Edmx>
```

Operation:

- Here, product_sales has a navigation to an entity, "category" that is in another context, product_category. The category entity in product_category refers to a type description from another context product_sales.
- The Namespace of the reference is the namespace of the schema it is referring to. Suppose the schema at URL: `https://host:port/java/odata/v4/product_details/$metadata` refers to a namespace, product_category, then the URL of the referred service would be: `https://host:port/java/odata/v4/product_category/$metadata.`

> **i** Note
>
> - If a navigated context is not published as an OData service, then the navigation fails and throws an appropriate error.
> - If a service is referencing to a type in a context which is not published, then the type is copied to the metadata that uses the type.

6.1.2.1.4 Using the OData Extension Framework

Extend the OData v4 service to enable modification actions.

The OData services you create are by default Read-only services. To support write operations such as CREATE, UPDATE, DELETE, you have to write extensions, which are custom Java code containing annotated methods that perform the required operations. There is no restriction in terms of the package where the extension classes must be located. However, typically you create the class in the default package created within the XS advanced application's Java module. An `extensionTemplate` class is created by default when you create a Java application module with OData support enabled.

The OData v4 extension framework provides access to a set of objects which enables you to obtain the values required to implement the extension, as illustrated in the following table:

OData v4 Extension Objects and Access

Action	Request Payload	Request Header	Key Info
CREATE	□	□	-
UPDATE	□	□	□
DELETE	-	□	□

The object `DataSourceParams` provides any parameters required that are specific to the named data source. In the context of CDS, this object enables access to the connection object for the schema in which the current application is deployed.

> → Tip
>
> If you have to write extensions where multiple exposed OData services have an entity with the same name, then that extension is called for the corresponding entity in both OData services. To resolve this ambiguity and maintain individual extensions for each of these entities, use the parameter `serviceName` with a value that is the name of the OData service to which the entity belongs.

Using the OData Extension Framework

You can use the extension framework to write code for OData methods, which are currently not supported by the framework, for example: `CREATE`, `UPDATE`, and `DELETE`. You can also use the extension for `READ`, for example, if it is necessary to perform the following actions:

- Completely override the framework handling for `READ`
- Perform some pre- or post-processing in addition to the framework's default implementation.

> ℹ Note
>
> The extension framework is **not** supported for the `QUERY` method.

For each (Entity Set, OData Method) combination for which you want to extend the default functionality, it is necessary to write a method. For example, if you want to handle `CREATE` or `UPDATE` for the entity set "Customers" and `DELETE` functionality for the entity set "SalesOrders", you must create three additional methods. These methods must be annotated with the annotation `ExtendDataProvider`, for which the following parameters are available:

- `entitySet`
  The value for the entity set for which the extension is being written
- `requestTypes`
  Typically it has a single method as the value. However, if you want to handle multiple OData methods within a single extension method, you can provide additional values
- `serviceName` (optional)
  In case you have to write extensions when multiple exposed OData services have an entity with the same name, then that extension is called for the corresponding entity in both OData services. To resolve this

ambiguity and maintain individual extensions for each of these entities, you need to use the parameter **serviceName**. This should have value as the name of the OData service to which the entity belongs to.

The method signatures would look like the following example:

Sample Code

```
@ExtendDataProvider(entitySet = { "Customers" }, requestTypes =
RequestType.CREATE)
public void createCustomer(ExtensionContext ecx)
@ExtendDataProvider(entitySet = { "Customers" }, requestTypes =
RequestType.UPDATE)
public void updateCustomer(ExtensionContext ecx)
@ExtendDataProvider(entitySet = { "SalesOrders" }, requestTypes =
RequestType.DELETE)
public void createSalesOrders(ExtensionContext ecx)
```

- The method should be annotated properly with `ExtendDataProvider` annotation.
- The method should return void.
- The method should take a parameter of type `ExtensionContext`.

Note

Currently, update through patch is not supported.

Related Information

Tutorial: Create OData v4 Services for XS Advanced Java Applications [page 581]
The ExtensionContext APIs [page 600]
Methods Under DataProviderExtensionContext Class [page 602]

6.1.2.1.4.1 The ExtensionContext APIs

The `ExtensionContext` object is used to retrieve the information required to write extensions for a particular OData method.

With the `CREATE` method, you must gain access to the request payload. For the `READ` method, you need a way to invoke the default framework implementation. For all the OData methods, there is only a single `ExtensionContext` object. For this reason, all methods are made available in extensions written for any OData method.

Caution

It is essential to use the correct methods for the particular scenario for which the extension is being written.

The following table lists the methods available, and where possible, provides some code examples:

ExtensionContext API Methods

API Method	Description	Example
`getDSParams()`	Used to get the data-source-specific parameters which might be useful for writing the extensions. For CDS, there is only the connection object which is accessed through this object.	`Connection conn = ((CDSDSParams) ecx.getDSParams()).getConnection();`
`getODataRequest()`	Used to get the current OData request object. This object can be used for getting the request headers. The following example shows how to check the Prefer header for return preference:	`Preferences.Return returnPreference = OData.newInstance().createPreferences(ecx.getODataRequest().getHeaders(HttpHeader.PREFER)).getReturn();`
`performDefault()`	This method can be used to invoke default implementation of the OData method for which the current extension is being written. This only makes sense with the READ method as the framework does not provide default implementation for CREATE,UPDATE, and DELETE methods.	N/A

`getUriInfo()`	Returns the `UriInfo` object that contains information about the URI, Query Options, etc. This is mandatory with DELETE and UPDATE where the key predicate is required.	`UriInfo uriInfo = ecx.getUriInfo();`
		
		`UriResourceEntitySet entity = (UriResourceEntitySet)uriInfo.getUriResourceParts().get(0);`
		`List<UriParameter> kp = entity.getKeyPredicates();`

`asDataProviderContext()`	Called to "specialize" the `ExtensionContext` so that additional methods are available for other use-cases. When this method is called, the result is an object of `DataProviderExtensionContext` which is a subclass of `ExtensionContext`.	N/A

Related Information

Tutorial: Create OData v4 Services for XS Advanced Java Applications [page 581]

Using the OData Extension Framework [page 598]

6.1.2.1.4.2 Methods Under DataProviderExtensionContext Class

A list of the methods available for the DataProviderExtensionContext class.

All the methods under `ExtensionContext` are also available under `DataProviderExtensionContext`. The following table lists and describes the additional methods that are available under`DataProviderExtensionContext` :

> **i** Note
>
> HANA DB returns Boolean values as 1 or 0. These values should be transformed to TRUE or FALSE while creating the response entity

DataProviderExtensionContext Class Methods

Method	Description	Examples

`getDeserializerResult()`	Used to retrieve the request payload. It returns an object of `DeserializerResult` on which you have to call `getEntity` method which will give the entity object corresponding to the request payload.	`DataProviderExtensionContext dpCtx = ecx.asDataProviderContext();`
		`DeserializerResult payload = dpCtx.getDeserializerResult();`
		`Entity customerEntity = payload.getEntity();`

`setResultEntity()`	Used to set the response entity which can be the result entity for CREATE, UPDATE or READ. This method should **not** be called if `setEntityToBeRead()` method is used to perform a READ operation.	N/A
`setEntityToBeRead()`	Used for CREATE and UPDATE requests, if you want to extend the framework to handle the READ part of the operation.	For the OData CREATE (and UPDATE) methods, the entity has to be created in the back end and returned as the response. However, the response part can be handled by the framework since the READ functionality is already supported. As a result, after finishing the creation step, rather than reading and forming the entity object in the extension itself, you can call this API, to make the framework handle the rest.

Method	Description	Examples
`setEntityToBeRead(Map<String, String> keys)`	Similar to the `setEntityToBeRead()` except that you can specify the key values of the entity to be read. This is useful in a case of an auto-generated key from the database in cases where the framework has no way of knowing which entity to read from the back-end. **i Note** Works for a deep insert, too.	N/A

Related Information

Tutorial: Create OData v4 Services for XS Advanced Java Applications [page 581]
Using the OData Extension Framework [page 598]
The ExtensionContext APIs [page 600]

6.1.2.1.4.3 Using Extension Framework for Actions and Function Imports

This document details you about extending the metadata to add actions and functions to your OData v4 application.

Operations allow execution of custom logic on parts of a data model. Functions are operations that do not have side-effects and can support further composition e.g. additional filter operations, functions or an action.

Actions are operations that allow side effects, such as data modification, and cannot be further composed in order to avoid non-deterministic behavior. Actions and functions can be categorized as:

1. Bound (called as members of an instance of that type)
2. Unbound (called as static operations)

Action imports and function imports enable unbound actions and functions to be called from the service root. To develop the required extension, follow the below steps

1. **Creating the metadata**
   To add functions and actions in your application, you need to write a method that takes no parameters and returns List<CsdlOperation>. Then you have to annotate the method with the annotation @ExtendMetadata and set the service name.

→ Tip

- Create all your actions and functions, and return the list.
- The developer can annotate as many methods with the annotation.
- If service name is not mentioned, then it is added to all the published OData service in the application.

Sample Code

```
Sample Code:
@ExtendMetadata(serviceName="SalesOrder")
public List<CsdlOperation> myOperations()
{
}
@ExtendMetadata
public List<CsdlOperation> myOperations()
{
}
```

2. **Processing the data request for the operation**
Now, you need to write the code to be executed when the corresponding action or function import is called. You can reuse the already available DataProvider extension and add a parameter called operationName. The guidelines for writing the code are same as writing extension for CRUD operations. The method name can be anything, but the parameter should be ExtensionContext.

→ Tip

- If no serviceName is mentioned, it is applicable on all services in the application.
- The logic remains the same as usual Extension for CUD operations.
- You should take care to set the proper return type as mentioned in the Action or Function return type. To set the return object, you can use either of the following two:
  - DataProviderExtensionContext dpCtx = ectx.asDataProviderContext();
  - dpCtx.setResultEntityCollection(<Olingo EntityCollection type>); or dpCtx.setResultEntity(<Olingo Entity type>); or dpCtx.setResult(<Object>);

Sample Code

```
@ExtendDataProvider(serviceName = "SalesOrder", operationName =
"countcategories")
public void count_cat(ExtensionContext ectx)
{
}
@ExtendDataProvider(operationName = "countcategories")
public void count_cat(ExtensionContext ectx)
{
}
```

Related Information

Action Imports and Function Imports

6.1.2.1.4.4 Streaming Data for OData Queries

Streaming is an optional feature that you can use to receive data in chunks. Streaming is typlically used to improve memory and time performance on the server side. There is very low memory footprint on the server side if a huge collection of data is being fetched. Clients can process the incoming streaming data. If the client doesn't support streaming, the transport layer takes care of accumulating the data and passing it on to the client.

> **i** Note
>
> Streaming is enabled by default.

The following restrictions apply to streaming for OData queries:

- The method `isStreaming()` is mandatory for a `DataProvider` and by default is set to **false**. To enable streaming for the `DataProvider`, you must set `isStreaming()` to **true**.
- By default, `CDSDataProvider` always streams data. However, you can override the `DataProvider` and `isStreaming()` method.
  You can switch streaming on or off according to your requirement. To turn off streaming, you can extend `CDSDataProvider` and add the `isStreaming()` method as illustrated in the following example:

  > Sample Code
  >
  ```
  @Override
  public Boolean isStreaming()
  {
  return false;
  }
  ```

> **!** Restriction
>
> Streaming is not supported for $batch and $apply calls.

Related Information

Defining OData v4 Services for XS Advanced Java Applications [page 580]
Tutorial: Create OData v4 Services for XS Advanced Java Applications [page 581]

6.1.2.1.4.5 Use Batch Operations with Change Set

In XS advanced the Java framework for OData v4 provides support for processing change sets within a batch request. To process a change set, the framework reuses the functionality provided for CRUD operation through an extension mechanism. To control the transaction, the framework monitors operations on the connection object provided to the extension developer. This ensures a seamless integration of existing extension functionalities with the new changes in the framework.

Context

The extension developer is expected to use traditional transaction logic while writing the extension, as follows:

Procedure

1. Get the connection object from the extension context, as illustrated in the following example:

   Sample Code

   ```
   Connection conn = ((CDSParams) ecx.getDSParams()).getConnection()
   ```

2. Write the required transaction logic.
3. Close the connection, for example, using the close-connection command: `Conn.close()`

   When a change set is requested, the framework internally ensures that the following calls on the connection object are ignored:

   - `connection.setAutoCommit (Boolean)`
   - `connection.rollback()`
   - `connection.rollback(Savepoint)`
   - `connection.commit()`
   - `connection.close()`

Related Information

Batch Requests

6.1.2.1.5 Enable Server Driven Pagination

Set limits for the amount of data to be fetched and sent over the network by OData queries.

Context

Server side pagination is used to limit the amount of data to be fetched and sent over the network to the client, in case generic query options yield huge amounts of data. You can request for responses that include only a partial set of the items identified by the request. For this, on the server side, you have to specify the number of entities to be returned in the response. The response contains a link, next link that allows you to retrieve the next partial set of items. You must treat the URL of the next link as opaque and must not append system query options to the URL of a next link.

To enable server-side pagination for your XS advanced Java application, perform the following steps:

Procedure

1. Create a new DataProvider class which extends CDS DataProvider.

   Once the project is created via of the archetype, add the Java class `MyDataProvider.java` to the Java module of your project.

2. Add your required page size in the `getPageSize()` method. As you are provided with UriInfo object, you can return different page size for different entity sets.

   The following code snippet shows how to turn off streaming and set the server-side page size:

   Sample Code

   ```
   package groupID.MySalesOrder;
   import org.apache.olingo.commons.api.edm.provider.CsdlEdmProvider;
   import org.apache.olingo.server.api.uri.UriInfo;
   import com.sap.gateway.v4.rt.cds.CDSDataProvider;
   import com.sap.gateway.v4.rt.cds.ODataToCDSProcessor;
   public class MyDataProvider extends CDSDataProvider {
   public MyDataProvider(CsdlEdmProvider edmProvider) {
   super(edmProvider);
   }
   public MyDataProvider(ODataToCDSProcessor jdbcProcessor, CsdlEdmProvider
   edmProvider) {
   super(jdbcProcessor, edmProvider);
   }
   /*To disable streaming, return false*/
   @Override
   public Boolean isStreaming()
   {
   return false;
   }
   /*To set the server side page size, return a long value. To disable server
   side paging,
   * return getPageSize() as null*/
   @Override
   public Long getPageSize(UriInfo uriInfo) {
   return (long)1000;
   }
   }
   ```

3. Change your web.xml file to update the value of the context parameter DPC to your new class.

   Sample Code

   ```
   <context-param>
   <param-name>DPC</param-name>
   <param-value>package.MyDataProvider</param-value>
   </context-param>
   ```

   The framework takes care to append the NextLink in case it is required.

Next Steps

Regulations imposed when you request a particular page size through header `odata.maxpagesize()`.

- When page size is null and server side pagination is switched off, this request is ignored.
- When requested page size is lesser than or equal to that set by developer, then framework services the new page size requested by you.
- When requested page size is greater than that set by developer, then framework ignore the request and serves the one set by the developer.
- In all cases when server driven pagination is active, the page size requested used in the collection query is present in the response header Preference-Applied : odata.maxpagesize.

Related Information

Defining OData v4 Services for XS Advanced Java Applications [page 580]
Tutorial: Create OData v4 Services for XS Advanced Java Applications [page 581]
Server-Driven Paging

6.1.2.1.6 Customize Applications with the User's Locale

In XS advanced the Java framework enables you to personalize your application with the user's locale.

Context

The locale can be retrieved from two headers in the following order:

1. Custom `x-sap-request-language` HTTP header
2. `Accept-Language` HTTP header
   Check if the locale is set in the request headers. If it is, set it for the connection so that the database can perform locale specific operations. If there is no locale, do not set anything in the connection.

   > **i Note**
   >
   > This functionality is always on.

   To use the locale personalization, follow the below procedure:

Procedure

1. Create a new `MyDataProvider` class that extends `CDSDataProvider`.

The implementation for setting the locale is already included in `CDSDataProvider`. The connection object comes from the `CDSDataprovider` class, you need not do anything else.

2. Change the `web.xml` file to update the value of the context parameter (`context-param`) to `DPC` in your new class (new DataProvider).

Sample Code

```
<context-param>
   <param-name>DPC</param-name>
  <param-value>package.MyDataProvider</param-value>
</context-param>
```

6.1.2.1.7 Performance Statistics for OData Requests

Enable access to performance statistics for OData requests.

In order to facilitate the measurement of performance metrics for OData end points provisioned in XS advanced, the application provides support for getting the same via query parameters and headers without the need to set up performance tools explicitly. This feature also helps in identifying the bottleneck area during network communication, which you can later troubleshoot.

You can access SAP performance statistics from the framework for each OData request. To obtain the SAP performance statistics, you can add `?sap-statistics-xsa-java-odatav4=true` at the end of the request URL, regardless of whether it is an HTTP method of type GET, POST, PUT, MERGE, PATCH or DELETE. Alternatively, you can enter `name=sap-statistics-xsa-java-odatav4, value=true` in the HTTP request header. The setting in the URL is treated with higher priority than setting it in the HTTP header.

The XS Advanced OData Provisioning framework (XS advanced framework for provisioning of OData) provides performance statistics to consumers in the HTTP response header in the following format:

- HTTP Header Name: `sap-statistics-xsa-java-odatav4`
- HTTP Header Value: `true`

The following table lists and describes the HTTP header values:

HTTP Header Values

HTTP Header Value	Description
`total`	The total processing time for executing an OData request starting from the time the JAVA application received the call to the time it sent back the first response
`fw`	The time taken by OData framework excluding the time taken to retrieve data from back-end; this includes time taken before sending request to back-end and after receiving the data from back-end. It also includes time taken for extension code to complete.
`ext`	The time taken by the custom code written for extensions to execute
`db`	The time taken to retrieve data from the back-end including the network round trip to the back-end.

> **i Note**
>
> - Due to SAP security policies, OData provisioning provides performance statistics only if the OData request is processed successfully.
> - All times are calculated in milliseconds.

Related Information

Defining Web-based Data Access in XS Classic
Defining OData v4 Services for XS Advanced Java Applications [page 580]

6.2 Data Access with XMLA in SAP HANA XS

In SAP HANA Extended Application Services (SAP HANA XS) , the persistence model (for example, tables, views and stored procedures) is mapped to the consumption model that is exposed to clients - the applications you write to extract data from the SAP HANA database.

You can map the persistence and consumption models with XML for Analysis (XMLA). With XMLA, you write multi-dimensional-expressions (MDX) queries wrapped in an XMLA document. An XML for Analysis (XMLA) application running in SAP HANA application services (SAP HANA XS) is used to provide the consumption model for client applications exchanging MDX queries (wrapped in XMLA documents) with the SAP HANA database.

> **→ Tip**
>
> For more information about using MDX queries to access SAP HANA cube models, see *Multidimensional Expressions (Client Interfaces Programming Reference)* in *Related Information* below.

XMLA uses Web-based services to enable platform-independent access to XMLA-compliant data sources for Online Analytical Processing (OLAP). XMLA enables the exchange of analytical data between a client application and a multi-dimensional data provider working over the Web, using a Simple Object Access Protocol (SOAP)-based XML communication application-programming interface (API).

Applications running in SAP HANA XS enable you to control the flow of data between the presentational layer, for example, in the Web browser, and the data-processing layer in SAP HANA itself, where the calculations are performed, for example in SQL or SqlScript. If you develop and deploy an XMLA service running in SAP HANA XS, you can take advantage of the access to SAP HANA that SAP HANA XS provides to improve end-to-end performance.

> **i Note**
>
> The XS advanced application `xmla` must not be installed in the `SAP` space.

If you are using multiple tenant databases, you must install the `xmla` application in a space (or spaces) other than the default space `SAP`, for example, `DEV` or `PROD`. In addition, the target space must already be mapped to a tenant database **before** you deploy the `xmla` application. You can map an XS advanced space to

a tenant database using the *SAP HANA Service Broker Configuration* tool that is included in the *XS Advanced Administration* tools.

> → Tip
>
> The XMLA interface for XS advanced is available either on the SAP HANA media or for download from SAP Service Marketplace for those people with the required S-User ID:
>
> *Service Marketplace* > *Products* > *Software download* > *SUPPORT PACKAGES & PATCHES* > *By Alphabetical Index (A-Z)* > *X* > *XSAC XMLA INTERFACE FOR HANA 1*

Related Information

Set up and Use the XMLA Interface in XS Advanced [page 612]
SAP Support Portal: Software Downloads
Multidimensional Expressions (Client Interfaces Programming Reference)

6.2.1 Define the Data to Expose with an XMLA Service in XS Advanced

Define the tables and views to expose with the XMLA service in XS advanced.

Prerequisites

- Data to expose
  If you already have a data model containing tables or views that can be exposed, you do not need to create additional elements. You can use the tables and views that are already available.
- Access to data
  Ensure that the suitable permissions are granted to the users and applications requiring access
- Access to the XMLA service in XS advanced
  Only XS advanced users have access to the XMLA service; you can clone existing SAP HANA users to XS advanced, for example, using the XS Advanced Administration Tools

Context

An XMLA service exposes data stored in database tables for analysis and display by client applications. However, first of all, you need to ensure that the tables and views to expose as an XMLA service actually exist and are accessible.

To define the data to expose using an XMLA service, you must perform at least the following tasks:

Procedure

1. Create a simple database schema.
2. Create a simple database table to expose with an XMLA service.
3. If required, create a simple database view to expose with an XMLA service.
4. Grant select privileges to the tables and views to be exposed with the XMLA service.

Related Information

Data Access with XMLA in SAP HANA XS [page 610]
Set up and Use the XMLA Interface in XS Advanced [page 612]

6.2.2 Set up and Use the XMLA Interface in XS Advanced

Enable and test the XMLA interface in XS advanced.

Prerequisites

Before you set up and use the XMLA interface for XS advanced, note the following prerequisites:

- The XMLA interface in XS advanced is available either on the SAP HANA media or for download from SAP Service Marketplace at the following location for those people with the required S-User ID:
  Service Marketplace > *Products* > *Software download* > *SUPPORT PACKAGES & PATCHES* > *By Alphabetical Index (A-Z)* > *X* > *XSAC XMLA INTERFACE FOR HANA 1*
- If you are using multiple tenant databases, you must deploy the `xmla` application in a space (or spaces) other than `SAP`, for example, `DEV` or `PROD`.
- The target space must already be mapped to a tenant database **before** you deploy the `xmla` application.

> **!** Restriction
>
> Administrator permissions are required to access the *SAP HANA Service Broker Configuration* tool you need to use to complete this mapping task.

Context

An XMLA service for XS advanced is configured by deploying the multi-target application (MTA) `xmla` to the XS advanced run-time environment. After successful deployment of `xmla` to the XS advanced run-time environment, the XMLA interface is available at the following URL:

`https://<SERVERNAME>:<PORT>/sap/hana/xmla/requestbasic`

Where:

- `<SERVERNAME>`
  Is the name of the host where your XS advanced instance is running
- `<PORT>`
  Is the number of the port the XMLA service is listening on, for example:
  `https://host1.acme.com:51028/sap/hana/xmla/requestbasic`

To set up and use the XMLA interface in XS advanced, perform the following steps:

Procedure

1. Check if the XMLA application is already running in the XS advanced run-time environment.

   To display a list of running applications in XS advanced, use the `xs apps` command, as illustrated in the following example:

   > **i** Note
   >
   > Check the state of the application is *STARTED* as displayed in the *requested state* column.

   ```
   xs apps
   Getting apps in org "acmeorg" / space "DEV" as dev1...
   Found apps:
   name                 requested state   instances   memory    disk
   urls
   ------------------------------------------------------------------------------
   -------------------------
   auditlog-db          STOPPED           0/1         1.00 GB   <unlimited>
   <none>
   auditlog-server      STARTED           1/1         512 MB    <unlimited>
   https://host1.acme.com:51002
   auditlog-broker      STARTED           1/1         64.0 MB   <unlimited>
   https://host1.acme.com:51003
   [...]
   xsa-admin-backend    STARTED           1/1         1.00 GB   <unlimited>
   https://host1.acme.com:51010
   xsa-admin            STARTED           1/1         1.00 GB   <unlimited>
   https://host1.acme.com:51009
   xmla                 STARTED           1/1         1.00 GB   <unlimited>
   https://host1.acme.com:51028
   [...]
   ```

   > **→** Tip
   >
   > To change the target space, use the command `xs target -s <SpaceName>`, for example, `xs target -s PROD`.

2. If you are using multiple tenant databases, map the XS advanced run-time space where you want to deploy `xmla` to the tenant database to which you want the XMLA interface to connect.

   You can deploy `xmla` to more than one space, and map each space to a different tenant database.

   > → Tip
   >
   > You can map an XS advanced space to a tenant database using the *SAP HANA Service Broker Configuration* tool that is included in the *XS Advanced Administration* tools. For more information about the *XS Advanced Administration* tools, refer to the *Application Run-time Services* section (for XS advanced) in the *SAP HANA Administration Guide*.

3. Deploy the XMLA application.

   You must install the `xmla` application in a space (or spaces) other than the default `SAP`, for example, `DEV` or `PROD`.

   > → Tip
   >
   > If you are using multiple tenant databases, the target space must already be mapped to a tenant database **before** you deploy the `xmla` application.

   To deploy an XS advanced application, use the `xs deploy` command, as illustrated in the following example:

   ```
   xs deploy <XMLA_MTA_ARCHIVE> --no-namespaces-for-services
   ```

4. Display the URL where the XMLA application is running.

   In the following example, the XMLA interface is available at the following URL: `https://host1.acme.com:51028`

   ```
   xs app xmla
   Showing status and information about "xmla"
     name:              xmla
     requested state:   STARTED
     instances:         1
     memory:            1.00 GB
     disk:              <unlimited>
     buildpack:         <default>
     urls:              https://host1.acme.com:51028
   Instances of droplet 1 created at Sep 21, 2016 5:17:09 PM
   index   created                  state     host                  internal
   port   os user
   ------------------------------------------------------------------------------
   ------------------
   1       Sep 30, 2016 2:43:10 AM   RUNNING   host1.acme.com:51028
   50032           acmexsa
   ```

5. Send an MDX query to the XMLA interface.

   The MDX query should be sent in the form of a POST-request using the following URL:

   ```
   https://<servername>:<PORT>/sap/hana/xmla/requestbasic
   ```

   The POST-request you send should include the following HEADER elements:

   - SOAPAction = "urn:schemas-microsoft-com:xml-analysis:Execute"
   - Content-Length = <length>

- Content-Type = text/xml
- Authorization = Basic <base64-encoded String "<Username>:<Password>">

The request BODY must be a valid XMLA request, as illustrated in the following example:

Sample Code

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
<Header>
<Version Sequence="100" xmlns="http://schemas.microsoft.com/
analysisservices/2003/engine/2" />
</Header>
<Body>
<Execute xmlns="urn:schemas-microsoft-com:xml-analysis">
<Command>
    <Statement>
        SELECT {[K3].[K3].[All].[(all)]} on 0, {Measures.M1} on 1 FROM
MDX_TEST_03_CV3J
    </Statement>
</Command>
<Properties>
    <PropertyList />
</Properties>
</Execute>
</Body>
</Envelope>
```

Caution

The request you send **must** have a body. If you send an empty request (without a body), the XMLA service displays an error message, for example, "`TypeError: Cannot read property 'substring' of undefined`".

Related Information

SAP Support Portal: Software Downloads
Define the Data to Expose with an XMLA Service in XS Advanced [page 611]
The XMLA Service in XS Advanced [page 615]
MDX (Client Interfaces Programming Reference)
SAP HANA Administration Guide

6.2.2.1 The XMLA Service in XS Advanced

The XMLA service is defined by deploying the Node.js application `xmla`.

The XMLA service is defined by deployment of the Node.js application `xmla`. Exposed data is available for analysis and display by client applications, for example, a Web browser that uses functions provided either by the XMLA service running in SAP HANA XS or by an XMLA client library running on the client system.

To expose data to XS advanced applications using XMLA, you first need to define database views that provide the data with the required granularity. The XMLA service is used to control access to the exposed data.

> **! Restriction**
>
> XS advanced supports XMLA version 1.1, which you can use to send MDX queries to the database.

If the `xmla` is deployed to (and running in) the XS advanced run-time environment, the XMLA interface is available at the following URL:

`https://<SERVERNAME>:<PORT>/sap/hana/xmla/requestbasic`

Where `<SERVERNAME>` is the name of the host where your XS advanced instance is running, and `<PORT>` is the number of the port the XMLA service is listening on, for example:

`https://host1.acme.com:51028/sap/hana/xmla/requestbasic`

In XS advanced, you can use the `xs apps` command to list details of all XS advanced applications running in the current space; the information displayed includes the URL and port number to use for access to the application. For more information about a specific application running in XS advanced, use the `xs app <appName>`, as illustrated in the following example:

```
xs app xmla
Showing status and information about "xmla"
  name:              xmla
  requested state:   STARTED
  instances:         1
  memory:            1.00 GB
  disk:              <unlimited>
  buildpack:         <default>
  urls:              https://host1.acme.com:51028
Instances of droplet 1 created at Sep 21, 2016 5:17:09 PM
index   created                  state     host                  internal
port   os user
------------------------------------------------------------------------------
--------------
1       Sep 30, 2016 2:43:10 AM   RUNNING   host1.acme.com:51028
50032           acmexsa
```

> **→ Tip**
>
> If the `xmla` is not running, you need to deploy it, for example, with the `xs deploy` command.

Currently, access to the XMLA service in XS advanced is only granted to users registered with the XS advanced run time. You can clone to XS advanced all SAP HANA users that need to be able to use the XS advanced XMLA service, for example, with the *User Management* tool included in the XS Advanced Administration Tools. The basic authentication request is performed with this name and password of the XS advanced user.

Related Information

Define the Data to Expose with an XMLA Service in XS Advanced [page 611]
Set up and Use the XMLA Interface in XS Advanced [page 612]
The XS Command-Line Interface Reference

6.2.2.2 XML for Analysis (XMLA)

XML for Analysis (XMLA) uses Web-based services to enable platform-independent access to XMLA-compliant data sources for Online Analytical Processing (OLAP).

XMLA enables the exchange of analytical data between a client application and a multi-dimensional data provider working over the Web, using a Simple Object Access Protocol (SOAP)-based XML communication application-programming interface (API).

Implementing XMLA in SAP HANA enables third-party reporting tools that are connected to the SAP HANA database to communicate directly with the MDX interface. The XMLA API provides universal data access to a particular source over the Internet, without the client having to set up a special component. XML for Analysis is optimized for the Internet in the following ways:

- **Query performance**
  Time spent on queries to the server is kept to a minimum
- **Query type**
  Client queries are stateless by default; after the client has received the requested data, the client is disconnected from the Web server.

In this way, tolerance to errors and the scalability of a source (the maximum permitted number of users) is maximized.

XMLA Methods

The specification defined in XML for Analysis Version 1.1 from Microsoft forms the basis for the implementation of XML for Analysis in SAP HANA.

The following list describes the methods that determine the specification for a stateless data request and provides a brief explanation of the method's scope:

- **Discover**
  Use this method to query metadata and master data; the result of the `discover` method is a rowset. You can specify options, for example, to define the query type, any data-filtering restrictions, and any required XMLA properties for data formatting.
- **Execute**
  Use this method to execute MDX commands and receive the corresponding result set; the result of the `Execute` command could be a multi-dimensional dataset or a tabular rowset. You can set options to specify any required XMLA properties, for example, to define the format of the returned result set or any local properties to use to determine how to format the returned data.

Related Information

Data Access with XMLA in SAP HANA XS [page 610]
MDX (Client Interfaces Programming Reference)

6.3 Data Access with InA in XS Advanced

Enable Web-based access to data for the purposes of analysis using the Information Access (InA) protocol.

The analytics adapter for SAP HANA extended application services, advanced model (analytics adapter) enables you to provide XS advanced applications and any data-analytics clients with remote access to data stored in SAP HANA as well as the data-analysis components provided by SAP HANA. The analytics adapter uses the InA protocol for access to SAP HANA 2.0 systems to provide an http(s) end point that analytics clients can connect to.

Analytics clients (for example, SAP Analytics Cloud or SAP Analysis for Office) use InA to connect to SAP HANA to execute analytic queries. Developers of XS advanced applications often need to add analytical capabilities to their application by providing connectivity to such analytics clients. These applications require an InA handler to enable access via the InA protocol to the data stored by the application in SAP HANA.

The information in this section not only explains how to set up and use the analytics adapter but also provides some important details concerning the underlying context and background.

The analytics adapter is intended for use in the following typical deployment scenarios:

- In the Cloud
  The analytics adapter is bound to the HDI container of the XS advanced application that wants to allow analytics clients access its data.
- On-premise
  Deployed XS applications bind their HDI container to the analytics adapter in order to enable analytics clients to access the application's data.
  Use InA in XS advanced to access the objects in the customer SAP HANA database - using different SAP HANA database users that can only see user-specific artifacts.

Related Information

Maintain the Analytics Adapter in XS Advanced [page 618]

6.3.1 Maintain the Analytics Adapter in XS Advanced

Set up the SAP HANA analytics adapter (`xsahaa`) for use with XS advanced applications.

Prerequisites

Before you set up and use the analytics adapter for SAP HANA extended application services, advanced model, note the following prerequisites:

- If you are using multiple tenant databases, you must deploy the analytics adapter in a space (or spaces) other than the default space (SAP), for example, in the space DEV.

- The target run-time space must already be mapped to a tenant database **before** you deploy the analytics adapter.
- Your SAP HANA version must be at least SAP HANA 2.0 SPS 01 (Revision 12) or a later version.
- You must have the SAP HANA EPM-MDS component installed in your SAP HANA database.
- Your application needs to provide a logged in user to the analytics adapter. This means that you need to have an App Router and a UAA service deployed together with the analytics adapter.

Context

The analytics adapter for SAP HANA extended application services, advanced model is an application that you need to deploy to XS advanced in combination with your application, either as a multi-target application (MTA) or using a manifest file. After successful deployment of your application and the analytics adapter, an additional service is started. The following URLs are available for use by analytics clients:

- `https://<SERVERNAME>:<PORT>/sap/bc/ina/service/v2/GetServerInfo`
- `https://<SERVERNAME>:<PORT>/sap/bc/ina/service/v2/GetResponse`

Procedure

1. Map the tenant database to an XS advanced run-time space.

   If you are using multiple tenant databases, map the XS advanced run-time space where you want to deploy the analytics adapter to the tenant database to which you want the analytics adapter XS advanced to connect. You can deploy the analytics adapter to more than one space, and map each space to a different tenant database.

   > → Tip
   >
   > You can map an XS advanced space to a tenant database using the *SAP HANA Service Broker Configuration* tool included in the *XS Advanced Administration* tools. For more information about the *XS Advanced Administration* tools, see the Application Run-time Services section (for XS advanced) in the SAP HANA Administration Guide.

2. Download the analytics adapter model for XS advanced.

   > → Tip
   >
   > You can download the analytics adapter model for XS advanced from the SAP Development Tools page at https://tools.hana.ondemand.com/#hanatools.

   a. Unpack the archive you downloaded.
   b. Create a folder called `xsahaa` inside your application project.

      > i Note
      >
      > Sub-steps b, c, and d are only required if you are planning to use the MTAR builder tool to build an MTA archive (MTAR) to deploy your application. If your application uses a `manifest.yml` file to

manage application deployment, you can skip the following sub-steps and proceed to step 3 (*Add the analytics adapter to your XS advanced application*).

c. Extract the file `java-xsahaa.war` from the analytics adapter model for XS advanced archive you downloaded and copy it the `xsahaa` folder.

d. Create a `pom.xml` file inside the `xsahaa` folder.

> **Note**
>
> This `pom.xml` file is required by the MTAR builder.

Add the following content to the new `pom.xml` file:

Sample Code

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.sap.xsahaa</groupId>
  <artifactId>java-xsahaa</artifactId>
  <version>1</version>
</project>
```

3. Add the analytics adapter model to the configuration of your XS advanced application.

   This can be a `manifest.yml` file or an MTA description file (`mta[d].yaml`).

   The following example shows the definition of the analytics adapter model module in an XS advanced application's MTA description file:

   Sample Code

   MTA Development Descriptor File (`mta.yaml`)

```
...
  - name: java-xsahaa
    type: java.tomee
    path: java-xsahaa.war
    provides:
      - name: java-xsahaa
        properties:
          url: "${default-url}"
          properties:
            JBP_CONFIG_RESOURCE_CONFIGURATION: "[
               'tomee/webapps/ROOT/WEB-INF/resources.xml': {'xsahaa-hdi-
container':'my-hdi-container'},
               'tomee/webapps/ROOT/WEB-INF/web.xml': {'cors-
origins':'https://my-analytics-client.acme.com'}]"
    requires:
      - name: my-hdi-container
      - name: uaa
...
```

   In this example, the data you want to access using the analytics adapter is stored in the HDI container `my-hdi-container`. For this reason, you must specify a binding to this container in the `JSP_CONFIG_RESOURCE_CONFIGURATION`. Note the binding to an instance of the User Account and Authentication (UAA) service, which in this example is called `uaa`.

4. Configure the application-router module.

The analytics adapter must be bound to your XS advanced application's `approuter`. You also need to add a route to the application router configuration, as described in the following steps:

a. Bind the analytics adapter to your XS advanced application's `approuter` module.

   The binding can be configured in the appropriate section of either the application's MTA descriptor (`mta[d].yaml`) or the application's deployment manifest (`manifest.yml`). The following example shows how to provide this binding configuration in an MTA descriptor.

   Sample Code

   Binding Configuration in the `mta.yaml` Development Descriptor

```
...
 - name: approuter
   type: javascript.nodejs
   path: approuter/
   requires:
     - name: java-xsahaa
       group: destinations
       properties:
         name: java-xsahaa
         url: ~{url}
         forwardAuthToken: true
     - name: uaa
...
```

   Note

   The binding to an instance of the UAA service, called `uaa` in this example, is mandatory.

b. Add a new route to the XS advanced application's `approuter` configuration.

   The XS advanced `approuter` is configured in the application descriptor (`xs-app.json`). The new route forwards the URLs, used by the analytics adapter to the appropriate destination, as illustrated in the following example:

```
...
  "routes": [
   ...,
    {
      "source": "/sap/bc/ina",
      "destination": "java-xsahaa",
      "httpMethods": ["POST", "GET"]
    },
    {
      "source": "/sap/bc/ina",
      "destination": "java-xsahaa",
      "httpMethods": ["OPTIONS"],
      "authenticationType": "none"
    }
  ]
...
```

   Tip

   To enable a direct connection to the analytics adapter for an SAP Analytics Cloud tenant You can enable support for cross-origin resource sharing (CORS), for example, with the `<CORS>` application-router environment variable. The `CORS` environment variable can also be used in the `manifest.yml` file to specify permitted connection origins.

5. Build and deploy the application you want to connect to analytics adapter.

   When you deploy your application, the analytics adapter should start along with any specified application services. You can verify that everything is running correctly by opening a command shell, logging into XS advanced run time, and running the following command:

```
xs app java-xsahaa
...
Showing status and information about "java-xsahaa"
  name:              java-xsahaa
  requested state:   STARTED
  instances:         1
  memory:            512 MB
  disk:              <unlimited>
  buildpack:         <default>
  urls:              https://host1.acme.com:51028
...
Instances of droplet 1 created at Sep 13, 2017 10:13:44 AM
index created                   state   host                  int. port  os
user
------------------------------------------------------------------------------
--
0     Sep 13, 2017 10:13:52 AM  RUNNING host1.acme.com:51028  50032
acmexsa
```

   The host name and port information from this output is necessary not only for the analytics clients that you want to connect to this service but also to verify that the service can connect to the SAP HANA database, which is described in the next step.

6. Verify the analytics adapter is running and can connect to the SAP HANA database.

   Open a Web browser and run the following URL:

```
https://<SERVERNAME>:<PORT>/sap/bc/ina/service/v2/GetServerInfo
```

   Where:

   - `<SERVERNAME>`
     The name of the host where the XS advanced instance is running
   - `<PORT>`
     The number of the port that the analytics adapter is listening on

   This URL should return JSON content with information about your SAP HANA system. If the `GetServerInfo` command returns an error, check that all prerequisites are met for this task and that you followed all the instructions in the previous steps.

Related Information

Data Access with InA in XS Advanced [page 618]

6.3.1.1 Information Access (InA) Queries

Use the InA query model for analytic requests.

The Information Access (InA) infrastructure enables applications to perform analytics, planning, and search on data stored in SAP HANA. The InA model allows you to define queries using rich or complex semantics. The

query can read data from all kinds of SAP HANA views and EPM plan data containers. The InA model includes spatial and search features, too.

Generally speaking, Information Access (InA) is a definition for a JSON object model that reflects the analytic or planning request of the user. The query model for analytic requests consists of a description of the designated result set. It describes the data selection and calculations in a multidimensional way, which is usually in the form of a two-dimensional grid (i.e. a pivot table). The query model is not, however, restricted to the two-dimensional form; it can also have fewer or more than two axes.

> → Tip
>
> A two-dimensional grid is defined by two axes, which are referred to as Rows and Columns in InA, and each axis can contain one or more dimensions. This makes it possible to analyze complex relationships between different business objects, for example, revenue by product and country for every year. The InA query can contain more complex calculations, too.

Sorting

It is often necessary to be able to sort the result set of analytic requests in different ways, sometimes even with combinations of different criteria. For example, the user can sort the result set by a given attribute value of a dimension, but in addition sort by a specific measure value (that may or may not be contained in the query).

Measures

An analytic query can contain several measures to be selected from the data source (Selection Members). InA also supports additional formula measures based on those measures (Calculated Members). This can be used, for example, to compute the difference between two Selection Members. It is also possible to specify an additional aggregation step after the execution of the basis query or to calculate a rank for the resulting measure values.

Dimensions

The dimensions of a multidimensional grid can contain hierarchies and hierarchy features, for example, current member navigation. For each dimension, the client application can specify which of the attributes defined in the model shall be included in the result set. For example, in a **product** dimension, you might want to see the color of each product in addition to the product identifier.

Batch Requests with Multiple Queries

For performance reasons, it is often not advisable to send a separate query for each analytic payload. InA enables you to collect queries in a "batch" which can be sent in a single network call. As long as there are no dependencies, the individual queries in the batch can then be executed in parallel in the back end. InA also supports the combination of results of different multidimensional result sets, which have some of the requested dimensions in common.

Spatial Features

Information Access queries support features included in the SAP HANA Spatial Engine, for example:

- Geometry filters
  Show only stores inside a user-defined polygon.
- Spatial clustering
  Show all stores but do not return them as single points, but clustered by their location.
- Spatial join
  Show all the stores within a certain distance of points of interest.

Metadata

Metadata are an important part of all work flows involving Multidimensional Services (MDS), the SAP HANA analytical processing engine that is used to gather metadata from the SAP HANA database metadata tables (`_SYS_BI.BIMC_*`) for all the database artefacts used for analytics. Metadata are used to validate every single InA query and contain all the information a client needs to know about a data set. For example, the client application needs to know which dimensions, hierarchies, attributes, measures, etc. are defined in the data set and also what the data type of each underlying column is. Since the metadata names are used to generate the multidimensional query, the names must match the data in the database to ensure that the query can be executed.

The metadata are used to specify constraints for different parts of the data set, for example, if a dimension can be aggregated or needs to be on one of the visible axes all the time. The metadata also contains defaults, for example, what aggregation is specified for a particular measure column to be used, even if the user might want to change this information during the analytic work flows.

Related Information

Data Access with InA in XS Advanced [page 618]
Maintain the Analytics Adapter in XS Advanced [page 618]

7 Writing the XS Advanced Application Code

Add business logic to your XS advanced application to enable it to work with the database artifacts.

The business logic is used to help retrieve data from the database, for example, using OData services. The data is requested from and displayed in the client user interface.

The SAP HANA XS, advanced model, run-time platform provides a number of services for managing the various container instances and their application run times. Containers are used to manage run times and allow for isolation, resource management, and shared service injection. The XS advanced application run times are lightweight processes that are invoked over HTTP and communicate remotely with the database. Execution environments such as JavaScript, including Node.js, and Java are supported.

The application services can be used to expose the database data model, with its tables, views and database procedures, to clients. This can be done in a declarative way using OData services or by writing native application-specific code that runs in the SAP HANA context. You can also use SAP HANA XS advanced model to build dynamic HTML5 UI applications.

JavaScript Applications

For applications written in JavaScript (Node.js or XS JavaScript), you would typically expect the following design-time file structure:

Sample Code

```
JavaScript-AppName
|- db/                          # Database artifacts
|  |- package.json
|  |- src/
|     |- .hdiconfig
|     |- mytable.hdbtable
|     \- myview.hdbview
|
|- js/                          # JavaScript artifacts
|  |- server.js                 # JavaScript application entry point
|  |- package.json              # JavaScript application details/dependencies
|  \- lib/                      # JavaScript source code
|  \- node_modules/             # Downloaded node module packages
|
|- ui/
|  |- xs-app.json               # Application-router configuration
|  |- package.json
|  \- resources/
|
|- xs-security.json
|- package.json
\- mtad.yaml
```

- Add a package description file (`package.json`) to your JavaScript resource-files folder with details of your application's contents and dependencies.

- Add a start-up file for your JavaScript application, for example, `main.js`.
- Create express routes.
- Create JavaScript source files (`.js` or `.xsjs`). These files must be placed in the corresponding application module's folder, for example, `js/src/*.js` or `xsjs/src/*.xsjs`.

Java Applications

For applications written in Java, a multi-target application would typically have the following design-time structure:

Sample Code

```
Java-AppName
|- db/                          # Database artifacts
|  |- package.json
|  |- src/
|     |- .hdiconfig
|     |- mytable.hdbtable
|     \- myview.hdbview
|
|- java/                        # Java artifacts
|  |- pom.xml                   # Java project object model (Maven)
|  \- src/                      # Java source code
|
|- ui/
|  |- xs-app.json               # Application-router configuration
|  \- resources/
|
|- xs-security.json
|- package.json
\- mtad.yaml
```

- Write the Java application's code. These files must be placed in the corresponding application module's folder, for example, `java/src/*`
- Look up the required data-source.
- Build the WAR file based on the project object model file (pom.xml), for example, with Maven.
- Use the Java Persistence API (JPA) to work with CDS entities (if required).

Python Applications

For applications written in Python, a multi-target application would typically have the following design-time structure:

Sample Code

```
Python-AppName
|- db/                          # Database artifacts
|  |- package.json
|  |- src/
|     |- .hdiconfig
|     |- mytable.hdbtable
```

```
|     \- myview.hdbview
|
|- pyapp/                        # Python artifacts
|  |- runtime.txt                # Python app run-time version declaration
|  |- requirements.txt           # Python app dependencies list
|  |- manifest.yml               # Python app build/deploy description
|  \- server.py                  # Application logic
|
|- ui/
|  |- xs-app.json                # Application-router configuration
|  \- resources/
|
|- xs-security.json
|- package.json
\- mtad.yaml
```

Related Information

The SAP HANA XS Advanced JavaScript Run Time [page 627]
The SAP HANA XS Advanced Java Run Time [page 716]
The SAP HANA XS Advanced Python Run Time [page 778]
XS Advanced Application Resource Files [page 107]
Create the XS Advanced Application Package Descriptor [page 818]

7.1 The SAP HANA XS Advanced JavaScript Run Time

SAP HANA XS advanced model provides a JavaScript run time environment to which you can deploy your Node.js and XS JavaScript applications.

SAP HANA XS advanced (XS advanced) enables you to build and deploy applications on Node.js. XS advanced makes no assumption about the frameworks you are using. However, it is recommended to connect to (and use) the SAP HANA deployment infrastructure (HDI) container and validate using JSON Web Tokens (JWT).

> **i** Note
>
> SAP has decided to discontinue development of the SAP Node.js runtime that ships with SAP HANA XS advanced. The last version of SAP Node.js to ship with XS advanced will be version 14. For JavaScript runtime versions higher than Node.js 14, XS advanced will provide the community edition of Node.js. Migrating to the community edition of Node.js could involve changes related to supportability and development features. For more information about the important differences between the two Node.js versions as well as the changes required, see SAP Note 2997493.

Authentication for node applications in XS advanced relies on a special usage of the OAuth 2.0 protocol by the User Account and Authentication service (UAA). The UAA vouches for the authenticated user's identity using a JSON web token (JWT) as OAuth 2.0 token. This is a signed text-based token in JSON syntax. For more information, see the README.md file of the `@sap/xssec` module in the XS advanced Container Security API, for example, for your node.js application.

Node.js Modules for Download

A collection of Node.js packages developed by SAP is provided as part of XS advanced; the Node.js packages are available on the public NPM registry at `https://registry.npmjs.org`. Packages include, among other things, an application router, a job scheduler, logging tools, text mining and analysis tools, and so on. The following list shows a selection of the more commonly used packages available on the NPM Registry:

- `@sap/approuter`
- `@sap/hdbext`
- `@sap/hana-client`
- `@sap/hdi`
- `@sap/hdi-deploy`
- `@sap/jobs-client`
- `@sap/logging`
- `@sap/xsenv`
- `@sap/xsjs`
- `@sap/xssec`
- `@sap/xss-secure`

> → Tip
>
> For more information about configuring and using the NPM Registry, see *Related Links*.

To install an `npm` module use the following command:

```
npm install --save @sap/xsenv
```

To use a module, for example `@sap/xsenv`:

```
var xsenv = require('@sap/xsenv');
```

SAP HANA Database Connections

To establish a connection to the SAP HANA database, a JavaScript application must first look up the connection properties from the bound services and create a database client object. For Node.js applications, the easiest way to do this is to add a dependency to `@sap/hdbext` to the application's `package.json` file, as illustrated in the following example:

> ℹ Note
>
> You might need to modify the version of `@sap/hdbext` to reflect the version required by your application.

> ‹≡› Sample Code

```
"dependencies": {
"@sap/hdbext": "^5.0.0"
  },
```

Next, the application has to create a connection, as illustrated in the following example:

Sample Code

```
var hdbext = require('@sap/hdbext');
var hanaConfig = { ... };
hdbext.createConnection(hanaConfig, function(error, client) {
  if (error) {
      return console.error(error);
  }
  client.exec(...);
});
```

Connection Pools

XS advanced applications can use the connection pooling functionality provided by the `@sap/hana-client` package. Connection pooling is enabled by adding the property `pooling` with value `true` (JavaScript boolean) in the `hanaConfig` object.

Security

For Node.js, the client security library is an `npm` module called `@sap/xssec`. If you want your application to use the security library `@sap/xssec`, you must add the dependency to `@sap/xssec` in the `dependencies` section of your application's package description file, `package.json`, as illustrated in the following example:

Sample Code

```
"dependencies": {
    "@sap/xssec": "^1.0.5"
  },
```

Outbound Connectivity

If a Node.js application needs to call external applications or services, it must do so by performing HTTP requests to the required services. The service can either be located in the same system (for example, an SAP HANA XS advanced system) or on an external system on the Internet. For connections to remote systems, Node.js applications can make use of the "`request`" module. For outbound connections, you need to consider the following:

- Proxy connections
  If your Node.js applications runs behind an HTTP proxy, the details of the proxy connection must be provided to the `request` module. Request can discover this information either from environment variables set in the application's manifest file or directly from the options object of the HTTP request.
- Authentication
  If the application needs to call a service that performs authentication by means of Security Assertion Markup Language (SAML) and JWT (authentication), and the calling service does so as well, then the SAML token can be taken from the incoming request and attached to the outgoing request.

Sample Code

```
var options = {
    url: remoteURL,
    headers: {
        'authorization': req.headers.authorization
    }
};
```

Note

For this so called “principal propagation” to work, both the calling and the called application must be bound to the same User Account and Authentication (UAA) service. Otherwise the SAML token attached to the request will not be recognized and the request cannot be authenticated.

Optimizing Application Memory Usage

To define a limit for the memory used by your JavaScript application, set the `<OPTIMIZE_MEMORY>` environment variable to `true`. If the memory property is not specified in the `manifest.yml` file, the default memory limit is 1GB.

Note

The memory-limit feature works only for applications with Node.js version 6.12.x or higher. For versions of Node.js older than 6.12.x, you have to manually set the property `--max_old_space_size` in the `start` command of the Node.js application, as illustrated in the following example. The recommended value is 75% of the memory limit you want to add to your application.

Sample Code

```
node --max_old_space_size=96 server.js
```

Related Information

Writing the XS Advanced Application Code [page 625]
The SAP HANA XS Advanced Java Run Time [page 716]
Standard Node.js Packages for XS Advanced [page 635]
The NPM Registry [page 687]

7.1.1 Secure Programming with JavaScript

Web-based applications for deployment to the XS advanced run-time must adhere to the general standard for Web application security as defined by the W3C.

To ensure the best possible protection against attacks, Web-based SAP HANA applications must implement not only **primary** security measures such as input validation or output encoding, but also secondary measures, for example a Content Security Policy (CSP) that helps defend against script-based injection vulnerabilities. In addition to any other security measures, developers of SAP HANA XS advanced applications must ensure that the Web-based applications comply with the following basic rules:

- Avoid using inline `JavaScript()` functions in a Web page.
- Avoid using the `eval()` command.
- Avoid using strings (that represent code to be executed) in place of simple functions in code generating-functions such as `setTimeout()`, `setInterval()`, or `function()`.
- Ensure that platforms provide HTTP header-setting functions which perform the following actions:
  - Enable CSP headers that can be maintained or customized (for example, as a configuration option). This is possible in either of the following ways:
    - Indirectly
      CSP settings are derived from higher-level configurations, for example, system landscape data.
    - Directly
      CSP settings are derived from modifications to the configuration in more simple environments.
  - Enable applications to set CSP headers (for example, as an API option).
- Make use of functions for controlling frame inclusion (for example, frame “ancestors”).
- Ensure the implementation of primary protection measures such as “input validation” and “output encoding”.

> **i Note**
>
> A CSP compliant application will only work if the underlying platform is also CSP compliant. For example, a CSP-compliant Web browser will not execute instructions that contradict the defined security policy.

Recommended Coding Rules for JavaScript Applications

It is recommended to adhere to the following rules when coding JavaScript applications that are rendered by HTML in a Web browser, for example: SAPUI5-based Fiori applications, mobile applications using HTML technology, JSP applications, Web Dynpro, BSP applications, EP iViews:

- Security policy:
  The CSP should be as close as possible to the following example:

```
Content-Security-Policy: default-src 'self'; style-src 'self' 'unsafe-
inline'; img-src 'self' data:
```

> **i Note**
>
> Certain adjustments might be needed to help ensure functionality, for example, adding source lists for `script-src`.

- `<script>`
  Loading scripts from an allowed source is possible using the `<script>` tag, for example:

  ```
  <script src = "foo.js" type = "text/javascript" > </script >
  ```

- `onClick()`
  To handle events, you can set the event handler attributes of elements, or call `element.addEventListener()`, from a script that has been loaded from an allowed site, as illustrated in the following example:

  ```
  element.onclick = someFunction;
  element. addEventListener( "click" , someFunction, false ) ;
  ```

- `setInterval()`

  ```
  window. setInterval( myFunc, 10000 ) ;
  ```

- `setTimeout()`

  ```
  self.timers.invite2xxTimer = window.setTimeout(function()
  {invite2xxRetransmission(retransmissions)}, timeout);
  ```

Coding Practices to Avoid in JavaScript Applications

It is recommended to avoid the following commonly used practices when coding Web-based JavaScript applications for SAP HANA XS advanced:

- `<script>`
  Avoid including any executable code in an HTML string:

  ```
  <script type = "text/javascript" > /* executable code */ </script >
  ```

- `onClick()`
  Avoid including any executable code in an HTML string:

  ```
  <a onclick = "another_evil_script()"  > Click this for some awesome fun! </a
  >
  ```

  Avoid specifying a JavaScript command as part of an HTML attribute; this is forbidden by the CSP policy:

  ```
  <a href = "javascript:do_something_evil()" > Click here for some awesome fun!
  </a >
  ```

- `setInterval()`
  Avoid including any executable code in an HTML string:

  ```
  window. setInterval(  "evil_code()" , 10000 ) ;
  ```

- `setTimeout()`
  In the following short example, if "`invite2xxRetransmission`" returns a string instead of a pointer, it will be still evaluated using the `eval()`function:

  ```
  self.timers.invite2xxTimer =
  window.setTimeout(invite2xxRetransmission(retransmissions), timeout);
  ```

SSL Setup with Node.js Applications

SAP provides the package (`@sap/hdbext`) which Node.js applications can use to connect to SAP HANA Cloud. If you want to use it, you must add it to your list of application dependencies.

> → Tip
>
> To ensure that both SSL and trust certificates are set up automatically, it is recommended to use @sap/hdbext to connect to SAP HANA rather than use @sap/hana-client. For more details about connections to SAP HANA, see *Related Information* below.

Related Information

Getting Started with Application Development in XS Advanced [page 27]
The SAP HANA XS Advanced JavaScript Run Time [page 627]
Connect to SAP HANA [page 703]
https://www.npmjs.com/package/@sap/hana-client

7.1.2 Download and Consume SAP Node.js Packages

A selection of SAP-specific and ready-to-use Node.js packages is available for download from various sources.

Context

A Node.js application often depends on packages that are present in the public `npm` registry or on specific SAP modules. A selection of SAP packages for Node.js is available for use in your Node.js applications. If your Node.js application depends on any of the SAP NPM packages, you can configure a source to resolve the dependencies. If no npm registry is configured, the public registry at `registry.npmjs.org` is used.

The standard SAP NPM packages are only available in the public NPM registry at https://registry.npmjs.org or in the SAP Web IDE for SAP HANA's default NPM cache. The NPM registry is the recommended source.

> → Tip
>
> For information about how to configure your Node.js application to use the NPM registry at `registry.npmjs.org` to resolve package requirements and dependencies, see *The NPM Registry* in *Related Information* below.

The following instructions explain how to configure the use of the NPM Registry to resolve NPM package dependencies when building and deploying Node.js applications:

Procedure

1. Configure command-line tools to use the NPM Registry to resolve package dependencies.

   If you are using command-line tools to build your Node.js applications for XS advanced, you can configure the tools to use a particular NPM registry to resolve the dependencies between resources specified in the deployment descriptors.

   > **i Note**
   >
   > The target registry is configured by default during staging in XS advanced. For local development you still have the option of setting the registry manually.

   To specify the NPM Registry, add the `"@sap:registry="` property to the `.npmrc` resource-configuration file, as illustrated in the following example:

   > **‘≡› Sample Code**
   >
   > NPM Registry Configuration in the `.npmrc` File
   >
   > ```
   > ...
   > @sap:registry=https://registry.npmjs.org
   > ...
   > ```

   You can also use the `npm` command to configure the target registry for the current console session:

   ```
   $ npm config set @sap:registry https://registry.npmjs.org
   $ npm install @sap/<node_package>
   ```

2. Configure the SAP Web IDE for SAP HANA to use the NPM Registry to resolve package dependencies.

   To specify a specific NPM registry that will be used for the resolution of dependency requirements when developing and building applications with SAP Web IDE for SAP HANA, add the `"UPSTREAM_LINK"` and (or) `"SAPUPSTREAM_LINK"` properties to the `di-local-npm-registry` module in the MTA **extension** descriptor (for example, `sapwebide.mtaext`) that is used when deploying SAP Web IDE, as illustrated in the following example:

   > **i Note**
   >
   > If a suitable MTA extension descriptor does not already exist for the deployment of SAP Web IDE, you must create one and add the suggested configuration details for the NPM registries. SAP Note 2510063 includes a link to a sample MTA extension file for SAP Web IDE. Download the sample extension configuration, add the following NPM registries, and save it in the same location as the Web IDE component you want to install and extend.

   > **‘≡› Sample Code**
   >
   > MTA Extension Descriptor: NPM Registry Configuration for SAP Web IDE Deployment
   >
   > ```
   > ...
   > ```

```
modules:
  - name: di-local-npm-registry
    properties:
      UPSTREAM_LINK: "http://registry.npmjs.org"
...
```

Related Information

Standard Node.js Packages for XS Advanced [page 635]
The NPM Registry [page 687]
The SAP HANA XS Advanced JavaScript Run Time [page 627]
SAP Software Download Center (Logon required)
SAP Note 2510063
SAP Web IDE for SAP HANA - Installation and Upgrade Guide

7.1.2.1 Standard Node.js Packages for XS Advanced

A collection of Node.js packages developed by SAP is provided to help you develop Node.js applications for SAP HANA XS advanced.

SAP HANA includes a selection of standard Node.js packages, which are available for download and use from the NPM public registry at `https://registry.npmjs.org`. For more information about the NPM public registry as well as how to configure your XS advanced application to use the registry, see *Related Information* below. The following table lists the SAP Node.js packages that are currently available and indicates where you can find them. For more details about the contents of each Node.js package as well as any useful configuration tips, see the `README` file that corresponds to the NPM package on Nodejs.com for which you can find a link in *Related Information* below.

> **i Note**
>
> Some packages are available for download but are neither listed nor documented here, for example, packages that are included in the registry only because they are required by other packages.

SAP Node.js Packages and Contents

Package Name	Description
`@sap/audit-logging`	Utilities for audit logging
`@sap/approuter`	The application router is the single entry point for the (business) application. It serves static content, authenticates users, rewrites URLs, and routes requests to other micro services as well as propagating user information.

Package Name	Description
`@sap/cds`	Core Data Services (CDS) client for Node.js **i Note** The `@sap/cds` library version 1.15.x for XS advanced is now considered feature complete; it will remain fully supported but will not receive any further enhancements in future releases. Versions 2.x.x and 3.x.x of the library `@sap/cds` are only for use in SAP Cloud Application Programming Model (SAP CAP CDS) scenarios.
`@sap/dwf-deploy`	Node.js client for SAP HANA Datawarehousing Foundation 2.0 (HANA DWF 2.0) to deploy design-time artifacts
`@sap/dwf-generator`	Node.js client for SAP HANA Datawarehousing Foundation 2.0 (HANA DWF 2.0) to generate design-time artifacts
`@sap/dwf-dws-client`	Node.js client for SAP HANA Datawarehousing Foundation 2.0 (HANA DWF 2.0) to host task types for the Task Orchestration Engine
`@sap/e2e-trace`	Utilities for end-to-end tracing (in particular the handling of SAP Passports)
`@sap/edmx2csn`	A utility to convert an OData V2 model (EDMX) to Core Schema Notation (CSN)
`@sap/hana-client`	A client library for communication with the SAP HANA database
`@sap/hdb-connection`	Utility functions for access to SAP HANA in Node.js **i Note** The functions included in the `@sap/hdb-connection` library have been superseded; use `@sap/hdbext` instead.
`@sap/hdbext`	Extends the functionality of the `@sap/hana-client` package **i Note** The functionality included in the `@sap/hdbext` library replaces and supersedes the functions previously provided by the `@sap/hdb-connection` library.
`@sap/hdi-deploy`	The Node.js based “HDI deploy” application (also known as “DeployApp”) which is based on HDI's SQL interface
`@sap/hdi-dynamic-deploy`	A Node.js-based HTTP server for the deployment of database content to dynamically created SAP HANA DI (HDI) containers
`@sap/hdi`	A Node.js-based client library for SAP HANA DI (HDI)
`@sap/instance-manager`	Node.js package for creating and deleting service instances per tenant within an XS advanced application at run time

Package Name	Description
`@sap/jobs-client`	A client library for the Job Scheduler service provided by XS advanced
`@sap/logging`	Provides logging and tracing functions for Node.js applications. Logs and traces are written in the standard SAP format in XS advanced and in the appropriate JSON format in CloudFoundry (from version ^4.0.0).
`@sap/node-vsi`	A Node.js client that contains the Virus Scanning Interface (VSI) binding for Node.js
`@sap/sds-deploy`	Node.js client for the SAP HANA Streaming Analytics option
`@sap/site-entry`	SAP Portal site entry point
`@sap/site-content-deployer`	Client for portal site content deployer
`@sap/site-app-server`	An application server for the independent apps scenario on the SAP portal
`@sap/textanalysis`	A utility for performing text analysis on unstructured text
`@sap/textmining`	A utility for performing text mining on unstructured text
`@sap/textbundle`	A simple tool for text internationalization in Node.js
`@sap/ui-annotations`	Annotations for OData version 4 in Core Data Services (CDS)
`@sap/xsenv`	Utility for easy setup and access of XS Advanced environment variables and services
`@sap/xsjs`	The compatibility layer that allows SAP HANA extended application services, classic model, applications to run in the JavaScript run time environment for SAP HANA extended application services, advanced model
`@sap/xsjs-test`	Unit test framework for the compatibility layer (XS classic runtime)
`@sap/xssec`	XS advanced security API for Node.js, including the XS advanced container security API for Node.js
`@sap/xss-secure`	Utilities for protection against cross-site scripting
`@sap/xb-msg`	Provides messaging capabilities with a message broker
`@sap/xb-msg-env`	Provides the functions needed to set up messaging client options from Cloud Foundry or XS advanced environment variables
`@sap/xb-msg-amqp-v091`	Provides a client implementation for AMQP v0.9.1

Package Name	Description
`@sap/xb-msg-amqp-v100`	Provides a client implementation for AMQP v100
`@sap/xb-msg-mqtt-v311`	Provides a client implementation for MQTT v3.1.1

@sap/approuter

The application router is the single entry point for the (business) application. The main tasks of the application router are to serve static content, authenticate users, rewrite URLs, and proxy requests to other micro services while propagating user information. For more information about the Node.js application router and how to configure it for your XS advanced application, see the section *Maintaining XS Advanced Application Routes and Destinations* in this guide or the *Related Links* section below.

@sap/cds

The Core Data Services for Node.js (`@sap/cds`) comprise a JavaScript client library for Core Data Services that enable Node.js applications to consume CDS artifacts natively in Node.js applications.

> **i** Note
>
> The `@sap/cds` library version 1.15.x for XS advanced is now considered feature complete; it will remain fully supported but will not receive any further enhancements in future releases. Versions 2.x.x and 3.x.x of the library `@sap/cds` are only for use in SAP Cloud Application Programming Model (SAP CAP CDS) scenarios.

Usage

> **‹≡›** Sample Code

```
var cds = require('@sap/cds');
```

CDS entities are imported by name. The import function takes a callback that is invoked when all imports have completed. Additional fields and overrides can be supplied for each entity.

> **i** Note
>
> The import operation is a regular asynchronous Node.js function; you can import entities at any point in time, and in as many calls as you want.

> **‹≡›** Sample Code

```
cds.importEntities([
    { $entity: "xsds.test.cds::ds_test.e1" },
```

```
    { $entity: "xsds.test.cds::ds_test.e2",
      $fields: {
          a: { $association: "xsds.test.cds::ds_test.e2",
               $viaBacklink: "b" }
      }
    }
], callback);
function callback(error, entities) {
    var E1 = entities["xsds.test.cds::ds_test.e1"];
    var E2 = entities["xsds.test.cds::ds_test.e2"];
    // ...
}
```

@sap/dwf-deploy

This package includes a Node.js client for SAP HANA Datawarehousing Foundation (HANA DWF). The following table describes the file-system layout required by the `HANA DWF Client` application:

HANA DWF Client File System Layout

File	Mandatory	Description
`package.json`	Yes	Used by the Node.js package manager (`npm`) to boot and start the application.
`src/`	Yes	Subdirectory containing `DWF` deployment definition files (optional, additional subfolders are allowed).
`cfg/`	No	Additional deployment subdirectories ...

> **i** Note
>
> All DWF-related files in the **root** directory are ignored by the `HANA DWF Client` application.

@sap/dwf-generator

This package includes a Node.js client for SAP HANA Datawarehousing Foundation 2.0 (HANA DWF 2.0) to generate designtime artifacts. The following table describes the file-system layout required by the `HANA DWF Client` application:

HANA DWF Client File System Layout

File	Mandatory	Description
`package.json`	Yes	Used by the `dwf-generator` to discover the `dwf-config`.

File	Mandatory	Description
`src/`	Yes	Subdirectory containing `DWF` deployment definition files (optional, additional subfolders are allowed).
`src/dlm_generated`	Yes	Subdirectory containing `XSA` or `DWF` deployment definition files (optional, additional subfolders are allowed).
`cfg/`	No	Additional deployment subdirectories, which are created in the designtime
`cfg/dlm_generated`	No	Additional deployment subdirectories, which are generated, based on the deployment definition files that have been created in the designtime.

> **i** Note
>
> All DWF-related files in the **root** directory are ignored by the `HANA DWF Client` application.

@sap/dwf-dws-client

This package includes a Node.js client for SAP HANA Datawarehousing Foundation 2.0 (HANA DWF 2.0) to host task types for the `Task Orchestration Engine`. The module does not imply any requirements to the file system structure.

@sap/edmx2csn

The `EDMX2CSN` utility converts an OData V2 model (EDMX) into Core Schema Notation (CSN), which is a JSON-like format that enable the compact representation of data and service models. This utility is primarily intended for developers who are building an extension application that connects to a remote OData V2 data source such as SAP S/4HANA. In such a case, converting the EDMX model from the OData V2 data source into CSN is the first step you take, along with building the CDS data model, in the process of defining the CDS service model for your application.

Usage

The compiler with its options is invoked like any other `npm` or Unix command:

- To generate a CSN file from an EDMX **file**, use the following command:
  - Windows (`edmx2csn.cmd`):

```
$homedir>./node_modules/.bin/edmx2csn.cmd -i ${input_folder}/metadata.xml
-o ${output_folder} -f
```

- Linux: (`edmx2csn.sh`):

```
$homedir>./node_modules/.bin/edmx2csn.sh -i ${input_folder}/metadata.xml
-o ${output_folder} -f
```

- To generate a CSN file from an EDMX **URL**, use the following command:
  - Windows (`edmx2csn.cmd`):

```
$homedir>./node_modules/.bin/edmx2csn.cmd -i ${service_url}/$metadata -o $
{output_folder}
```

  - Linux: (`edmx2csn.sh`):

```
$homedir>./node_modules/.bin/edmx2csn.sh -i ${service_url}/$metadata -o $
{output_folder}
```

The following table provides details of the options used in the examples of the `edmx2csn` command above:

Option	Description
-i	Specifies the name and location of the OData model to be converted to CSN (using either a file path or the URL of a service metadata end point)
-o	Specifies the location of the CSN file produced by the conversion process
-f	Overwrite an existing CSN output file

@sap/hdb-connection

This package comprises a small library that enables you to establish a connection to an SAP HANA server using the credentials either from stand-alone environments or, if required, from a Cloud Foundry environment, for example defined in the "`VCAP_SERVICES`" environment variables.

> ! Restriction
>
> The functions included in the `@sap/hdb-connection` library have been superseded; use `@sap/hdbext` instead.

@sap/hdbext

This is a small Node.js package that extends the functionality of the `@sap/hana-client` package. The functionality included in the `@sap/hdbext` library replaces and supersedes the functions previously provided by the `@sap/hdb-connection` library.

> → Tip
>
> To ensure that both SSL and trust certificates are set up automatically, it is recommended to use @sap/hdbext to connect to SAP HANA rather than use @sap/hana-client.

Usage

Sample Code

```
var hdbext = require('@sap/hdbext');
```

The following code example shows how to create a connection to the SAP HANA database:

Sample Code

```
var hanaConfig = {
  host : 'hostname',
  port : 30015,
  user : 'user',
  password : 'secret'
};
hdbext.createConnection(hanaConfig, function(error, client) {
  if (error) {
    return console.error(error);
  }
  client.exec(...);
});
```

If you are using table parameters,with the new API you can pass an array of objects and it will auto convert it into a table parameter, as illustrated in the following example:

Sample Code

```
create procedure PROC_DUMMY (in a int, in b int, out c int, out d DUMMY, out
e TABLES)
  language sqlscript
  reads sql data as
  begin
    c := :a + :b;
    d = select * from DUMMY;
    e = select * from TABLES;
  end
```

With the `loadProcedure()` API, you can make the following call:

Sample Code

```
hdbext.loadProcedure(client, 'MY_SCHEMA', 'PROC_DUMMY', function(err, sp) {
  sp({ A: 3, B: 4 }, function(err, parameters, dummyRows, tableRows) {
    if (err) {
      return console.error(err);
    }
    console.log('C:', parameters.C);
    console.log('Dummies:', dummyRows);
    console.log('Tables:', tableRows);
  });
});
```

To use connection pooling, specify the property `pooling` with value of `true` (JavaScript boolean) in the `hanaConfig` object.

To acquire a client from the connection pool, use the `pool.acquire` function, as illustrated below:

Sample Code

```
pool.acquire(options, function(err, client) {});
```

Note

Use the `options` object if you need to change the settings configured for the pooled connection.

If the client is no longer needed, release it to the connection pool, as illustrated in the following example:

Sample Code

```
pool.release(client);
```

Alternatively, you can close the client connection, as illustrated below:

Sample Code

```
client.close();
```

@sap/hana-client

This package comprises a small client library for communication with the SAP HANA database. The `@sap/hana-client` driver supports node.js 4.x, 6.x and 8.x.

Tip

Since this driver communicates with the native SAP HANA libraries, it requires platform-specific native binaries.

Usage

Sample Code

```
var hana = require('@sap/hana-client');
```

The following code example shows how to get started with setting up a connection to the SAP HANA database:

Sample Code

```
var hana = require('@sap/hana-client');
var conn = hana.createConnection();
var conn_params = {
  serverNode : 'myserver:30015',
  uid        : 'system',
  pwd        : 'manager'
};
conn.connect(conn_params, function(err) {
```

```
  if (err) throw err;
  conn.exec('SELECT Name,
            Description FROM Products WHERE id = ?', [301],
            function (err, result) {
    if (err) throw err;
    console.log('Name: ', result[0].Name, ', Description: ',
                result[0].Description);
    // output --> Name: Tee Shirt, Description: V-neck
    conn.disconnect();
  })
});
```

A database connection object is created by calling `'createConnection'`. The connection is established by calling the connection object's `'connect'` method and passing in an object that represents the connection parameters. The following code example shows how to set up a connection to the SAP HANA database:

Sample Code

```
conn.connect({
  host : 'myserver',
  port : '30015',
  uid  : 'system',
  pwd  : 'manager'
});
```

You can call `'conn.disconnect'` to close the client connection, as illustrated in the following example:

Sample Code

```
conn.disconnect(function(err) {
  if (err) throw err;
  console.log('Disconnected');
});
```

For more information about how to use the features included in the `@sap/hana-client` package, see the `README.md` file included in the NPM client archive you installed. For more information about Node.js application programming with the SAP HANA client, see *SAP HANA Client Interface Programming Reference for SAP HANA Platform* in *Related Information* below.

@sap/hdi-deploy

`@sap/hdi-deploy` is the Node.js deploy application (known as "`DeployApp`") for the SAP HANA Deployment Infrastructure (HDI), which is based on the HDI's SQL interface. The following table describes the file-system layout required by the `DeployApp` application:

HI Deployer File System Layout

File	Mandatory	Description
`package.json`	Yes	Used by the Node.js package manager (`npm`) to boot and start the application.

File	Mandatory	Description
`src/`	Yes	Subdirectory containing `.hdiconfig` and HDI deployment definition files (optional, additional subfolders are allowed).
`cfg/`	No	Additional deployment subdirectories ...

> **i** Note
>
> All HDI-related files in the **root** directory are ignored by the `DeployApp` application.

For more information about setting up and using the Node.js-based SAP HANA Deployment Infrastructure (HDI) Deployer for XS advanced, see *Configuring the HDI Deployer* in the *Related Links* below.

Environment Variables

The following table lists the environment variables that can be used with the `DeployApp` application:

HDI Deployer Environment Variables for XS Advanced Applications

Variable	Mandatory	Description
`TARGET_CONTAINER`	Yes	Service name that specifies the HDI target container (required if more than one service is bound to the `DeployApp` application).
`SERVICE_REPLACEMENTS`	No	JSON-structured list of service replacements, `[ { "key": "logical-service-name-1", "service":"real-service-name-1" }, { "key": "logical-service-name-2", "service":"real-service-name-2" } ]` The **logical** service names refer to the names in the HDI content; the **real** service names refer to the services which are bound to the HDI Deployer via `VCAP_SERVICES`. If the HDI content references a service name which is not listed in the replacements, this name is used as a real service name.

The structure of the `<SERVICE_REPLACEMENTS>` environment variable illustrated in the following example is intended to enable and facilitate MTA group assignments.

> **‹≡›** Code Syntax
>
> Service Replacements list in the `manifest.yml` file

```
applications:
- name: app-db
  path: db
  services:
    - app-database
    - real-grantor-service
    - real-external-service
```

```
  env:
    TARGET_CONTAINER: app-database
    SERVICE_REPLACEMENTS:
    [
      {
        "key" : "logical-grantor-service",
        "service" : "real-grantor-service"
      },
      {
        "key" : "logical-external-service",
        "service" : "real-external-service"
      }
    ]
```

Removing Deployed SAP HANA Node.js Artifacts

The `DeployApp` deletes (recursively) all artifacts found in the folder `src/` (and `cfg/`, if the optional subfolder exists) before the deployment of changed or new artifacts, but deleted artifacts are not removed; they remain deployed. To remove deployed artifacts, you have the following options:

- Specify the deployed files to remove in an `undeploy.json` file placed in the root directory of the deployment artifacts; the file lists the paths to and the name of the artifacts to remove:

  Sample Code

  ```
  [
    "src/AddressBook.hdbcds"
  ]
  ```

- Start the `DeployApp` with the `--autoUndeploy` option
  `node deploy --autoUndeploy`

@sap/hdi-dynamic-deploy

This module provides a Node.js-based HTTP server that can be used to deploy database content to dynamically created HDI containers, for example, containers created by means of the XS advanced Instance Manager. The HDI Dynamic Deployer is intended for use in not only in XS Advanced on-premise scenarios but also in the SAP Cloud Platform for Cloud Foundry (CF). `@sap/hdi-dynamic-deploy` is based on `@sap/hdi-deploy` which should be used if a static deployment at deployment time is sufficient.

Usage

Typically, `@sap/hdi-dynamic-deploy` is installed by including a dependency in the `package.json` file of your XS advanced application's database module, as illustrated in the following example:

Sample Code

`myApp/db/package.json`

```
{
  "name": "deploy",
  "dependencies": {
     "@sap/hdi-dynamic-deploy": "1.5.9"
  },
  "scripts": {
```

```
    "start": "node node_modules/@sap/hdi-dynamic-deploy/"
  }
}
```

For more detailed information about using the `@sap/hdi-dynamic-deploy`, see the `README.md` file included in the NPM package.

Configuration

Before using the HDI Dynamic Deployer, you must configure the following connection-related environment variables:

- `PORT`
  The port the HTTP server listens on
- `hdi_dynamic_deploy_user`
  The user name to supply for HTTP-based basic authentication
- `hdi_dynamic_deploy_password`
  The password to use for HTTP-based basic authentication

The `PORT` variable is set automatically by XS advanced; both the user name and password must be defined in the XS advanced application's development descriptor, `mta.yaml`, as illustrated in the following example:

Code Syntax

HDI Dynamic Deployer Configuration in `mta.yaml`

```
modules:
  - name: db
    type: com.sap.xs.hdi-dynamic
    path: db
    properties:
      hdi_dynamic_deploy_user: ${generated-user}
      hdi_dynamic_deploy_password: ${generated-password}
    provides:
    - name: db_deployment
      properties:
         url: ${default-url}
         user: ${generated-user}
         password: ${generated-password}
```

Multi-Tenancy Scenarios

A multi-target application (MTA) for use in multi-tenancy scenarios, for example, with the XS advanced Instance Manager, typically includes multiple database modules:

- N static database modules (`type: com.sap.xs.hdi`),
  For configuration or shared data. A static module depends on `@sap/hdi-deploy`; it does not depend on `@sap/hdi-dynamic-deploy`.
- M dynamic database modules (`type: com.sap.xs.hdi-dynamic`)
  Which contain the business data for a certain type of tenant. A dynamic module depends on `@sap/hdi-dynamic-deploy`, which internally depends on `@sap/hdi-deploy` for deployment to the correct tenant.

Code Syntax

Dynamic Deployment in a Multi-Tenancy Scenario

```
modules:
```

```
  - name: db-static-1
  - type: com.sap.xs.hdi
  - path: db-static-1
  - name: db-static-2
    type: com.sap.xs.hdi
    path: db-static-2
  - name: db-dynamic-1
    type: com.sap.xs.hdi-dynamic
    path: db-dynamic-1
  - name: db-dynamic-2
    type: com.sap.xs.hdi-dynamic
    path: db-dynamic-2
  - name: db-dynamic-3
    type: com.sap.xs.hdi-dynamic
    path: db-dynamic-3
```

@sap/hdi

This module provides a Node.js-based client library for SAP HANA DI (HDI). The client library provides the following asynchronous methods to enable access to HDI functionality:

- `connect`
- `disconnect`
- `configureDI`

  > ⚠ Caution
  >
  > Deprecated; use `configureDIParameters` instead.

- `configureDIParameters`
- `createContainer`
- `dropContainer`
- `configureContainer`

  > ⚠ Caution
  >
  > Deprecated; use `configureContainerParameters` instead.

- `configureContainerParameters`
- `listLibraries`
- `configureLibraries`
- `listConfiguredLibraries`
- `status`
- `read`
- `listDeployed`
  Read object-related metadata.
- `readDeployed`
  Read object-related metadata **and** its content.
- `write`
- `delete`

- `make`
- `makeAsync`
- `grantContainerApiPrivileges`
- `grantContainerApiPrivilegesWithGrantOption`
- `revokeContainerApiPrivileges`
- `grantContainerSchemaPrivileges`
- `revokeContainerSchemaPrivileges`
- `grantContainerSchemaRoles`
- `revokeContainerSchemaRoles`

Overview

After installing the module using `npm`, an application must create an instance of the client, for example:

```
hdi = new HDI(container, logger, credentials);
```

In this example:

- "`container`"
  The name of an existing HDI container
- "`logger`"
  A callback function used to write logging information
- "`credentials`"
  An object containing the host name, port, user, and password that are used to establish the connection to the database, for example:

  ```
  { host : '<myhost.kumasi.acme.com>',
    port : 30015,
    user : 'Kwame',
    password : 'etisei!131'}
  ```

The connection to the database is established with the `connect()` method, which takes only a callback function as parameter. When the application finishes, it should close the connection by calling the method:

```
hdi.disconnect();
```

@sap/instance-manager

`@sap/instance-manager` is a Node.js package for creating and deleting service instances per tenant within an XS advanced application at runtime.

Usage

The `@sap/instance-manager` package provides a client for the Instance Manager: a component that creates and deletes service instances (by means of a REST API) for a specified key. Instance Manager can be used in the context of multi-tenant applications where the key an instance is associated with is the tenant ID.

With the `@sap/instance-manager` package, a Node.js application can dynamically create and delete service instances per tenant at run time. An instance of a managed service of the desired type is first created and then bound to the application. For an SAP HANA database, the managed service is called "`managed-hana`". The

service binding only provides parameters (for example, HTTP end points and credentials) which can later be used by the application for creating and deleting service instances of the desired type for each tenant.

Application Programming Interface (API)

The following example shows how to use the instance manager API to manage service instances for the XS advanced application used by multiple tenants:

Sample Code

XS Advanced Service-Instance Manager API

```
var createInstanceManager = require('@sap/instance-manager').create;
var options = { /* properties from service binding */ };
createInstanceManager(options, function (err, instanceManager) {
  if (err) {
    return console.log('Create instance manager error:', err.message);
  }
  instanceManager.create('my-tenant', function (err, instance) {
    if (err) {
      return console.log('Create error:', err.message);
    }
    // consume instance.credentials
    console.log(instance);
    instanceManager.get('my-tenant', function (err, instance) {
      if (err) {
        return console.log('Get error:', err.message);
      }

      // same instance
     console.log(instance);
      instanceManager.delete('my-tenant', function (err) {
        if (err) {
         return console.log('Delete error:', err.message);
        }
        console.log('Instance deleted');
      });
    });
  });
});
```

Options

The managed service that is bound to the multi-tenant application (for example `"managed-hana"`) provides credentials as well as REST-based end points of the Instance Manager, which is the component that handles the creation and deletion of services. The credentials and the end points are mandatory.

The service `create` and `delete` operations are executed asynchronously on the server side. This library also implements polling until an operation is finished, and the polling can be tuned by means of some optional properties. Since operations involve network activity, the package also caches the created instances. The cache options can also be provided by developers.

Instance Manager Configuration Options

Property	Mandatory	Description	Default
`user`	□	User name for authentication	-
`password`	□	Password for `"user"`	-

Property	Mandatory	Description	Default
`post_managed_instance_url`	□	The REST end point used for creating a new service instance for a tenant	-
`get_managed_instance_url`	□	The REST end point used for retrieving details of a specific tenant service instance	-
`get_all_managed_instances_url`	□	The REST end point used for retrieving details of **all** instances (for **all** tenants)	-
`delete_managed_instance_url`	□	The REST end point used for deleting a service instance	-
`polling_interval_millis`	-	The number of milliseconds to wait between requests in the polling phase	300ms
`polling_timeout_seconds`	-	The maximum amount of time (in seconds) that can be spent in polling.	120 secs
`cache_max_items`	-	The maximum size of the cache.	500
`cache_item_expire_seconds`	-	The maximum number of seconds after which a cache entry expires.	600 secs

> **i** Note
>
> A managed service binding contains all of the mandatory properties mentioned above.
>
> It is recommended to have a single instance-manager JavaScript object for each managed service that is bound to the application.

Methods

The following table lists the methods available with the instance-manager API:

Instance-Manager API Methods

Method	Description
`create`	Creates a service instance for the provided tenant. The method polls until the instance is successfully created and then invokes the callback. Reports error if an instance for this tenant already exists.

Method	Description
`get`	Retrieves the corresponding service instance for the specified tenant either from the cache or from the server. A returned value of `null` indicates that no service instance exists for the specified tenant. **! Restriction** The `get` method only polls if the service instance is in status `CREATION_IN_PROGRESS`. In all other cases it returns the service instance as it is on server. There is no guarantee that the `credentials` property on the `instance` object is included in the callback.
`delete`	Removes the service instance for the provided tenant. The method polls until the instance is successfully deleted and then invokes the callback. An error is reported if no instance exists for the specified tenant.

→ Tip

If the callback of a method is invoked with an error and the error is caused by an unexpected HTTP response code received from the server, then the error object will have a `statusCode` property with the same status code as the received HTTP response.

Debug Logs

You can enable debug logs for the `@sap/instance-manager` package by adding `instance-manager` to the `<DEBUG>` environment variable.

@sap/jobs-client

`@sap/jobs-client` is a small Node.js package to integrate jobs into your Node.js application. The package contains utilities that enable you to create REST calls conforming to the format expected by the SAP HANA XS advanced job scheduler service, which is used to register or unregister jobs, update job schedules, and display the job status.

This package works with job descriptor objects and uses properties defined according to the requirements of the corresponding service in Job Scheduler.

Usage

Sample Code

```
var jobsc = require('@sap/jobs-client');
var options = {
  host: 'localhost',
  port: 4242,
  timeout: 15000,
  user: 'username',
  password: 'password',
  baseURL: 'http://apphost:port/'
};
var myJob = { /* according to job scheduler documentation */ };
```

```
var scheduler = new jobsc.Scheduler(options);
var scJob = { job: myJob };
scheduler.createJob(scJob, function (error, body) {
  if (error) {
    return console.log('Error registering new job %s', error);
  }
  // job was created successfully job.id = body._id;
});
```

@sap/logging

Provides logging and tracing functions for Node.js applications; logs and traces are written using the standard SAP formats in XS advanced and in the appropriate JSON format in CloudFoundry (from version ^4.0.0).

Usage

Sample Code

```
var logging = require('@sap/logging');
var express = require('express');
var app = express();
var appContext = logging.createAppContext();
app.use(logging.middleware({ appContext: appContext, logNetwork: true }));
app.get('/demo', function (req, res) {
  var logger = req.loggingContext.getLogger('/Application/Network');
  var tracer = req.loggingContext.getTracer(__filename);
  logger.info('Retrieving demo greeting ...');
  tracer.info('Processing GET request to /demo');
  res.send('Hello World!');
});
app.listen(3000, function() {
  console.log('Server started');
});
```

You need to perform the following actions:

1. Create the application context by initializing the logging library with some application-wide options.
2. Specify the middleware to use to extract information about a specific request. The specified middleware is used to extract request-specific information and attach the `loggingContext` property to the request object.

   > **i** Note
   >
   > This middleware definition should come **before** the `request-user` is set.

3. Instantiate a logger and a tracer by using the `loggingContext` property of the request.
4. Log and trace whatever you need.

Severity Levels

The following severity levels can be used in logging and tracing:

- Logging:
  "**info**" (default), "warning", "error", and "fatal"
- Tracing:

"debug", "path", "info", "warning", "**error**" (default), and "fatal"

If you deploy your application to the XS advanced On-Premise run time, you can change the severity level of categories and script files dynamically at run time without needing to restart the deployed application. To set (or reset) the logging level at run time, use the `xs set-logging-level` command, as illustrated in the following example:

Sample Code

```
xs set-logging-level
```

```
xs set-logging-level <application-name> <category-or-path-to-file> <logging-level>
```

It is possible to set a maximum priority level for all loggers and tracers by using the environment variable `XS_APP_LOG_LEVEL`. Its value is a level from "`debug`" to "`fatal`". The value from the environment variable (if valid) will be used instead of all already set levels. This is especially suitable for debugging purposes.

Note

If `XS_APP_LOG_LEVEL = none`, all logging and tracing is disabled. This setting is useful for automated testing.

Using Loggers

The following example shows how to create a logger:

Sample Code

```
var logger = req.loggingContext.getLogger('/Application/Network');
```

The request context has the "`getLogger`" function that takes a single string argument; namely, the "category". Categories define the names of functional areas in an application. It is recommended to prefix the name of your categories with "`/Application`", for example, `/Application/Network`. The categories form a hierarchy with forward slash as the separator. The following code shows how to use the `getLevel()` command to retrieve the severity level (as a string) of a logger:

Sample Code

```
var level = logger.getLevel();
```

It is also possible to check whether an entry with a specific severity level will be logged with the current level configuration:

Sample Code

```
var willItBeLogged = logger.isEnabled('info');
```

The following logging entries are available: "`info`", "`warning`", "`error`", and "`fatal`".

Sample Code

```
logger.info('Successful login of user %s - ', user, new Date());
```

```
logger.warning('Job did not complete successfully. An app admin must
retrigger the job.');
logger.error(new Error('Sorry! An error occurred'));
logger.fatal('A fatal error has occurred; the application has stopped!');
```

Using Tracers

To obtain a tracer, use the `getTracer()` command, as illustrated in the following example:

Sample Code

```
var tracer = req.loggingContext.getTracer(__filename);
```

In the same way as with loggers, you can also retrieve and check tracer instances, as illustrated in the following example:

Sample Code

```
var level = tracer.getLevel();
var willItBeTraced = tracer.isEnabled('path');
// etc.
```

The tracing API provides the following methods to enable you to set up and use tracing in Node.js applications:

- Entering
  Record that a function has been entered in the program flow. You can pass all of the arguments of your function to the entering function so that they are traced.
- Exiting
  Typically used in combination with the “entering” method. If you provide an argument, it will be considered as the return value of your function.
- Throwing
  Used to trace when the code is about to throw an error; you can pass the error that is about to be thrown as an argument.
- Catching
  Used in catch blocks; you can pass the caught error as an argument.

Sample Code

Tracing Methods: Entering and Exiting

```
function myFunction(tracer, a, b ,c) {
  tracer.entering(a, b, c);
  var result = // some logic here ...
  tracer.exiting(result);
  return result;
}
```

Sample Code

Tracing Methods: Throwing and Catching

```
function func1(tracer) {
  var error = new Error('An error has occurred');
  tracer.throwing(error);
  throw err;
```

```
}
function func2(tracer) {
  try {
    func1(tracer);
  } catch (err) {
    tracer.catching(err);
    // logic for processing the error
  }
}
```

@sap/textanalysis

The `@sap/textanalysis` package is a Node.js module on the XS advanced platform that supports text analysis. Text analysis performs linguistic analysis and entity extraction on unstructured text documents.

`@sap/textanalysis` is implemented as an interface to the SQL stored procedure `TA_ANALYZE` and provides the single API function: `analyze()`.

For more information about text analysis and the `TA_ANALYZE` stored procedure, see the *Text Analysis Developer Guide*.

Usage

First, a connection must be established to the SAP HANA database. Then the client database object can be passed to the `analyze()` method along with the input parameters. The input parameters set the input variables for the `TA_ANALYZE` stored procedure.

Sample Code

```
var ta = require('@sap/textanalysis');
var client;
async.series([
  function connect(callback) {
    client = hdb.createClient(options);
    client.connect(callback);
  },
  function analyze(callback) {
    var values = {
      DOCUMENT_TEXT: '<!DOCTYPE html><html><body><h1>My First Heading</
h1><p>My first paragraph.</p></body></html>',
      LANGUAGE_CODE: 'EN',
      CONFIGURATION: 'EXTRACTION_CORE',
      RETURN_PLAINTEXT: 0
    };
    ta.analyze(values, client, function done(err, parameters, rows) {
      if (err) { return console.error('error', err); }
      callback();
    });
  },
  function end(callback) {
    client.end(callback);
  }], done
);
```

@sap/textmining

The `@sap/textmining` package is a Node.js module on the XS advanced platform that supports text mining. Text mining makes determinations about the content of unstructured text documents by examining the terms used within them.

`@sap/textmining` is implemented as an interface to the Text Mining SQL API in SAP HANA. The API functions and parameters follow the pattern of the Text Mining XS classic API:

- `categorizeKNN()`
- `getRelatedDocuments()`
- `getRelatedTerms()`
- `getRelevantDocuments()`
- `getRelevantTerms()`
- `getSuggestedTerms()`
- `initialize()`

For more information about the Text Mining SQL API, see the section about Advanced Data Processing in the *SAP HANA SQL and System Views Reference*; for information about `initialize()`, see the section *ALTER FULLTEXT INDEX*.

For more information about the Text Mining XS classic API, see the *SAP HANA Text Mining XS JavaScript API Reference*.

For more information about text mining in general, see the *SAP HANA Text Mining Developer Guide*.

Usage

Sample Code

```
var textmining = require('@sap/textmining');
var hdb = require('hdb');
var db = {
  "host": "HOST",
  "port": 3XX15,
  "user": "USERNAME",
  "password": "PASSWORD"
}
var client = hdb.createClient({
  host: db.host,
  port: db.port,
  user: db.user,
  password: db.password
});
var p = {
  inputTermText : "term",
  top : 10,
  threshold : 0.3
}
var config = {
  client : client,
  referenceTable : 'SCHEMA.TABLE',
  referenceColumn : 'COLUMN'
}
var tmd = new textmining(config);
client.connect(function(err){
  if(err)
  {
    client.end();
```

```
    throw err;
    return;
  }
  tmd.getSuggestedTerms(p, function(err, result){
    if(err)
    {
      client.end();
      throw err;
      return;
    }
    console.log(result);
    client.end();
  });
});
```

@sap/textbundle

This is a small Node.js package to help enable text for internationalization in Node.js. Based on the same concept as SAP UI5, this package works with UTF-8 encoded `.properties` files. The language default feature is also borrowed from SAP UI5 with the idea that both the UI and the server-side code use the same approach to text internationalization.

Usage

Assuming that you have the following properties files in the folder `./test/properties`:

- `i18n_en_EN.properties`

  Code Syntax

  ```
  greeting = Hello {0}, you are {1} years old.
  ```

- `i18n_de.properties`

  Code Syntax

  ```
  greeting = Hallo {0}, Sie sind {1} Jahre alt.
  ```

Sample Code

```
var TextBundle = require('@sap/textbundle');
var bundle = new TextBundle({path: './test/properties/i18n', locale:
'en_EN' } );
bundle.getText('greeting'); // returns 'Hello {0}, you are {1} years old.'
bundle.getText('greeting', ['Stefan']); // returns 'Hello Stefan, you are
undefined years old.'
bundle.getText('greeting', ['Stefan', '21']); // returns 'Hello Stefan, you
are 21 years old.'
// using DE locale
bundle = new TextBundle({path: './test/properties/i18n', locale: 'de' } );
bundle.getText('greeting', ['Stefan', '21']); // returns 'Hallo Stefan, Sie
sind 21 Jahre alt.'
```

Loading Text Bundles

Bundles can be loaded by providing the absolute path to the resource bundle or relative path. The example above shows how to load a bundle using a relative path. You can also load the bundle with an absolute path:

Code Syntax

```
var TextBundle = require('@sap/textbundle');
var path = require('path');
var bundle = new TextBundle({path: path.resolve(__dirname, './test/properties/
i18n'), locale: 'de' } );
```

The default file extension is `.properties`. If your file has a different extension, you must append it to the path you provide. The format of the file must be UTF-8, and the structure has to conform to the `.properties` file format.

Code Syntax

```
var TextBundle = require('@sap/textbundle');
var txtBundle = new TextBundle({path: './test/txt/i18n.txt'});
```

@sap/xsenv

This is a small Node.js package to enable easy setup of (and access to) environment variables and services in XS advanced.

Usage

The following example shows you how to look up the specific services bound to your application:

Sample Code

```
var xsenv = require('@sap/xsenv');
var services = xsenv.getServices({
  hana: { tag: 'hdb' },
  scheduler: function(service) { return service.label === 'jobs' }
});
var hanaCredentials = services.hana;
var schedulerCredentials = services.scheduler;
```

The search criteria for the required services is specified in the query parameter to `getServices`. Each property of the query object specifies the query value for one service.

Note

`getServices` can return only the requested services; it cannot return any of the standard SAP HANA XS advanced services (for example, `hana`, `uaa`, `jobs`) unless they are explicitly specified in the query parameter.

You can also pass a custom file from which you load a default service configuration. For example, the following example shows how to load defaults from the file `my-services.json`:

Sample Code

```
var xsenv = require('@sap/xsenv');
var services = xsenv.getServices({
  hana: { tag: 'hana' },
  uaa: { tag: 'uaa' }
}, 'my-services.json');
```

Service Queries

Both `getServices` and `filterCFServices` use the same service query values.

Query Values for getServices and filterCFservices

Query Value	Description
`{string}`	Matches the service with the same service instance name (`name` property). Same as `{ name: '<string>' }`.
`{object}`	All properties of the given object should match corresponding service instance properties as they appear in VCAP_SERVICES
`{function}`	A function that takes a service object as argument and returns true only if it is considered a match

If an object is given as a query value, it may have the following properties:

Object Properties for Query Values

Property	Description
`name`	The name of the service instance; the name you use to bind the service
`label`	Service name - the name shown by the command: "`xs marketplace`"
`tag`	Should match any of the service tags
`plan`	The name of the service instance plan specified in the command: "`xs create-service`"

Loading SSL Certificates

If SSL is configured in the XS advanced on-premise run-time environment, it will provide one or more trusted CA certificates that applications can use to make SSL connections. If the certificates are available, the file paths to these certificates are listed in the `<XS_CACERT_PATH>` environment variable separated by a path delimiter, for example, a colon (:) on Linux-based systems and a semi-colon (;) on Windows-based systems.

```
loadCerticates([certPath])
```

The `loadCertificates()` function enables you to load the certificates listed in the given path. If the `([certPath])` argument is not provided, `loadCertificates()` uses the `<XS_CACERT_PATH>` environment variable instead. If neither `([certPath])` nor `<XS_CACERT_PATH>` is defined, the `loadCertificates()` function returns "undefined".

→ Tip

The `loadCertificates()` function is synchronous and returns an array even if only a single certificate is provided.

The following example shows how to load trusted CA certificates so they are used for all subsequent outgoing HTTPS connections:

Sample Code

```
var https = require('https');
var xsenv = require('@sap/xsenv');
https.globalAgent.options.ca = xsenv.loadCertificates();
```

The following example shows how to load SSL certificates for use in SAP HANA:

Sample Code

```
var hdb = require('hdb');
var xsenv = require('@sap/xsenv');
var client = hdb.createClient({
  host : 'hostname',
  port : 30015,
  ca : xsenv.loadCertificates(),
  ...
});
```

@sap/xsjs

The package enables SAP HANA XS Classic applications to run in the compatibility layer in the JavaScript run time for SAP HANA extended application services, advanced model (SAP HANA XS advanced).

Usage

Sample Code

```
'use strict';
var xsenv = require('@sap/xsenv');
var xsjs = require('@sap/xsjs');
var port = process.env.PORT || 3000;
var options = xsenv.getServices({
    uaa: 'xsuaa',
    hana: 'hana-hdi',
    jobs: 'scheduler',
    mail: 'mail',
    secureStore: 'secureStore'
}));
xsjs(options).listen(port);
console.log('Node XS server listening on port %d', port);
```

The starting function takes an object that contains service credentials and application options.

Property Options

Property	Default	Usage
`rootDir`	`lib`	Location of XS JavaScript files
`rootDirs`	-	Same as `rootDir`, but an array of directories can be provided; overrides `rootDir` if both are set
`uaa`	-	UAA configuration necessary to enable authentication by JSON Web Tokens (JWT) and business-user propagation to SAP HANA
`hana`	-	Object containing SAP HANA database connection parameters; used for database connectivity
`secureStore`	-	Object containing SAP HANA database connection parameters; used for secure store connectivity
`jobs`	-	Job scheduler connection parameters used to register jobs during application start up and later for updating job execution status when the job has finished
`mail`	-	Mail options, used by XS JavaScript `$.net.Mail` API
`maxBodySize`	1 MB	Maximum body size accepted by xsjs. The value is passed to the bytes library for parsing. For more information, see **Bytes Utility** in the **Related Links** section below.
`anonymous`	`false`	Enable anonymous access; that is, access without user credentials
`formData`	-	Special restrictions over form-data submitted to server
`destinationProvider`	-	Custom function, synchronous or asynchronous, to be used when `$.net.http.readDestination` is called in XS JavaScript code.
`ca`	Certificates listed in the environment variable `<XS_CACERT_PATH>`	Trusted SSL certificates for any outgoing HTTPS connections, an array of loaded certificates
`compression`	`true`	By default text resources over 1,000 are compressed.
`auditLog`	-	Object containing audit-log credentials.
`context`	`{}`	Extend the default context in `@sap/xsjs` scripts
`libraryCache`	`{}`	Contains the `@sap/xsjs` libraries that should be cached.

Property	Default	Usage
`redirectUrl`	-	If specified, a redirect to this URL is triggered when the root path is requested. **i Note** If `@sap/xsjs` is behind a reverse proxy (for example, an application router), the value of this property should be aligned with the path-rewriting rules that may apply.

SAP HANA XS advanced applications connect to SAP HANA with a fixed technical user provided by means of XS advanced services (environment variables). The actual (business) user of the application is retrieved from the JWT token and propagated to SAP HANA. The connection to the Job scheduler service is performed with a fixed technical user provided by the XS advanced service binding.

hana

The `hana` object has the following properties:

Property Options

Property	Mandatory	Usage
`host`	Yes	SAP HANA database host name
`port`	Yes	SAP HANA database port number
`user`	Yes	Technical user for database connections
`password`	Yes	Password for technical user specified in `user`
`schema`	-	If provided, sets current schema for database connections
`connectWithLoggedUser`	-	Possible values are `true` or `false` (default). If provided the database connection will be done made the SAML assertion contained in the JWT token of the logged user. **⚠ Caution** This option is provided only for the SAP HANA Cockpit transition to SAP HANA XS advanced. In general, this option should be avoided.
`sqlcc`	-	Object containing all SQLCC configurations as properties with name after SQLCC name used in XS JavaScript code
`ca`	-	Trusted SSL certificates intended explicitly for use by SAP HANA connections, or an array of loaded certificates. If not provided, the certificate from the service binding is used. If no certificates are available, the SAP HANA connection cannot be encrypted.

The SQLCC property can be initialized from the bound services as shown in the following example:

Sample Code

```
options.hana.sqlcc = xsenv.getServices({
  'com.sap.my.sqlcc_config': 'SQLCC__NAME',
  'com.sap.my.other_sqlcc_config': 'OTHER_SQLCC_UPS_NAME'
});
```

To use it in your XS JavaScript code:

Sample Code

```
var connection = $.db.getConnection('com.sap.my.sqlcc_config');
```

secureStore

The `secureStore` object has the following properties:

Property Options

Property	Mandatory	Usage
`host`	Yes	SAP HANA database host name
`port`	Yes	SAP HANA database port number
`user`	Yes	Technical user for database connections
`password`	Yes	Password for technical user specified in `user`
`schema`	-	If provided, sets current schema for database connections

formData

The `formData` object has the following properties:

Property Options

Property	Default	Usage
`maxFilesSizeInBytes`	10485760	restricts the total size of all the uploaded files.

mail

The `mail` object has the following properties:

Property Options

Property	Mandatory	Usage
`host`	Yes	SMTP server host name
`port`	Yes	SMTP server port number

Property	Mandatory	Usage
`ignoreTLS`	-	`True` or `false` (default). Defines if a STARTTLS command should be invoked if available by the mail server.
`secure`	-	`True` or `false` (default). Defines if the connection should be over TLS/SSL.
`connectionTimeout`	-	Connection timeout in ms. Default =60000.
`authMethod`	-	Authentication method to use. Options are: "`PLAIN`", "`LOGIN`", or "`CRAM-MD5`"
`auth`	-	Authentication credentials, for example: `{user: 'user', pass: 'pass'}`; default is no authentication.

destinationProvider

If your application requires different mechanisms for destination configuration, for example, dynamic configuration changes or dynamically adding new destinations to your application, you can provide your own function that retrieves these configurations from your storage. Support is provided for both synchronous and asynchronous destination provider function. The number of parameters your function has determines if the function is called synchronously or asynchronously. The signatures for both are as follows:

Sample Code

```
function getDestinationSync(packagename, objectname, dtDescriptor) {
 }
function getDestinationAsync(packagename, objectname, dtDescriptor, callback)
{
}
```

Parameter Options

Parameter	Description
`packagename`	The package of the destination supplied to `$.net.http.readDestination`
`objectname`	The object name of the destination supplied to `$.net.http.readDestination`
`dtDescriptor`	The object containing all properties contained in the corresponding HTTP destination configuration file (`.xshttpdest`), if available; otherwise "undefined"
`callback`	Provided only in the asynchronous case and should be called by your provider function to return the destination or report error

@sap/xsjs-test

This client library provides the unit-test framework for the XS JavaScript compatibility layer (XS run time). To use the `@sap/xsjs-test` tools in your MTA project, you will need to perform the following steps:

- Declare a development dependency to `@sap/xsjs-test` in your XSJS application project (for example, in the `package.json`)
- Your tests are located in test folder (`test/`) parallel to `package.json` and `lib/`
- Configure a `xstest` script in your application's `package.json`
- Run the test tools, for example with the command `npm run xstest`

For the dependency to `@sap/xsjs-test`, you need to verify which version of `@sap/xsjs-test` is released, for example, 3.1.2, and refer to the released version. By establishing this developer dependency, you ensure that `@sap/xsjs-test` is installed only in the (local) development installation; it is not available for use in a productive installation. To avoid the necessity of any test dependencies, you can install the `@sap/xsjs-test` package on your development machine, for example, with the following command:

```
npm install -g @sap/xsjs-test
```

Normally the `xsjs` run-time files are located in the `xsjs/` folder, which means that the following paths are expected:

- `<>/xsjs/package.json`
- `<>/xsjs/lib/`
- `<>/xsjs/test/`
  Contains your test scripts; if you choose to put the test scripts in a different folder, a special configuration is required.

Test script

There is a standard binary script defined in the `bin/` folder. Typically, you define a script in the application `package.json`, as illustrated in the following example:

Sample Code

```
"scripts": {
  "xstest" : "xstest"
}
```

Usage

To start the XSJS unit tests from the command line, use the following command:

Sample Code

```
npm run xstest
```

@sap/xssec

The client security library includes the XS advanced container security API for Node.js.

Authentication for node applications in XS advanced relies on a special usage of the OAuth 2.0 protocol, which is based on central authentication at the SAP HANA User Authentication & Authorization (UAA) server that then vouches for the authenticated user's identity by means of a so-called OAuth Access Token. The current implementation uses as access token a JSON web token (JWT), which is a signed text-based token formatted according to the JSON syntax.

In a typical deployment scenario, your node application will consist of several parts, which appear as separate application packages in your manifest file, for example:

- Database artifacts
  This application module (`<myAppName>/db/`) describes and defines the HANA database content
- Application logic
  This application module (`<myAppName>/js/`) contains the application logic: code written in Node.js. This module can make use of this *XS Advanced Container Security API* for Node.js).
- UI client
  This application module (`<myAppName>/web/`) is responsible for the UI layer; this module can make use of the application router functionality (defined in the file `xs-app.json`).

> **i Note**
>
> The application logic written in Node.js and the application router should be bound to one and the same UAA service instance so that these two parts use the same OAuth client credentials.

To use the capabilities of the XS Advanced container security API, add the package "`@sap/xssec`" to the dependencies section of your application's `package.json` file. To enable tracing, set the environment variable `DEBUG` as follows: `DEBUG=xssec:*`.

Usage

In order to authenticate requests that reach the Node.js back end, `sap-xssec` offers the following mechanisms:

- The XS Advanced Container Security API
- The "passport" strategy included in `@sap/xssec`

To use the XS advanced container security API, it is necessary to provide a JWT token. The examples below rely on users and credentials that you should substitute with the ones in your context. The typical use case for calling this API lies from within a container when an HTTP request is received. The authorization header (with keyword "`bearer`") already contains an access token; you can remove the prefix bearer and pass the remaining string to the API as "`access_token`", as illustrated in the following example:

> **‹≡› Sample Code**

```
xssec.createSecurityContext(access_token, xsenv.getServices({ uaa:
'uaa' }).uaa, function(error, securityContext) {
  if (error) {
      console.log('Security Context creation failed');
      return;
  }
  console.log('Security Context created successfully');
});
```

The example above uses the Node.js package `xsenv` to retrieve the configuration of the default services; the default services are read either from the environment variable `<VCAP_SERVICES>` or, if not set, from the

default configuration file. However, only the required User Authentication & Authorization configuration (`uaa`) is passed to the method `createSecurityContext`.

The creation function `xssec.createSecurityContext` is used for an end-user token (for example, with "grant type" `password` or `authorization_code`) where user information is expected to be available within the token and, as a consequence, within the security context. `createSecurityContext` also accepts a token of grant type `client_credentials`, which enables the creation of a limited security context where certain functions are not available, as described in more detail in the Security API documentation below.

If you use the Node.js packages `express` and `passport`, you can plug in a ready-made authentication strategy, as illustrated in the following example:

Sample Code

```
var express = require('express');
var passport = require('passport');
var JWTStrategy = require('@sap/xssec').JWTStrategy;
var xsenv = require('@sap/xsenv');
...
var app = express();
...
passport.use(new JWTStrategy(xsenv.getServices({uaa:{tag:'xsuaa'}}).uaa));
app.use(passport.initialize());
app.use(passport.authenticate('JWT', { session: false }));
```

Tip

It is recommended to disable the session as in the example above. In XS advanced, each request comes with a JWT token so it is authenticated explicitly and identifies the user. If you still need the session, you can enable it but then you should also implement user serialization-deserialization and some sort of session persistency.

Offline Validation

The trust relationship required for offline validation is created by binding the UAA service instance to your application. The key used to validate tokens is included in the credentials section of the environment variable `<VCAP_SERVICES>`. By default, the offline validation check in Cloud environments only accepts tokens intended for the same OAuth 2.0 client in the same UAA subaccount. However, if an application needs to consume tokens that were issued either for different OAuth 2.0 clients or for different subaccounts, it is possible to specify an access control list (ACL) entry in the environment variable `<SAP_JWT_TRUST_ACL>`, as illustrated in the following example:

! Restriction

Subaccounts are not used in on-premise XS advanced systems.

Sample Code

Client ACL in `<SAP_JWT_TRUST_ACL>` Environment Variable

```
SAP_JWT_TRUST_ACL:
[ {"clientid":"<OAuth_2.0_client_ID>","subaccount":"<subaccount>"},...]
```

The name of the OAuth 2.0 client has the prefix "`sb-`"; the corresponding content is a JSON string that contains an array of subaccounts and OAuth 2.0 clients. To establish trust with **any** OAuth 2.0 client and/or

subaccounts, you can use an asterisk (*). For security reasons, however, this is not recommended in a productive environment. The value for the subaccount is "`uaa`".

> **i Note**
>
> Changes have been made to the container security API for Node.js >=3.0.6. If you plan to use the security API, please note that it is no longer possible to use the parameter `SAP_JWT_TRUST_ACL` to specify a dedicated access control list (ACL) for JWT tokens. Changes also apply regarding the granting of security scopes, which are defined and granted in the application security descriptor (`xs-security.json`). For example, if a business application A wants to call another application B, it is now mandatory that application B grants at least one scope to the calling business application A. For more information about security scopes, see the *Application Security Descriptor Configuration Syntax* in *Related Information* below.

Container Security API

The complete details of the XS advanced Container Security API including all parameters and options are available in the README file included in the `@sap/xssec` package.

Container Security API

API	Description
`createSecurityContext`	Creates the "security context" by validating the received access token against credentials put into the application's environment via the UAA service binding
`getLogonName`	Returns the user's logon name *
`getGivenName`	Returns the user's given (first) name *
`getFamilyName`	Returns the user's family name *
`getEmail`	Returns the user's email address *
`checkLocalScope`	Checks a scope that is published by the current application in the `xs-security.json` file. Returns "true" if the checked scope is included in the user's authorization scopes (otherwise "false")
`checkScope`	Checks a scope that is published by an application. Returns "true" if the checked scope is included in the user's authorization scopes (otherwise "false")
`getToken`	Returns a token that can be used to connect to the SAP HANA database. If the token that the security context has been instantiated with is a foreign token (meaning that the OAuth client contained in the token and the OAuth client of the current application do not match), "`null`" is returned instead of a token.
`getHdbToken`	Returns a token that can be used to connect to the SAP HANA database. If the token that the security context has been instantiated with is a "foreign" token (the OAuth client contained in the token and the OAuth client of the current application do not match), "`null`" is returned instead of a token.

API	Description
`requestTokenForClient`	Requests a token with `grant_type=user_token` from another client. The requesting client must also have `grant_type=user_token`, and the current user token must include the authorization scope `uaa.user`.
`hasAttributes`	Returns "`true`" if the token contains any XS advanced user attributes; otherwise "`false`".
`getAttribute`	Returns the attribute exactly as it is contained in the access token. If no attribute with the given name is contained in the access token, "`null`" is returned. If the token that the security context has been instantiated with is a foreign token (the OAuth client contained in the token and the OAuth client of the current application do not match), "`null`" is returned regardless of whether the requested attribute is contained in the token or not.
`getAdditionalAuthAttribute`	Requires the `name` of the additional authentication attribute requested. Returns the requested attribute exactly as it is contained in the access token. If no attribute with the given `name` is contained in the access token, `null` is returned. **i Note** Additional authentication attributes are also returned in "foreign" mode (in contrast to `getAttribute`).
`isInForeignMode`	Returns "`true`" if the token, that the security context has been instantiated with, is a foreign token that was not originally issued for the current application, otherwise "`false`".
`getIdentityZone`	Returns the identity zone that the access token has been issued for
`getExpirationDate`	Returns the expiration date of the access token as a Javascript "Date" object
`getCloneServiceInstanceId`	Returns the id of the cloned service instance, if the XSUAA broker plan is used
`getGrantType`	Returns the grant type of the JWT token, for example: `authorization_code`, `password`, `client_credentials`, etc.

i Note

* Not available for tokens of grant type `client_credentials`

@sap/xss-secure

This is a Node.js package that includes the cross-site scripting (XSS) security implementation taken from SAP UI5.

Usage

Sample Code

```
var xsssecure = require('@sap/xss-secure');
```

The `@sap/xss-secure` package includes the following APIs:

- `encodeCSS`
  For Cascading Style sheets
- `encodeHTML`
  For HTML markup
- `encodeJS`
  For JavaScript code
- `encodeURL`
  For URL components, for example, parameter names or values
- `encodeXML`
  For XML code

Cross-site Scripting (XSS) is the name of a class of security vulnerabilities that can occur in Web applications. It summarizes all vulnerabilities that allow an attacker to inject HTML Markup or JavaScript into the affected Web application's front-end. XSS can occur whenever the application dynamically creates its HTML, JavaScript, or CSS (either in the back end or in the front-end part of the application in the user's Web browser), and values controlled by the attacker are used in this process. If this happens, the values are included in the generated HTML, JavaScript, or CSS without proper validation or encoding. As a result, an attacker is able to include arbitrary HTML, JavaScript, or CSS into an application's client front end; this code is, in turn, rendered by the victim's Web browser and, in this way, interpreted in the victim's current authentication context.

@sap/e2e-trace

SAP Passports allow the identification of a specific request in an end-to-end scenario involving several components, which need to communicate with each other. The client sends a special header containing the SAP Passport (a hex string with a special structure) to the first component. The client can send the SAP passport by means of a browser plug in or the front-end component of an SAPUI5 application. Whenever an application component receives an SAP Passport, this component should ensure that the unique identifiers of the SAP Passport are included in its log and trace files, update the SAP passport with component-specific data, and then forward it to the next system.

An application receives an SAP Passport in the `@sap/passport` header. The same header is used when the SAP Passport is forwarded to another system with the HTTP protocol. If the SAP passport is sent to SAP HANA, the SAP Passport should be set as the `'SAP_PASSPORT'` session variable of the database connection.

To load the `@sap/e2e-trace` package, use the following command:

Sample Code

```
var SAPPassport = require('@sap/e2e-trace').Passport;
```

The code in the following example shows how to create an instance of an SAP Passport:

Sample Code

```
function requestHandler(req, res) {
  var encodedPassport = req.headers[SAPPassport.HEADER_NAME];
  if (encodedPassport) {
    var passport = new SAPPassport(encodedPassport);
  }
}
```

The library provides the constant `SAPPassport.HEADER_NAME` for the `'@sap/passport'` header; the passport variable is an instance which you can use to read (or modify) the SAP Passport in your application component. The following code example shows how to read the unique identifiers of an SAP Passport:

Sample Code

```
var identifiers = passport.readUniqueIdentifiers();
```

The returned value (assigned to the identifiers variable) is an object that has the following properties: `transactionID`, `rootContextID`, `connectionID`, and `connectionCounter`. These fields of the SAP Passport must be present in the logs and traces of the application component. If you are using the `@sap/logging` library, refer to its documentation to check if the specific version of the library is capable of handling SAP Passports.

To see how to update the SAP Passport before forwarding it to the next application component, have a look at the following code example:

Sample Code

```
passport.update({
  previousComponent: 'my-application',
  connectionID: '00112233445566778899AABBCCDDEEFF',
  connectionCounter: 36
});
```

This method takes an object with the following **mandatory** properties:

Mandatory SAP Passport Properties

Property	Value	Semantics
`previousComponent`	ASCII string (up to 32 characters)	The name of the current component which is about to make an outbound connection to another component.
`connectionID`	GUID (16-byte hex)	Every connection between the current component and the next component should have an ID which is passed as that property.
`connectionCounter`	Positive integer (up to 4 bytes)	The amount of time for the connection with the given `connectionID` is being reused.

! Restriction

The SAP HANA database has a limitation regarding the size of the SAP Passport. It is recommended to call the method `passport.compact();` before forwarding an SAP Passport to SAP HANA.

The following code example shows how to generate a hex string from the updated SAP Passport; the SAP Passport can be sent to other components in this format:

‹≡› Sample Code

```
var http = require('http');
var url = require('url');
var encodedPassport = passport.serialize();
var options = url.parse('http://my-host:1234/my/path');
options.headers = {};
options.headers[SAPPassport.HEADER_NAME] = encodedPassport;
var request = http.request(options);
request.on('error', function (err) { /* ... */ });
request.on('response', function (response) { /* ... */ });
request.end();
```

@sap/node-vsi

The `@sap/node-vsi` package contains the Virus Scan Interface (VSI) binding for Node.js. The package also includes the native libraries required to run on Windows, Linux, and MacOS operating systems.

The following example shows how to use `@sap/node-vsi` to scan for the standard test file from the European Institute for Computer Antivirus Research at “www.eicar.com”. All anti-virus scanners must be able not only to find the file but also detect that it is a virus:

‹≡› Sample Code

```
var vsi = require('@sap/node-vsi');
var vsiProfile = vsi.vsiProfile;
var v = new vsiProfile("");
v.scanBytes("eicar.txt","X5O!P%@AP[4\\PZX54(P^)7CC)7}$EICAR-STANDARD-
ANTIVIRUS-TEST-FILE!$H+H*", 68);
console.log("\nResult of eicar scan is rc " + v.getLastErrorCode() + " (" +
v.getScanErrorName() + ") with error message: \n" + v.getLastError() + "\n" );
```

Getting Started

To start using the VSI `@sap/node-vsi` run the following command from your project directory:

```
$ var vsi = require('@sap/node-vsi');
```

@sap/sds-deploy

The `@sap/sds-deploy` package includes the Node.js client for the deployment of application projects for the SAP HANA Streaming Analytics option. The `@sap/sds-deploy` library compiles the Continuous Computation

Language (CCL) content of a Streaming Analytics project (consisting of a CCL file and an optional CCR file) and deploys the Streaming Analytics project to the Streaming Analytics service installed with the SAP HANA platform.

The Streaming Analytics application project is mapped to a Streaming Analytics module in the multi-target application (MTA). The Streaming Analytics application project consists of two configuration files:

- `mySDS_Project-Definition.ccl`
  The Streaming Analytics project-definition file is written in Continuous Computation Language (CCL), which is the primary event-processing language of SAP HANA Streaming Analytics. You define a Streaming Analytics project using CCL, which is a language that is based on Structured Query Language (SQL), and adapted for stream processing.
- `mySDS_Project_Configuration.ccr`
  A project-configuration file (`.ccr`) is an XML document that is used to control specific run-time properties of a Streaming Analytics project, including the data stream's URI bindings as well as any adapter properties, parameter values, and advanced deployment options.

The following example of a simple `.ccl` file shows some code that runs a simple Tuple test:

Sample Code

Example `random_tuple_test2.ccl`

```
CREATE INPUT WINDOW InputStream1 SCHEMA ( Column1 integer ,
Column2 integer , Column3 integer , Column4 string ) PRIMARY
KEY(Column1,Column2,Column3,Column4) KEEP 10000 ROWS;
ATTACH INPUT ADAPTER RandomTuples2 TYPE randomtuplegen_in TO InputStream1
PROPERTIES Rate =10000 , RowCount =0 , SecondDateFormat = '%Y-%m-%d
%H:%M:%S' , MsDateFormat = '%Y-%m-%d %H:%M:%S' ;
CREATE OUTPUT STREAM OutStream1 AS SELECT InputStream1.Column1
SimpleResultExpression2 , InputStream1.Column2 SimpleResultExpression1 FROM
InputStream1;
CREATE OUTPUT STREAM OutStream2 AS SELECT InputStream1.Column1
SimpleResultExpression2 , InputStream1.Column2 SimpleResultExpression1 FROM
InputStream1;
```

The following code example shows the contents of the corresponding `.ccr` file for the `random_tuple_test2.ccl`:

Sample Code

Example `random_tuple_test2.ccl`

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://www.sybase.com/esp/project_config/2010/08/">
 <Deployment>
    <Project ha="false">
      <Options>
        <Option name="time-granularity" value="5"/>
        <Option name="debug-level" value="7"/>
        <Option name="ignore-config-topology" value="false"/>
      </Options>
      <Instances>
        <Instance>
          <Failover enable="false"/>
          <Affinities/>
        </Instance>
      </Instances>
    </Project>
  </Deployment>
```

```
</Configuration>
```

@sap/audit-logging

Provides audit logging functionality for Node.js applications.

Audit logging concerns writing entries in a specific format to a log storage. The entries written to the audit log concern any events of significant importance, for example, security events which may impact the confidentiality, the integrity, or the availability of a system. Access to sensitive personal data (both reading and altering) such as bank accounts, political opinion, health status is also an event that is written to the audit log. Audit logs are not the same as ordinary log files. Ordinary log files are used by the system administrator who need to keep track of the state of a system; audit logs are read by an auditor. Audit logs are subject to legal requirements, which in some countries are stricter than in others. In general, the events that are supposed to be recorded in the audit log can be grouped into the following categories:

- Changes to system configurations (which may have significant effect on the system itself)
- Access to sensitive data (related to data privacy)
- General security events (for example, starting or stopping a system, failed authorization checks, etc.)

→ Tip

For more detailed information about the audit-log NPM package, see the `README.md` file included in the `@sap/audit-logging` package.

@sap/ui-annotations

Enables the use of UI annotations in CDS; currently, this feature is supported by the XS advanced Java framework in combination with OData v4. The following UI annotations are currently supported:

- `Hidden`
- `Identification`
- `LineItem`
- `Mandatory`
- `Masked`
- `MultiLineText`
- `Optional`
- `ReadOnly`
- `StatusInfo`
- `Chart`
- `Contact`
- `FieldGroup`
- `DataPoint`
- `SelectionFields`

- `HeaderInfo`

To enable UI annotations, add the entry "`using sap.common::UI;`" to the start of the CDS documents which contain elements that you want to publish with OData v4.

Sample Code

Enable UI Annotations in an OData v4 Service

```
namespace acme.com;
using sap.common::UI;
@OData.publish : true
context ContextA {
  MyEntity1 {
    key ID : Integer;
  };
  MyEntity2 {
    key ID : Integer;
    type Address1
  };
};
```

To use the features provided in the XS advanced NPM package `@sap/ui-annotations`, it is necessary to ensure that the `package.json` file in the XS advanced application's HDB module specifies the dependency to "`@sap/ui-annotations`", as illustrated in the following example:

Sample Code

Using UI Annotations

```
{
 "name": "deploy",
 "dependencies":
  {
   "@sap/hdi-deploy": "2.2.0"
   "@sap/ui-annotations": "1.0.0-1"
  },
 "scripts":
   {
    "start": "node node_modules/@sap/hdi-deploy/"
   }
}
```

@sap/xs-msg

Provides messaging capabilities with a message broker. This package supports the RabbitMQ message broker with the following protocols:

- amqp v091
- amqp v100
- mqtt v311

Prerequisites

A message broker must be available, for example, RabbitMQ.

Usage

The package folder `./examples` includes some test programs that can be used if a broker can be reached locally at the protocol-specific default port. The `examples/cfg` folder also includes some sample configurations.

The following example shows how to create a new messaging client and set it up to start consuming incoming messages:

Sample Code

```
const msg = require('@sap/xb-msg');
const env = require('@sap/xb-msg-env');
/* get options from cf/xsa environment */
const options = env.msgClientOptions('msg-instance-01', ['MyInpA'], []);
/* start messaging */
const client = new msg.Client(options);
client.istream('MyInpA')
   .on('subscribed', () => {
       console.log('subscribed');
   })
   .on('data', (message) => {
       console.log('message: ' + message.payload.toString());
       message.done();
   });
client.connect();
```

The following example shows how to create a new messaging client and set it up to start producing messages:

Sample Code

```
 const msg = require('@sap/xb-msg');
const env = require('@sap/xb-msg-env');
/* get options from cf/xsa environment */
const options = env.msgClientOptions('msg-instance-01', [], ['MyOutB']);
/* start messaging */
const client = new msg.Client(options);
client.ostream('MyOutA')
    .on('ready', () => {
        send();
    })
    .on('drain', () => {
        send();
    })
    .on('error', (error) => {
        console.log(error);
    });
client.connect();
```

After the client is created and connected the application can start writing to (and/or reading from) the named stream instances. Based on the client options, one or more connections are created for each of the supported protocols and each providing multiple inbound or outbound streams. All stream IDs must be unique in the scope of a client instance.

Note

For more information about features and functionality, see the `README.md` file included in the Node.js package `@sap/xb-msg`.

Connections via `'net'` and `'tls'` are supported for all protocols. For MQTT, WebSocket can be used either with `'https'` (wss) or `'http'` (ws). If multiple settings are provided, the order of preference is as follows: `'tls'`, `'net'`, `'wss'`, and `'ws'`.

@sap/xb-msg-env

This package provides the functions needed to set up messaging client options from Cloud Foundry or XS advanced environment variables.

The following clients are supported:

- `@sap/xb-msg`
  A protocol-agnostic API with multiple destinations per single client
- `@sap/xb-msg-amqp-v100`
  A protocol-specific API with a single connection per client
- `@sap/xb-msg-amqp-v091`
  A protocol-specific API with a single connection per client
- `@sap/xb-msg-mqtt-v311`
  A protocol-specific API with a single connection per client.

`@sap/xb-msg-env` can use the following environment variables:

Environment Variable	Description
`<VCAP_SERVICES>`	Defines bindings to RabbitMQ or Enterprise Messaging services
`<SAP_XBEM_SERVICE_LABEL>`	Defines details of an alternative service label for Enterprise Messaging
`<SAP_XBEM_BINDINGS>`	Defines details of incoming and (or) outgoing message streams (`"istreams"` and `"ostreams"`)

Prerequisites

See the `README` file in the NPM package for information about the versions of the various messaging packages required, for example, `@sap/xb-msg` version "0.2.3".

Usage

The following example shows how to use environment variables for one Rabbit MQ instance named '`myService`'. The messaging service's incoming and outgoing message streams can be maintained as `"inputs"` and `"outputs"` in another environment variable named `SAP_XBEM_BINDINGS`. In the following example, one output named '`myOutA`' is defined.

Sample Code

```
{
  "VCAP_SERVICES": {
    "rabbitmq": [
        {
            "credentials": {
                "hostname": "10.11.11.11",
                "ports": {
```

```
                    "15672/tcp": "8888",
                    "5672/tcp": "9999"
                },
                "port": "9999",
                "username": "user",
                "password": "password",
                "uri": "amqp://user:password@10.11.11.11:9999"
            },
            "syslog_drain_url": null,
            "volume_mounts": [],
            "label": "rabbitmq",
            "provider": null,
            "plan": "v3.6-container",
            "name": "myService",
            "tags": [
                "rabbitmq",
                "mbus",
                "pubsub",
                "amqp"
            ]
        }
    ]
  },
  "SAP_XBEM_BINDINGS": {
    "outputs": {
      "myOutA" : {
        "service": "myService",
        "address": "topic:Cars/Velocity/milesPerHour",
        "reliable": false
      }
    }
  }
}
```

The following call can be used to provide the client options for `@sap/xb-msg`:

```
const env = require('@sap/xb-msg-env');
const opt = env.msgClientOptions('myService', [], ['myOutA']);
```

The generated options for a `@sap/xb-msg` client using the `amqp-v091` protocol should look like the following example:

Sample Code

Messaging Client Options for `@sap/xb-msg`

```
{ "destinations": [
    {
      "name": "myService",
      "type": "amqp-v091",
      "net": {
        "host": "10.11.11.11",
        "port": 9999
      },
      "sasl": {
        "user": "user",
        "password": "password"
      },
      "amqp": {
        "vhost": "/"
      },
      "istreams": {
      },
      "ostreams": {
        "out": {
```

```
            "channel": 1,
            "exchange": "amq.topic",
            "routingKey": "Cars.Velocity.milesPerHour",
            "confirms": false
          }
        }
      }
    ]
  }
```

@sap/xb-msg-amqp-v091

This package provides a client implementation for AMQP v0.9.1.

A single client instance represents one connection to the broker. Either TLS or NET socket is used depending on the options defined for the client . The API works completely asynchronous based on callbacks, often providing also “done” (resolve) and “failed” (reject) callbacks. This means it would be simple to use “Promise” objects in the application even if the client library so far does not use it. AMQP v0.9.1 defines classes and methods (like remote procedure calls). Unfortunately, some of them do not allow a key-based mapping of responses to requests. Hence, for those methods the client has to wait for the response before a second request can be sent. The client encapsulates this and behaves always asynchronously for the caller.

Prerequisites

A message broker must be available, for example, RabbitMQ.

Usage

The package folder `./examples` includes some test programs that demonstrate the following usage scenarios:

- How to use the `publisher.js` and `subscriber.js` APIs
- How to use the unified streams `producer.js` and `consumer.js`

All examples support the use of custom settings, for example, to use a remote host or to try different stream settings. Settings can be provided in a JavaScript (`.js`) file that is included as a command-line parameter for the `node` command. The file exports a client-option object. Defaults will still be used for undefined fields.

> **i** Note
>
> For more information about features and functionality, see the `README.md` file included in the Node.js package `@sap/xb-msg-amqp-v091`.

The following example shows how to create a new client instance:

> Sample Code

```
const options = {
    tls: {
        host: 'localhost',
        port: 5671,
        ca: [
            fs.readFileSync('../../../truststore/cacert.pem'),
            fs.readFileSync('../../../truststore/cert.pem')
        ]
```

```
    },
    net: {
        host: 'localhost',
        port: 5672,
    },
    sasl: {
        user: 'guest',
        password: 'guest'
    },
    amqp: {
        vhost: '/',
    }
};
const client = new AMQP.Client(options);
```

→ Tip

It is also possible to provide connection data as a URI or an array of URIs.

In the following example, the client starts by using the first URI in the array to establish a connection. If the connection attempt is not successful, the client tries further URIs automatically in the sequence provided until a connection can be established. If the client fails with all provided URIs, it stops and waits for another explicit call to connect. At this point an event 'disconnected' is raised.

‹≡› Sample Code

Client Connection Details as URLs

```
const options = {
    uri: [
         'amqp://guest:guest@localhost:5672/vh111',
         'amqp://guest:guest@localhost:5672/vh222'
    ]
};
const client = new AMQP.Client(options);
```

xb-msg-amqp-v100

This package provides a messaging client as well as the classes required to create a server for AMQP 1.0.

Prerequisites

A message broker must be available, for example, RabbitMQ, and the AMQP 1.0 plug-in.

Usage

A single client instance represents one connection to the broker. Either TLS or NET socket is used depending on defined client options. In addition to plain TCP/IP, WebSocket is also supported, with and without OAuth 2.0, grant flows `ClientCredentialsFlow`, and `ResourceOwnerPasswordCredentialsFlow`.

The API works asynchronously based on callbacks, typically providing done (`resolve`) and failed (`reject`) callbacks. This means it is easy to use Promise objects in the application even if this library does not currently use them.

The package folder `./examples` includes some test programs that demonstrate the following usage scenarios:

- How to use the client as a “producer”, a “consumer”, or a “counter”
- How to configure a server with basic settings for a protocol gateway

> **Note**
>
> For more information about features and functionality, see the `README.md` file included in the Node.js package `@sap/xb-msg-amqp-v100`.

All client examples run with the provided default settings, for example, with RabbitMQ installed at `localhost:5672` with user “guest/guest”, and with the AMQP 1.0 plug-in enabled. Alternatively, the producer can run in combination with the “gateway” example. All examples can be configured with custom settings, for example, to use a remote host or to try different stream settings. The modified settings can be included in a JavaScript file, for example, `my-options.js`, that is provided as a command-line parameter, as illustrated in the following example, where the settings file is located in the `config\` directory used for all the examples:

```
node .\examples\producer.js ..\config\my-options.js
```

The following example can be used to start testing:

> **Note**
>
> The data section is only for use with the example programs; it is ignored by the client.

> **Sample Code**

```
'use strict';
module.exports = {
    net: {
        host : '127.0.0.1',
        port : 5672
    },
    sasl: {
        mechanism : 'PLAIN',
        user : 'guest',
        password : 'guest'
    },
    data: {
        source : 'q001', // a queue name, source address for a receiver
        target : 'q002', // a queue name, target address for a sender
        payload : new Buffer.allocUnsafe(50).fill('X'),
        maxCount : 10000,
        logCount : 1000
    }
};
```

@sap/xb-msg-mqtt-v311

This package provides a client implementation for MQTT v3.1.1.

A single client instance represents one connection to the broker, and the connection uses either TLS or NET socket depending on the options defined in the client connection. Since the API works asynchronously based

on callbacks, often providing “done” (resolve) and “failed” (reject) callbacks, it should be simple to use Promise objects in the application even if the client library does not yet do so.

Prerequisites

A message broker must be available, for example, RabbitMQ, with the MQTT plug-in enabled.

Usage

The package folder `./examples` includes some test programs that demonstrate the following usage scenarios:

- How to use the `publisher.js` and `subscriber.js` APIs
- How to use the unified streams `producer.js` and `consumer.js`

All examples support the use of custom settings, for example, to use a remote host or to try different stream settings. Settings can be provided in a JavaScript (`.js`) file provided as command-line parameter to the `node` command. The file exports a client-option object. Defaults will still be used for undefined fields.

> **Note**
>
> For more information about features and functionality, see the `README.md` file included in the Node.js package `@sap/xb-msg-mqtt-v311`.

The following example shows how to create a new client instance using plain TCP:

Sample Code

```
const options = {
   net: {
       host: 'localhost',
       port: 1883
   },
   credentials: {
       user: '',
       password: ''
   }
};
const client = new MQTT.Client(options);
```

The following example shows how to create a new client instance using plain TCP with TLS connection:

Sample Code

```
const options = {
   net: {
       host: 'localhost',
       port: 1883
       ca: [
           fs.readFileSync('../../../truststore/cacert.pem'),
           fs.readFileSync('../../../truststore/cert.pem')
       ]
   },
   credentials: {
       user: '',
       password: ''
   },
   mqtt: {
       clientID : '',
       cleanSession : true,
```

```
        keepAlive : 30
    }
};
const client = new MQTT.Client(options);
```

@sap/site-entry

This component is the Web entry for Fiori Launchpad portal sites. It includes an `approuter` component that serves as its Web server for requests from client-side resources and back-end calls. This component uses the portal-service that exists in the XS advanced environment.

Usage

Create the folder structure shown in the following example:

Sample Code

Application Sources Folder Structure

```
<site-entry-folder-name>/
| - package.json
| - xs-app.json
```

The following code sample shows the contents of the application package descriptor (`package.json`):

Sample Code

`package.json`

```
{
  "engines": {
    "node": ">=4.0.0"
  },
  "name": "site-entry",
  "dependencies": {
    "@sap/site-entry": "^1.9.1"
  },
  "scripts": {
    "start": "node --harmony node_modules/@sap/site-entry/server.js"
  }
}
```

In the application descriptor `xs-app.json`, add the route to your SAPUI5 application's static resources:

Sample Code

Application Descriptor (`xs-app.json`)

```
{
"source": "^/app1/sap/demo/(.*)",
"target": "$1",
"localDir": "app1/sap/demo/",
"cacheControl": "public, max-age=31536000,must-revalidate"
}
```

The `mtad.yaml` file must contain the following information:

- The site-entry module needs to be bound to the `portal-service` service; it should require a resource of the following type: `com.sap.portal.site-host`.
- The path to the SAPUI5 that is deployed on the XS advanced model host
- The path to the XS User Account and Authentication (UAA) system

Sample Code

Application Deployment Descriptor (`mtad.yaml`)

```
…
modules:
 - name: site-entry-<unique site id>
   parameters:
      memory: 64M
   requires:
    - name: sap-portal-services-host-<unique site id>
    - name: myuaa
    - name: sapui5-provider
      properties:
         sapui5url: ~{url}
   type: javascript.nodejs
 - name: site-content-<unique site id>
   parameters:
      memory: 32M
   requires:
    - name: sap-portal-services-client-<unique site id>
    - name: myuaa
   type: com.sap.portal.site-content
resources:
 - name: sap-portal-services-host-<unique site id>
   parameters:
      config:
         siteId: <unique site id>
   type: com.sap.portal.site-host
 - name: myuaa
   parameters:
      service-plan: space
   type: com.sap.xs.uaa
 - name: sapui5-provider
   parameters:
      provider-id: com.sap.ui5.dist.sapui5-dist-
xsa.XSAC_UI5_FESV3:sapui5_fesv3
      version: '>=1.42.0'
      provider-nid: mta
   type: configuration
 - name: sap-portal-services-client-<unique site id>
   parameters:
      config:
         siteId: <unique site id>
   type: com.sap.portal.site-content
```

@sap/site-content-deployer

This component is combined with the `@sap/site-entry` node module, which in turn acts as the Web-entry server for the Fiori Launchpad portal site. This component is used to deploy the Fiori Launchpad portal site configuration (for example, the configuration of tiles, groups and catalogs) into the XS advanced environment. This component uses the portal-service that exists in the XS advanced environment.

Usage

Create the folder structure shown in the following code sample and copy the `manifest.json` file of your SAPUI5 application to your app folder, `MySAPUI5App`, as illustrated in the following example:

Sample Code

Application Sources Folder Structure

```
<site-content-deployer-folder-name>/
|- package.json
|- applications/
|   \- MySAPUI5App/
|      |- manifest.json
|      \- i18n/
\- src/
   |- site-content.json
   \- i18n/
```

The following code is an example of a `package.json` file:

Sample Code

Application Package Descriptor (`package.json`)

```
{
  "engines": {
    "node": "4.x"
  },
  "name": "site-content-deployer",
  "version": "1.9.1",
  "description": "portal deployer package",
  "dependencies": {
    "@sap/site-content-deployer": "^1.9.1"
  },
  "scripts": {
    "start": "node --harmony node_modules/@sap/site-content-deployer/
deploy.js"
  }
}
```

The `site-content-deployer` module must be bound to the `portal-service` service. For example, in the `mtad.yaml` file, the `site-content-deployer` should require a resource of type "`com.sap.portal.site-content`", as illustrated in the following example:

Sample Code

Application Deployment Descriptor (`mtad.yaml`)

```
…
modules:
- name: site-content-deployer-<unique site id>
   type: javascript.nodejs
   requires:
    - name: sap-portal-services-client-<unique site id>
…
resources:
- name: sap-portal-services-client-<unique site id>
   type: com.sap.portal.site-content
  parameters:
      config:
```

```
          siteId: <unique site id>
```

For more information about developing Fiori Launchpad modules, see *Related Information*.

@sap/site-app-server

This component is a Web server for static resources used by applications that run in a Fiori Launchpad portal site. This component is coupled with the `@sap/site-entry` node module.

Related Information

Download and Consume SAP Node.js Packages [page 633]
The NPM Registry [page 687]
Configuring the HDI Deployer [page 253]
Maintaining XS Advanced Application Routes and Destinations [page 912]
Developing SAP Fiori Launchpad Modules [page 1223]
@sap packages README files on nodejs.com
Bytes Utility
Application Security Descriptor Configuration Syntax [page 999]

7.1.2.2 The NPM Registry

The public NPM registry includes SAP Node.js modules for use by application developers.

Whether you are using command-line tools or graphical tools to build your Node.js application, you can configure the build process to use the packages that are available in registries which make Node Package Modules (including those developed and maintained by SAP) available for public consumption, for example, `"registry.npmjs.org"`.

- NPM Registry Configuration for Command-Line and Build Tools [page 688]
- NPM Registry Configuration for SAP Web IDE for SAP HANA [page 688]

> **i** Note
>
> The SAP NPM registry at `https://npm.sap.com` has been retired and its contents have been migrated to the public NPM registry at `https://registry.npmjs.org`. In future, updates to SAP NPM packages will only be available on `https://registry.npmjs.org`.

NPM Registry Configuration for Command-Line and Build Tools

If you are using command-line tools to build your Node.js applications for XS advanced, you can configure the tools to use a particular NPM registry to resolve the dependencies between resources specified in the deployment descriptors. To specify the NPM registry, add the `"@sap:registry="` property to the `.npmrc` resource-configuration file, as illustrated in the following example:

> ### Sample Code
>
> NPM Registry Configuration in the `.npmrc` File
>
> ```
> ...
> @sap:registry=https://registry.npmjs.org
> ...
> ```

You can also use the `npm` command to configure the target registry for the current session:

```
$ npm config set @sap:registry https://registry.npmjs.org
$ npm install @sap/<node_package>
```

> ### Note
>
> The public NPM registry is now enabled by default for the resolution of Node.js module dependencies in the `@sap` scope during the staging of Node.js applications in the Cloud Foundry environment on SAP Cloud Platform.

If you publish Node.js applications to the SAP Cloud Platform's Cloud Foundry environment without dependencies, it is no longer necessary to specify the NPM registry by means of an environment variable or in the NPM configuration file (`.npmrc`). If, however, you want to run or test a Node.js application locally, you still need to configure the NPM registry as described above.

NPM Registry Configuration for SAP Web IDE for SAP HANA Build Tools

If you are using the build tools included in SAP Web IDE for SAP HANA to build your Node.js applications for XS advanced, you can configure SAP Web IDE for SAP HANA to use one or more specific NPM registries to resolve the dependencies between the resources specified in the development and deployment descriptors.

To specify a specific NPM registry that will be used for the resolution of dependency requirements when developing and building applications with SAP Web IDE for SAP HANA, add the `"UPSTREAM_LINK"` property to the `di-local-npm-registry` module in the MTA **extension** descriptor (for example, `sapwebide.mtaext`) that is used when deploying SAP Web IDE, as illustrated in the following example:

> ### Note
>
> If a suitable MTA extension descriptor does not already exist for the deployment of SAP Web IDE, you must create one and add the suggested configuration details for the NPM registries.

Sample Code

MTA Extension Descriptor: NPM Registry Configuration for SAP Web IDE Deployment

```
...
modules:
  - name: di-local-npm-registry
    properties:
      UPSTREAM_LINK: "https://registry.npmjs.org/"
      SAPUPSTREAM_LINK: "https://registry.npmjs.org/"
...
```

Tip

For more information about configuring the SAP Web IDE for SAP HANA, see *Customizing the Environment* in *Related Information* below.

Related Information

Download and Consume SAP Node.js Packages [page 633]
Standard Node.js Packages for XS Advanced [page 635]
Customizing the Environment (SAP Web IDE for SAP HANA Install Guide)

7.1.3 Download and Install JavaScript Data Services

Node.js Data Services (`cds`) provide a native JavaScript client for using Core Data Services for SAP HANA functionality in the JavaScript (node.js) run time container on the XS advanced platform.

Context

JavaScript Data Services (`cds` API) include a native JavaScript client and query builder for Core Data Services for SAP HANA (CDS) for Node.js on SAP HANA XS Advanced Model; `cds` enables programs to consume SAP-HANA-based records as native JavaScript objects. The `cds` library version 1.x is the successor to the XS Data Services (XSDS) library available for the SAP HANA XS engine and supports basic CRUD operations and complex queries on all data types (native SAP HANA and defined in CDS for SAP HANA), and covers associations, transactions, and lazy navigation.

Note

The `@sap/cds` library version 1.15.x for XS advanced is now considered feature complete; it will remain fully supported but will not receive any further enhancements in future releases. Versions 2.x.x and 3.x.x of the library `@sap/cds` are only for use in SAP Cloud Application Programming Model (SAP CAP CDS) scenarios.

To use the `cds` API functions in your `Node.js` programs, perform the following steps.

Procedure

1. Add a dependency to the `@sap/cds` package in the `package.json` file of your Node.js application.

   Sample Code

   ```
   {
       "dependencies": {
           "@sap/cds": "1.x", …
       }, [...]
   }
   ```

2. Import the `@sap/cds` module into your into your Node.js application.

   Use the `require()` statement, as illustrated in the following example:

   Sample Code

   ```
   var cds = require("@sap/cds");
   ```

Related Information

Node.js Data Services [page 690]
Getting Started with Node.js Data Services [page 691]

7.1.3.1 Node.js Data Services

Node.js Data Services (`cds`) provide a native JavaScript client that enables you to use Core Data Services for SAP HANA functionality in the Node.js container on the SAP HANA XS Advanced model platform.

Core Data Services (CDS) are a cross-platform set of concepts and tools to define semantically rich data models for SAP HANA applications. A central part of CDS is the object-relational mapping that links semantically meaningful entities to their technical representation as records in the SAP HANA database.

The `cds` client frees the application developer from using the low-level `hana-client` interface for interaction with SAP HANA data by embedding records as native JavaScript objects in the application logic. The `@sap/cds` module understands CDS meta-data such as associations and maintains consistency when manipulating entity instances as JavaScript objects. Advanced queries are constructed using a fluent query API that follows the CDS Query Language specification as closely as possible.

The `cds` library uses native SAP HANA and CDS functionality where available but provides its own implementation where the core implementation is still incomplete. The `cds` library can also be used without CDS, for example, for legacy application data, while still benefiting from a similar level of convenience. The `cds` library supports basic CRUD operations and complex queries on all native SAP HANA and CDS-defined data types, and covers associations, transactions, and lazy navigation, too.

Related Information

Download and Install JavaScript Data Services [page 689]

7.1.3.2 Getting Started with Node.js Data Services

A quick overview of basic `cds` API functionality.

To give you an idea of what you can do with the `cds` API, this short overview of the basic capabilities of the `cds` library covers the following areas:

- Defining and Generating the JavaScript Objects [page 691]
- Connecting to the Database [page 692]
- Retrieving and Updating Entity Instances [page 693]
- Enabling Lazy Navigation [page 694]
- Using Advanced Queries [page 695]
- Extending cds Beyond CDS [page 697]

Defining and Generating the JavaScript Objects

The following example of a CDS document shows the underlying tables for a bulletin board application that allows users to post short text messages and to reply to other users' posts. In the CDS data model shown here, we define two entities; namely, "user" and "post", as illustrated in the following example code:

Sample Code

CDS Document

```
namespace sap.hana.xs2.cds.examples.data;
context bboard {
    entity user {
    key uid: Integer not null;
    Name: String(63) not null;
    Email: String(63);
    Created: UTCDateTime;
};
type text {
    Text: String(255);
    Lang: String(2);
};
entity post {
    key pid: Integer not null;
    Title: String(63) not null;
    Text: text;
    Author: association [1] to user;
    Parent: association [0..1] to post;
    Rating: Integer;
    Created: UTCDateTime;
};
```

If you use the `@sap/cds` module to import this CDS definition into your Node.js application code, a JavaScript object is generated for each of the entities defined (for example, `user` and `post`. These entity objects are then used to create and retrieve entity instances, which are native JavaScript objects that have all the properties of the underlying CDS data model.

→ Tip

Required type and key information will be inferred automatically.

‹≡› Sample Code

Node.js Application Importing a CDS Model

```
cds.$importEntities([
    { $entity: "sap.hana.xs2.cds.examples.data::bboard.post" },
    { $entity: "sap.hana.xs2.cds.examples.data::bboard.user" }
], main);
function main(error, entities) {
    if (error) {
        console.error(error);
    } else {
        var Post = entities["sap.hana.xs2.cds.examples.data::bboard.post"];
        var User = entities["sap.hana.xs2.cds.examples.data::bboard.user"];
        // …
    }
}
```

Connecting to the Database

Before you can work with entity instances, you need to request a database connection from a connection pool and open a transaction context:

‹≡› Sample Code

```
cds.$getTransaction(function (err, tx) {
    // … transaction context
});
```

When the transaction is complete, close and release the connection by calling the `$close` method:

‹≡› Sample Code

```
tx.$close();
```

Alternatively, you can reuse an existing database connection, for example, a connection provided by some surrounding framework such as “express”:

i Note

The operation is still asynchronous, as the status of the database connection may be “disconnected”.

Sample Code

```
cds.$getTransaction(dbconn, function (err, tx) {
    // ... transaction context ...
});
```

Retrieving and Updating Entity Instances

The `$get` method of a transaction object retrieves an entity instance asynchronously by its key, as illustrated in the following example:

Sample Code

```
tx.$get(Post, { pid: 101 }, function(error, instance) {
    // ...
});
```

The instance argument of the `$get` callback method receives a native JavaScript object whose JSON representation looks something like the following example:

Sample Code

```
{
    pid: 101,
    Title: "First Post!",
    Rating: 1,
    Text: {
        Lang: "EN",
        Text: "This is my first post."
    },
    Author: {
        uid: 1,
        Name: "Kojo",
        Email: "kojo.mensah@acme.com",
        Created: "2014-06-26T11:44:00.000Z"
    },
    Created: "2014-06-26T11:44:00.000Z"
}
```

Note

Associated instances such as the `Author` instance are retrieved automatically by default and become part of the `Post` instance.

Modifications to an entity instance remain local to the transaction context of the current Node.js application. To transfer any changes to the database, call the `$save` method of the modified instance:

Sample Code

Update both a Post and a User

```
// update post and user
post.Rating++;
```

```
post.Author.Email = "kojo.mensah@acmehana.com";
tx.$save(post, function(error, instance) { /* ... */ });
```

To create a new instance, invoke `$save` with the entity type and the skeleton value for the new instance to create:

> **i Note**
>
> Unless you are using SAP HANA's sequence mechanism to generate new values automatically, it is necessary to supply your own key values.

Sample Code

```
// create new post
tx.$save({
    $entity: Post,
    pid: 102,
    Title: "New Post",
    Text: { Text: "Hello BBoard!", Text: "EN" },
    Author: post.Author,
    Rating: 1,
    Created: new Date()
}, function (error, newPost) { … });
```

Enabling Lazy Navigation

The `cds` library enables you to perform a "lazy" loading of associated entity instances; lazy loading is normally used to help reduce the amount of data transferred to the application server.

Sample Code

```
cds.$importEntity({
    $entity: "sap.hana.xs2.cds.examples.data::bboard.post",
    $fields: { Author: { $association: { $lazy: true } } }
}, function(error, entities) { ... });
```

If lazy loading is enabled, `cds` only retrieves those instances that the application actually needs to process. To retrieve the targets of a lazy association, we use the `$load` function, as illustrated in the following example:

Sample Code

```
tx.$get(Post, { pid: 101 }, function (error, post) {  // retrieve Post only
    if (post.Rating > 0) {
        post.Author.$load(function (error, author) {  // now retrieve
associated User
            var sender = author.Email;
            // ...
        });
    }
});
```

→ Tip

Although lazy loading can cause additional round trips to the database, it can also reduce the total amount of data transferred if lazy associations are followed only infrequently.

Using Advanced Queries

For complex ad hoc queries `cds` provides a query builder based on the CDS Query Language. The `$query` method of the entity returns a query object that enables you to build a complex query step-by-step:

‹≡› Sample Code

The `cds` `$query` Method

```
var qPosts = Post.$query(tx);
```

Queries can be refined with (or derived from) existing queries by using relational operators such as `$project`, `$where`, or `$matching`. The `$project` operator method specifies the fields which should be returned by the query:

‹≡› Sample Code

```
var qTitleAndAuthor = Post.$query().$project({
    Author: {
        Name: true,
        Email: true
    },
    Title: true
});
```

The argument to `$project` is a JavaScript object where the fields to be included are marked "`true`". Without projection, all fields defined in the entity are returned by default. The `$where` operator method filters the result of the query by some conditional expression:

‹≡› Sample Code

```
var qImportantTitleAndAuthor =
qTitleAndAuthor.$where(Post.Author.Name.$eq("Kojo"));
```

The `$matching` operator method provides a more concise, JSON-based syntax with a subset of the full expressions language supported by `$where` :

‹≡› Sample Code

```
qImportantTitleAndAuthor = qImportantTitleAndAuthor.$matching({ Rating:
{ $gt: 2 } });
```

None of the query building steps shown above involves any kind of communication with the index server. To execute the query and retrieve the result, invoke the `$execute` method, as illustrated in the following example:

Sample Code

```
qImportantTitleAndAuthor.$execute(function (error, results) {
    for (var r in results) { ... }
});
```

`cds` creates the SQL query in the background, automatically inserting JOINs based on all the projection and selection operations defined in the query. The above sequence of statements results in the SQL query illustrated in the following example:

Sample Code

```
SELECT "t0.Author"."Name" AS "t0.Author.Name",
       "t0.Author"."Email" AS "t0.Author.Email",
       "t0"."Title" AS "t0.Title"
  FROM "sap.hana.xs2.cds.examples.data::bboard.post" "t0"
  JOIN "sap.hana.xs2.cds.examples.data::bboard.user" "t0.Author"
    ON "t0"."Author.uid"="t0.Author"."uid"
 WHERE ("t0"."Rating" > 2) AND ("t0.Author"."Name" = 'Kojo)
```

The following example shows the probable query result:

Sample Code

```
[{
    Author: {
        Name: "Kojo",
        Email: "kojo.mensah@acme.com"
    },
    Title: "Fail"
}]
```

Note

The result returned by an advanced query is a plain value, not a managed entity instance. The main difference between a plain value and an entity instance is the handling of updates and associated entities.

You may freely convert between plain values and managed instances, however, or you could use the `$find` method (instead of `$matching`):

Sample Code

```
tx.$find(Post, {
    Rating: { $ge: 1, $lt: 5 },
    Parent: { $null: false }
}, function (error, posts) { /* ... */ });
```

Extending cds Beyond CDS

The `cds` library enables you to import existing CDS entities, but it can also transform entity definitions and derive new entities. This extensibility allows you to work with legacy SQL data models, for example, by enriching the data model to make implicit associations between plain database tables **explicit**:

Sample Code

```
cds.$importEntities([{
    $schema: "SCHEMA",
    $table: "BBOARD_POSTS",
    $name: "bbposts",
    $fields: {
        Author: {
            $association: { $entity:
"sap.hana.xs2.cds.examples.data::bboard.user" },
            uid: { $column: "POSTED_BY" }
        }
    }
}], function (error, entities) { /* ... */ });
```

Note

`cds` imports every table column as an entity property. However, since plain SQL lacks the metadata for the associations, you must supply the information about the associations manually.

Related Information

Download and Install JavaScript Data Services [page 689]
Node.js Data Services [page 690]

7.1.3.3 Node.js Data Services in XSJS Compatibility Mode

An extension of Node.js Data Services that provides compatibility with XS Classic Data Services (XSDS) library.

If you have code to run in the XS JavaScript (XSJS) compatibility mode, you can use an extension to the XS Advanced Node.js data services that is compatible with the basic XS Classic data services library (XSDS).

Restriction

This extension is intended to be used only within the XSJS Compatibility Layer run-time container.

To initialize the XS Advanced Node.js extension from an XSJS file, it needs to be invoked as follows:

Sample Code

```
var conn = $.db.getConnection();
var cds = $.require("@sap/cds").xsjs(conn);
```

The connection to the database is managed by the connection provided in the argument to the XSJS call; the connection object can be modified through the `cds.Transaction` object, for example, you can close the connection with the method `cds.Transaction.$close();` .

Similarly to the way it is done in standard (not compatibility-mode) `node-cds`, you can import entities using the syntax and the synchronous style of XSDS, as illustrated in the following example:

Sample Code

The `cds.$importEntity` Method

```
var Post = cds.$importEntity("sap.hana.xs2.cds.examples.data", "bboard.post",
    { Comments:
        { $association:
            { $entity:
                "sap.hana.xs2.cds.examples.data::bboard.user",
                $viaBacklink: "Post"
            }
        }
    });
```

You can retrieve entities either in **managed** or in **unmanaged** mode, using the operations already known from standard (not compatibility-mode) `node-cds`, but in the synchronous approach of XSJS. For example, the `$get` method of an entity will retrieve an entity instance by its key:

Sample Code

```
var post = Post.$get({ pid: 101 });
```

To create a new instance, invoke the “new” operator and persist the resulting object:

Sample Code

```
// create new post
var newpost = new Post({ pid: 102,
                         Title: "New Post",
                         Text: { Text: "Hello BBoard!", Text: "EN" },
                         Author: post.Author,
                         Rating: 1,
                         Created: new Date() });
newpost.$save();
```

For complex ad hoc queries, you can use the same query builder as the one available in standard (not compatibility-mode) `node-cds`. The `$query` method of the entity constructor returns a query object that enables you to build a complex query step by step:

Sample Code

The `$query` Method

```
var results = Post.$query().$project({
    Author: {
        Name: true,
        Email: true
    },
    Title: true
}).$execute();
```

Related Information

Download and Install JavaScript Data Services [page 689]
Getting Started with Node.js Data Services [page 691]

7.1.4 Tutorial: Setting up your JavaScript Application in XS Advanced

Tutorials that show you how to set up JavaScript applications for XS advanced.

This section includes a selection of tutorials that show you how to set up various basic but important aspects of the application that you plan to deploy on the XS advanced JavaScript run time.

> ℹ Note
>
> The tutorials in this section focus on tools provided with the XS advanced command-line client.

Prerequisites

Before you start the tutorials in this section, bear in mind the following prerequisites:

- SAP HANA XS advanced model must be installed on your SAP HANA system and you can log in to it using the XS advanced command-line client. For more information, see *Related Links* below.
- Node.js is installed locally. If you use a proxy to connect to the Internet, you might need to configure it. For more information about proxy configuration settings for NPM, see *npm Documentation* in the *Related Links* below.
- SAP Node.js packages
  The archive (`xs_javascript-<version>-bundle.tar.gz`) is available on the SAP HANA media and the SAP Marketplace (SMP); it includes a selection of Node.js packages developed by SAP for use with the Node.js applications running in the XS advanced run time.

  > → Tip
  >
  > Alternatively, you can configure NPM to resolve SAP-related NPM dependencies using the public NPM Registry (at `https://registry.npmjs.org`).

Related Information

Standard Node.js Packages for XS Advanced [page 635]
The NPM Registry [page 687]
Deploy a Node.js Hello-World Application using the XS CLI [page 82]
Working with the XS Advanced Command-Line Client [page 75]

7.1.4.1 Create a Node.js Application

Create a sample Node.js application.

Procedure

1. Create a new directory named `node-tutorial`.
2. Create a `manifest.yml` file in the `node-tutorial` directory with the following content:

   ```
   ---
   applications:
   - name: myapp
     path: myapp
   ```

   This descriptor is used to describe applications and where their sources are located.
3. Create a new directory inside `node-tutorial` named `myapp`.
4. Inside the `myapp` directory, create a new file called `package.json` with the following content:

   ```
   {
     "name": "myapp",
     "description": "My App",
     "version": "0.0.1",
     "dependencies": {
       "express": "4.16.2"
     },
     "scripts": {
       "start": "node start.js"
     }
   }
   ```
5. Inside the `myapp` directory, create another file called `start.js` with the following content:

   → Tip

   the XS Advanced platform provides "`XS_BIND_ADDRESS`" to ensure that internal application ports can only be reached through the XS advanced platform router.

   ```
   var express = require('express');
   var app = express();
   app.get('/', function (req, res) {
     res.send('Hello World!');
   });
   var port =
   process.env.PORT || 3000;
   var bind_address = process.env.XS_BIND_ADDRESS || undefined;
   app.listen(port, bind_address, function () {
     console.log('myapp listening on port ' + port);
   });
   ```
6. To install dependencies execute the `npm install` command in the `myapp` directory.
7. Deploy the application on XS advanced.

   Execute the following command in the `node-tutorial` directory.

   ```
   xs push
   ```

You can check the state and URL of your application by using the `xs apps` command.

Output Code

```
> xs apps
Getting apps in org <orgname> / space <spacename> as XSA_ADMIN...
Found apps:
name    requested state   instances   memory    disk          urls
-------------------------------------------------------------------
myapp   STARTED           1/1         1.00 GB   <unlimited>
<URL>
```

8. Open a browser window with your application URL.

   You should see the `Hello World!` message.

7.1.4.2 Consume XS Advanced Services

XS advanced provides services, which can be used by applications.

Prerequisites

You have configured you command-line interface to use the public NPM registry. For more information, see *Related Links* below.

Context

XS advanced provides a convenient package (`@sap/xsenv`) for Node.js applications, which can be used to read bound services. To use the `@sap/xsenv` package, you have to add it to your application's dependencies, which are specified in a corresponding `package.json` file.

Procedure

1. Navigate to the `myapp` directory.
2. Execute `npm install @sap/xsenv@^1.2.9 --save` to install the `@sap/xsenv` module and add it as a dependency in the `package.json` file.

   After the installation of the package is complete, the dependencies section in the `package.json` file should have the following content:

```
"dependencies": {
  "express": "4.16.2",
  "@sap/xsenv": "^1.2.9"
},
```

3. Update the `start.js` file with the content shown in the following example:

```
var express = require('express');
var app = express();
var xsenv = require('@sap/xsenv');
var services = xsenv.getServices({ hana:'myhana' });
app.get('/', function (req, res) {
  res.send('Using HANA ' + services.hana.host + ':' + services.hana.port);
});
var port = process.env.PORT || 3000;
app.listen(port, function () {
  console.log('myapp listening on port ' + port);
});
```

4. Create a SAP HANA service instance named `myhana` .

   The new service instance must have the service type "`hana`" and the service plan "`hdi-shared`", as illustrated in the following command:

```
xs create-service hana hdi-shared myhana
```

5. Replace the content of the `manifest.yml` file in the `node-tutorial` directory with the following:

```
---
applications:
- name: myapp
  path: myapp
  services:
    - myhana
```

   The XS advanced run time will bind the `myhana` service instance to the `myapp` application during deployment.

6. Update the application on XS advanced.

   Install the modified version of the application to XS advanced run-time environment by running the `xs push` command in the `node-tutorial` directory, as follows:

```
xs push
```

7. Display the URL for accessing the updated application after deployment.

```
xs app myapp --urls
```

8. Open a browser window and paste the displayed URL in to start the application.

   You should see a message similar to: Using SAP HANA `<hana host>:<hana port>`.

Related Information

Standard Node.js Packages for XS Advanced [page 635]
The NPM Registry [page 687]

7.1.4.3 Connect to SAP HANA

Context

XS advanced provides a package (`@sap/hdbext`) for Node.js applications, which can be used to connect to SAP HANA. To use it, you have to add it to your application.

> → Tip
>
> To ensure that both SSL and trust certificates are set up automatically, it is recommended to use @sap/hdbext to connect to SAP HANA rather than use @sap/hana-client.

Procedure

1. Navigate to the `<myapp>` directory.
2. Execute `npm install @sap/hdbext@^5.0.0 --save` to install the `@sap/hdbext` module and add it as a dependency in the `package.json` file.

   After the installation of the package is complete, the dependencies section in the `package.json` file should have the following content:

   ```
   "dependencies": {
     "express": "4.16.2",
     "@sap/xsenv": "^1.2.9"
     "@sap/hdbext": "^5.0.0"
   },
   ```

3. Update the `start.js` file and replace this content:

   ```
   app.get('/', function (req, res) {
     res.send('Using HANA ' + services.hana.host + ':' + services.hana.port);
   });
   ```

   with the following:

   ```
   var hdbext = require('@sap/hdbext');
   app.use('/', hdbext.middleware(services.hana));
   app.get('/', function (req, res, next) {
       req.db.exec('SELECT CURRENT_UTCTIMESTAMP FROM DUMMY', function (err,
   rows) {
       if (err) { return next(err); }
       res.send('Current HANA time (UTC): ' + rows[0].CURRENT_UTCTIMESTAMP);
     });
   });
   ```

   `hdbext.middleware` will connect to SAP HANA automatically on each access to the specified path (`/`) in this case. Afterwards the connection is available in `req.db`. This is the client object of the hdb driver. The connection is closed automatically at the end of the request.

4. Update the application on XS advanced.

   Execute the following command in the `node-tutorial` directory.

   ```
   xs push
   ```

5. Open a browser window with the `<myapp>` application URL.

   You should see the current SAP HANA time.

Related Information

Standard Node.js Packages for XS Advanced [page 635]
@sap/hdbext

7.1.4.4 Set up Authentication for your XS Advanced JavaScript Application

Instructions to help you set up user authentication for your JavaScript application in XS advanced.

Context

Authentication and authorization in the XS advanced run time are implemented via OAuth2 with the help of the User Account and Authentication (UAA) service. For more information, see *Maintaining Application Security* in the *Related Links* below.

Procedure

1. Create a UAA service instance named `myuaa`.

   ```
   xs create-service xsuaa default myuaa
   ```

2. Add the *myuaa* service to the application's `manifest.yml` file.

   The updated `manifest.yml` should look like the following example:

   ```
   ---
   applications:
   - name: myapp
     path: myapp
     services:
       - myhana
       - myuaa
   ```

The XS advanced JavaScript run time will bind the *myuaa* service instance (as well as *myhana*) to the *myapp* application during deployment.

> **i Note**
>
> Authentication and authorization are topics that are related to more components than just one application.

The XS advanced run time supports the microservices concept in which a single business application my contain several different applications (microservices). A single entry point for these microservices is provided by the application router (`@sap/approuter`). The application router receives all the requests, performs authentication, and forwards the authenticated requests to the appropriate microservice along with a JSON Web Token (JWT). The application router can also perform first-level authorization.

To complete the authorization part of the tutorial you have to add one more application - the application router.

3. Create a directory named `web` in the `node-tutorial` directory.
4. In the `web` directory, create a subdirectory named `resources`.
5. In the `resources` directory, create file `index.html` and add the following content.

```
<html>
<head>
    <title>JavaScript Tutorial</title>
</head>
<body>
  <h1>JavaScript Tutorial</h1>
  <a href="/myapp/">myapp</a>
</body>
</html>
```

This will be the application's entry page.

6. Create a `package.json` file in the `web` directory with the following content:

```
{
    "name": "myapp-entry-point",
    "scripts": {
        "start": "node node_modules/@sap/approuter/approuter.js"
    }
}
```

7. In the `web` directory execute `npm install @sap/approuter --save` to install the `@sap/approuter` module and add it as a dependency in the `package.json` file.

   After the installation of the package is complete, the dependencies section in the `package.json` file should have the following content:

```
  "dependencies": {
    "@sap/approuter": "^2.7.0"
  },
```

8. Add the following content to the end of the `manifest.yml` file in the `node-tutorial` directory:

```
- name: web
  path: web
  env:
    destinations: >
      [
        {
          "name":"myapp",
          "url":"<myapp url>",
```

```
          "forwardAuthToken": true
        }
      ]
  services:
    - myuaa
```

The `destinations` environment variable defines the destinations to which the application router forwards requests.

Set the `url` property to the URL of the `myapp` application, which is displayed in the output generated by the command: `xs app myapp --urls`.

> **i** Note
>
> The `myuaa` service is bound to the `web` application during deployment.

9. Create the application descriptor (`xs-app.json`) in the `web` directory and add the following content:

```
{
  "routes": [
    {
      "source": "^/myapp/(.*)$",
      "target": "$1",
      "destination": "myapp"
    }
  ]
}
```

10. In the `myapp` execute `npm install @sap/xssec --save` to install the `@sap/xssec` module and add it as a dependency in the `package.json` file.
11. Execute `npm install passport --save` to install the `passport` module and add it as a dependency in the `package.json` file.

    The list of dependencies should look like the following example:

```
"dependencies": {
  "express": "4.16.2",
  "@sap/xsenv": "^1.2.9",
  "@sap/hdbext": "^5.0.0",
  "@sap/xssec": "^2.1.6",
  "passport": "^0.3.2"
}
```

12. Verify that the request is authenticated by checking the JWT token included in the request.

    Replace the content of the application startup file (`myapp/start.js`) with the following.

```
var express = require('express');
var xsenv = require('@sap/xsenv');
var passport = require('passport');
var JWTStrategy = require('@sap/xssec').JWTStrategy;
var app = express();
var services = xsenv.getServices({ hana:'myhana', uaa:'myuaa' });
passport.use(new JWTStrategy(services.uaa));
app.use(passport.initialize());
app.use(passport.authenticate('JWT', { session: false }));
app.get('/', function (req, res, next) {
  res.send('Application user: ' + req.user.id + '<br>' + 'HANA user: ' +
services.hana.user);
});
var port = process.env.PORT || 3000;
app.listen(port, function () {
  console.log('myapp listening on port ' + port);
```

```
});
```

13. Go to the `node-tutorial` directory and deploy the application by executing the following command.

    ```
    xs push
    ```

    This command updates the `myapp` application and also deploys the `web` application.

    > **i Note**
    >
    > From this point in the tutorial we make requests using the URL of the `web` application, which forwards the requests to the *myapp* application.

14. Find the URL of the `web` application using the `xs apps` command.
15. Request the `web` application with path `/myapp/`.
16. Enter the credentials of a valid user. If no custom identity provider (IDP) is configured, you should use the credential of a known SAP HANA user.

    You should see a message similar to the following in the browser:

    - *Application user:* `<login user>`
      The name of the user who logs on to the application.
    - *HANA user:* `<db user>`
      The technical user used to connect to SAP HANA.

    > **i Note**
    >
    > An application user is not the same as a database user.

    > **i Note**
    >
    > Both `myapp` and `web` applications are bound to the same UAA service instance - `myapp`. In this scenario authentication is handled by the `web` application.

Related Information

Maintaining Application Security in XS Advanced [page 957]

7.1.4.5 Set up Authorization for your XS Advanced JavaScript Application

Instructions to help you set up user authorization for your JavaScript application in XS advanced.

Context

The web application can also handle user authorization, for example, with defined scopes and role templates. Scopes and role templates are defined in the application security descriptor (`xs-security.json`), and the application router's descriptor (`xs-app.json`) can contain configurations with authorization restrictions, for example, the specific scopes a user requires to access a resource. For information about setting up authorization scopes, role assignment, and role collections, see *Related Links* below.

It is a general recommendation to implement authorization in the different microservices as well.

Procedure

1. Add an authorization check to your application.

   To add an authorization check in the `myapp` application, you need to modify the application's startup file `start.js` file; replace the express handler for `GET` requests to path "`/`" with the following code:

   ```
   app.get('/', function (req, res, next) {
     var isAuthorized = req.authInfo.checkScope('uaa.user');
     if (isAuthorized) {
       res.send('Application user: ' + req.user.id);
     } else {
       res.status(403).send('Forbidden');
     }
   });
   ```

2. Deploy the modified application.

   You can update the `myapp` application by redeploying, for example, by executing the command `xs push myapp` in the `node-tutorial` directory.

   In this example the `myapp` application checks whether the current user has the authorization scope `uaa.user`. Since this is a default scope, the authorization check is successful.

   > → Tip
   >
   > It is possible to create a UAA service instance with scopes that are specific to your application.

Related Information

Standard Node.js Packages for XS Advanced [page 635]

Maintaining XS Advanced Application Routes and Destinations [page 912]

7.1.4.6 Set up Logging and Tracing for your XS Advanced JavaScript Application

Step-by-step instructions to help you write log and trace files for your JavaScript application in XS advanced.

Context

SAP provides a Node.js package for logging and tracing (`@sap/logging`).

> **i** Note
>
> Logs are for the administrator of an application; trace files are for developers or support.

Events that need to be logged are related to how an application operates. For example, entries are created in logs and trace files if a user has been refused access to an application after too many failed log-in attempts, or the application cannot display results returned by a remote HTTP service because the remote server is down. The administrator of an application does not need to know how it is implemented; they should just be able to determine the state of the application itself.

Traces are mainly used when a problem has occurred and further investigation is necessary at the JavaScript code level.

To consume the `@sap/logging` library, execute the following steps:

Procedure

1. Go to the `myapp` directory.
2. Execute `npm install @sap/logging --save` to install the `@sap/logging` module and add it as a dependency in the `package.json` file.

```
"dependencies": {
  "express": "4.16.2",
  "@sap/xsenv": "1.2.9",
  "@sap/hdbext": "^4.7.0",
  "@sap/xssec": "^2.1.6",
  "passport": "^0.3.2",
  "@sap/logging": "^4.0.2"
}
```

3. Replace the content of the application startup file (`start.js`) with the following code:

```
var express = require('express');
var xsenv = require('@sap/xsenv');
var passport = require('passport');
var JWTStrategy = require('@sap/xssec').JWTStrategy;
```

```
var logging = require('@sap/logging');
var appContext = logging.createAppContext();
var app = express();
app.use(logging.middleware({ appContext: appContext }));
passport.use(new JWTStrategy(xsenv.getServices({uaa:{tag:'xsuaa'}}).uaa));
app.use(passport.initialize());
app.use(passport.authenticate('JWT', { session: false }));
app.get('/', function (req, res, next) {
  var logger = req.loggingContext.getLogger('/Application');
  var tracer = req.loggingContext.getTracer(__filename);
  var isAuthorized = req.authInfo.checkScope('example.scope');
  if (isAuthorized) {
    tracer.debug("Authorization success. User: " + req.user.id + ", Path:
'/'.");
    res.send('Application user: ' + req.user.id);
  } else {
    logger.info("Authorization failed. User: " + req.user.id + ", Path:
'/'.");
    res.status(403).send('Forbidden');
  }
});
var port = process.env.PORT || 3000;
app.listen(port, function () {
  console.log('myapp listening on port ' + port);
});
```

> **i Note**
>
> To demonstrate what happens with a failed authorization, the used scope (`example.scope`) is not among the scopes the test user should possess.

4. Update the `myapp` application by executing the command `xs push myapp` in the `node-tutorial` directory.
5. Call the `web` application using the path `/myapp/`.

   There will be a message in the Web browser saying you are not authorized.
6. Check the logs of the `myapp` application.

   In a command shell, execute the command `xs logs myapp --recent`. The displayed output should contain an entry in the log format used by the `@sap/logging` library with the severity level "`info`" and a message similar to the following example:

   ```
   Authorization failed. User: <HANA-user>, Path: '/'.
   ```

   If you change the required user scope in the JavaScript code to one already assigned to the requesting user (for example, "`uaa.user`") and update the application in the XS advanced run time, for example, using the `xs push myapp` command, then a trace entry with severity status `debug` is generated.

   > **→ Tip**
   >
   > The `debug` level is not enabled by default; this entry is only generated if the `debug` level is enabled for the application.
7. Enable `debug` tracing for the application.

   ```
   xs set-logging-level myapp /start.js debug
   ```

> **Note**
>
> If you are using SAP Web IDE for SAP HANA, you can use the `<XS_APP_LOG_LEVEL>` environment variable to set the logging level. In SAP Web IDE for SAP HANA, the default application logging level is `ERROR`.

8. Call the `web` application using the path */myapp/*.

   The result should contain the name of the requesting user.

9. View the application trace files.

   In a command shell, execute the command `xs logs myapp --recent` to view the trace entries. The trace entries with severity `debug` should contain a message similar to the following example:

   ```
   Authorization success. User: <HANA-user>, Path: '/'
   ```

Related Information

Standard Node.js Packages for XS Advanced [page 635]

7.1.4.7 Use the XSJS Compatibility Layer in XS Advanced

The XSJS compatibility layer is a framework that allows you to run XS classic JavaScript applications in XS advanced.

Procedure

1. Create a new directory inside `node-tutorial` named `xsjs`.
2. Create `package.json` file in the `xsjs` directory containing the following code.

   ```
   {
     "name": "xsjs",
     "version": "0.0.1",
     "scripts": {
       "start": "node start.js"
     }
   }
   ```

3. Execute `npm install @sap/xsenv@^1.2.9 --save` to install the `@sap/xsenv` module and add it as a dependency in the `package.json` file.
4. Execute `npm install @sap/xsjs@^3.3.1 --save` to install the `@sap/xsjs` module and add it as a dependency in the `package.json` file.
5. Create `start.js` file in the `xsjs` directory containing the following.

   ```
   var xsjs = require('@sap/xsjs');
   var xsenv = require('@sap/xsenv');
   var port = process.env.PORT || 3000;
   ```

```
var options = xsenv.getServices({ hana:{tag:'hana'}, uaa:{tag:'xsuaa'} });
xsjs(options).listen(port);
console.log('XSJS application listening on port %d', port);
```

The XSJS compatibility layer takes an object containing configurations. In this example SAP HANA and UAA configurations are provided.

6. Create a new directory named `lib` inside the directory `xsjs`.
7. Create a new file named `get-time.xsjs` inside the `lib` directory and add the following content to it.

```
var conn = $.hdb.getConnection();
var resultSet = conn.executeQuery('SELECT CURRENT_UTCTIMESTAMP FROM DUMMY');
$.response.setBody('Current HANA time (UTC): ' +
resultSet[0].CURRENT_UTCTIMESTAMP);
conn.close();
```

8. Navigate to the `node-tutorial` directory and add the following content at the end of the `manifest.yml` file.

```
- name: xsjs
  path: xsjs
  services:
    - myhana
    - myuaa
```

9. Deploy the XS JavaScript application with the following command executed from the `node-tutorial` directory:

   ```
   xs push xsjs
   ```

10. Get the URL of the `xsjs` application by running the `xs app xsjs --urls` command.

> **i Note**
>
> Authentication is enabled by default in the compatibility layer; you are not able to directly access the `xsjs` application.

11. Update the configuration for the Web application in the `manifest.yml` file in the `node-tutorial` directory:

```
- name: web
  path: web
  env:
    destinations: >
      [
        {
          "name":"myapp",
          "url":"<myapp url>",
          "forwardAuthToken": true
        },
        {
          "name":"xsjs",
          "url":"<xsjs url>",
          "forwardAuthToken": true
        }
      ]
  services:
    - myuaa
```

Set the `url` property of the `xsjs` destination to the URL of the `xsjs` application.

> **Note**
>
> The application-router based application can forward requests to two microservices: the `myapp` application and the `xsjs` application. This should be configured in the XS advanced application descriptor (`xs-app.json`) located in the `web` directory.

12. Navigate to the `web` directory and replace the content of the application descriptor (`xs-app.json`) with the following content:

```
{
  "routes": [
    {
      "source": "^/myapp/(.*)$",
      "target": "$1",
      "destination": "myapp"
    },
    {
      "source": "^/xsjs/(.*)$",
      "target": "$1",
      "destination": "xsjs"
    }
  ]
}
```

13. Add a link to the `xsjs` application in `web/resources/index.html`, which should then look like the following example:

```
<html>
<head>
    <title>JavaScript Tutorial</title>
</head>
<body>
  <h1>JavaScript Tutorial</h1>
  <a href="/myapp/">myapp</a>
  <p>
  <a href="/xsjs/get-time.xsjs">xsjs</a>
</body>
</html>
```

14. Navigate to the `node-tutorial` directory and execute the command `xs push web` to update the `web` application.
15. Display the URL required for access to the XS advanced application `web` by executing the command `xs apps` command and use the URL to start the application in a Web browser.

> **Note**
>
> You may need to log in again, for example, if your session has expired.

16. Choose the link `xsjs` to see a page displaying the current SAP HANA time.

Related Information

Tutorial: Setting up your JavaScript Application in XS Advanced [page 699]
The SAP HANA XS Advanced JavaScript Run Time [page 627]

7.1.4.8 Trouble Shoot XS Advanced JavaScript Applications

Investigate problems that occur during deployment and execution of an XS advanced application.

Enhancing the logs

There are several possibilities to enhance the logs for Node.js applications in XS advanced. SAP-specific packages usually use the `@sap/logging` module. With the `@sap/logging` module, you can set the logging and tracing severity level to the most verbose level in the following ways:

- Set the environment variable `XS_APP_LOG_LEVEL` to `debug`.

  > **i** Note
  >
  > You must restart the application to reset the logging level.

- Use the `xs set-logging-level` command, as illustrated in the following example:
  `xs set-logging-level <application-name> '*' debug`

  > **i** Note
  >
  > You do not need to restart the application to reset the logging level.

Other Node.js packages (or dependencies of SAP-specific packages) might use different logging mechanisms. For example, many applications use the `debug` package. Additional traces can also be enabled by setting the `<DEBUG>` environment variable; setting `<DEBUG>` to * will write a large amount of information to the logs and traces very quickly, which could cause problems with disk space.

Internal Node.js traces can be enabled with the `NODE_DEBUG` environment variable.

> ⚠ Caution
>
> Enabling some of these options may write sensitive data to logs and traces in an insecure form.

Related Information

The SAP HANA XS Advanced JavaScript Run Time [page 627]
Download and Consume SAP Node.js Packages [page 633]
Tutorial: Setting up your JavaScript Application in XS Advanced [page 699]

7.1.4.9 Debug an XS Advanced JavaScript Application Using Command-Line Tools

You can use this procedure to debug your Node.js applications.

Prerequisites

The XS command-line client is installed on the machine where you will debug a Node.js application deployed on the XS advanced run time.

> → Tip
>
> For more information about downloading and installing the XS CLI client, see *Related Links* below.

Context

To debug an XS advanced JavaScript application using the XS CLI, perform the following steps:

Procedure

1. Enable debug **mode** for the Node.js application you want to debug, for example, using the following command:

   ```
   xs enable-debugging <application-name> --startup
   ```

   This will open a secure debugging connection from the machine on which the XS command line tool is running to the application instance. The port the debugger should attach to is part of the output.

   > → Tip
   >
   > You can configure which port to be used with the `--port` option.

2. Configure a debugger of choice to attach to the host where the XS command-line tool is running and the port displayed in the output of the `xs enable-debugging` command.

   > i Note
   >
   > If you are using a custom debugging tool, bear in mind that tools that try to read sources directly from the file system are not suitable for debugging applications deployed on the XS advanced run time.

   The standard command-line debugger client for Node.js is a suitable debugging tool; it can be started with the following command:

   ```
   node inspect localhost:<debug-port>
   ```

The `inspect` command replaces `debug` in recent versions of Node.js, and `<debug-port>` is the port number displayed in the output returned by the **`xs enable-debugging`** command. For more information about how to use the standard XS advanced debugging tool, see *Related Links* below.

Related Information

The XS Command-Line Interface Reference
Node.js Debugger Documentation

7.2 The SAP HANA XS Advanced Java Run Time

SAP HANA XS advanced provides a Java run time to which you can deploy your Java applications.

The Java run time for SAP HANA XS advanced provides the Tomcat, TomEE, TomEE 7, and Java Main run times to deploy your Java code. During application deployment, the build pack ensures that the correct SAP Java Virtual Machine (JVM) is provided and that the appropriate data sources for the SAP HANA HDI container are bound to the corresponding application container.

> **Note**
>
> The XS advanced run-time platform makes no assumptions about which frameworks and libraries to use to implement the Java micro service.

XS advanced does, however, provide the following components to help build a Java micro service:

- Tomcat, TomEE, TomEE 7, and Java Main run times
- Setup of SAP HANA data sources
- Java library to support authentication with JSON Web Tokens (JWT)
  Used for communication between the application router, micro services, and the database
- Java library to better support SAP HANA Core Data Services (CDS)

To use Tomcat on SAP JVM 8, specify the `java-buildpack` in your XS advanced application's deployment manifest (`manifest.yml`), as illustrated in the following example:

> **Sample Code**

```
applications:
  - name: my-java-service
    buildpack: path/to/java-buildpack
    services:
      - my-hdi-container
      - my-uaa-instance
```

When you use the Tomcat, TomEE or TomEE 7 run time, you can rely on certain standard APIs:

> **Note**
>
> SAP plans to remove the outdated TomEE 1.7 version from the SAP Java Buildpack for SAP HANA XS Advanced. Although TomEE 1.7 will continue to be supported until the end of the deprecation period, it is

recommended to migrate to TomEE 7 as soon as possible, as described in the *TomEE Migration Guide* listed in *Related Information* below and in SAP Note 3208765. TomEE 7 is already included in previous versions of the SAP Java Buildpack for SAP HANA XS Advanced.

Run time	Tomcat/TomEE	Supported specification version
Tomcat	Apache Tomcat 8	Java Servlets 3.1
		Java ServerPages (JSP) 2.3
		Expression Language (EL) 3.0
		Debugging Support for Other Languages 1.0
		Java API for WebSocket 1.1
TomEE	Apache TomEE JAX-RS (Java EE 6)	Java Servlets 3.0
		Java ServerPages (JSP) 2.2
		Expression Language (EL) 2.2
		Debugging Support for Other Languages 1.0
		Standard Tag Library for JavaServer Pages (JSTL) 1.2
		Java API for WebSocket 1.1
		Java ServerFaces (JSF) 2.0
		Java Transaction API (JTA) 1.1
		Java Persistence API (JPA) 2.0
		Java Contexts and Dependency Injection (CDI) 1.0
		Java Authentication and Authorization Service (JAAS)
		Java Authorization Contract for Containers (JACC) 1.3
		JavaMail API 1.4
		Bean Validation 1.0
		Enterprise JavaBeans 3.1
		Java API for RESTful Web Services (JAX-RS) 1.1

Run time	Tomcat/TomEE	Supported specification version
TomEE 7	Apache TomEE 7 (Java EE 7 Web Profile)	Java Servlet 3.1 JavaServer Pages (JSP) 2.3 Expression Language (EL) 3.0 WebSocket 1.1 Standard Tag Library for JavaServer Pages (JSTL) 1.2 Java API for RESTful Web Services (JAX-RS) 2.0 JavaServer Faces (JSF) 2.2 Debugging Support for Other Languages 1.0 Enterprise JavaBeans (EJB) Lite 3.2 Java Transaction API (JTA) 1.2 Java Persistence API (JPA) 2.1 Bean Validation 1.1 Managed Beans 1.0 Interceptors 1.2 Common Annotations for Java Platform 1.2 Dependency Injection for Java 1.0 Contexts and Dependency Injection for Java EE platform 1.1

Related Information

TomEE Migration Guide
SAP Note 3208765

7.2.1 Download and Consume Java Libraries

A selection of SAP-specific and ready-to-use Java client libraries is available for download from the SAP Service Marketplace.

Prerequisites

- You have installed Maven on your development system.

Context

The recommended way to consume SAP Java libraries is to use the Maven Central repository. If, however, you want to download the SAP Java Client libraries to a local directory, perform the following steps:

> **i Note**
>
> The most recent versions of the standard SAP Java libraries are available on the Maven Central repository, which is now the default way to consume the Java libraries provided by SAP. For more details, see *Related Information* below.

Procedure

1. Download the package containing the library you need.

   The ZIP archive `XS JAVA 1` is available on SAP Service Marketplace in the following location: *Support Portal* > *Download Software*.

   > **→ Tip**
   >
   > To find the ZIP file for download, type "XS JAVA 1" in the *Search* box.

   For more information about locating individual SAP Java libraries on Maven Central, see *Related Information* below.
2. Extract the contents of the `XS JAVA 1` archive (or the library from Maven Central) into a folder in your local file system.
3. Run a Maven build.

   Navigate to the folder in your local file system where you extracted the Java client libraries and run a Maven build with the following command:

   ```
   mvn clean install
   ```

   The Java client libraries are installed in the local Maven repository from where they can be consumed by your Maven project.

Related Information

Standard Java Client Libraries for XS Advanced [page 720]
Tutorials: Setting Up the Java Run-Time Environment in XS Advanced [page 735]
Maven Central Repository

7.2.1.1 Standard Java Client Libraries for XS Advanced

A collection of Java client libraries developed by SAP is provided to help you develop Java applications for SAP HANA XS advanced.

SAP HANA includes a selection of standard Java libraries, which are available for download either on the Java Maven Central repository or, for customers and partners who have the appropriate access authorization, from the SAP Service Marketplace (SMP). On SAP Service Marketplace, search for the software component named `XS_JAVA`, which is an archive that contains the packages listed in the following table:

> → Tip
>
> The most recent versions of these libraries are available on Maven Central, which is now the default and recommended way to consume the Java libraries provided by SAP. For more details of the library's features and contents, see the `README` file in the corresponding package or, where available, the JavaDoc reference guide. See *Related Information* below for more information about the Maven Central repository.

SAP JAVA Library Contents

Library	XS_JAVA SPS05	Maven Central	Description
`xs-env`	Yes	Yes	A utility for setting up and accessing XS advanced environment variables and services
`audit-java-client-api`	Yes		A client that enables applications to use the XS advanced audit-log service
`java-container-security` *	Yes		A convenient Java wrapper for the security context of an authentication
`java-security`		Yes	For Java applications as replacement for `java-container-security` above
`spring-xsuaa`		Yes	For Spring Boot applications as replacement for `java-container-security` above
`java-js-client`	Yes	Yes	A convenient Java wrapper for the Job Scheduler REST API end points

Library	XS_JAVA SPS05	Maven Central	Description
`jds`	Yes	Yes	A set of tools that facilitate the consumption of Core Data Services (CDS) artifacts in Java applications
`gateway.v4`	Yes	Yes	A client library that enables you to set up an OData (v4) service for use by a Java application in an XS advanced
`sap-java-hdi`	Yes	Yes	The SAP HANA Deployment Infrastructure (HDI) client library for Java applications

> **Note**
>
> As of SPS05 the Java library `java-container-security` (*) is deprecated and replaced by the open-source SAP Cloud Platform Java security client `java-security`, which is available on GitHub and on Maven Central.

xs-env

A utility that enables easy setup of and access to XS advanced environment variables and services.

Objects of type `VcapApplication` store information about applications and objects of type `VcapServices` store information about service instances.

The following example shows how to create `VcapApplication` from the `<VCAP_APPLICATION>` system property:

> **Sample Code**

```
public static VcapApplication fromSystemProperty() throws
IllegalArgumentException {
       return VcapApplication.from(System.getProperty(VCAP_APPLICATION));
}
```

If the system property is not set, an empty instance is returned. To check if the system property exists, use `getProperty()`, as illustrated in the following example:

```
System.getProperty(VCAP_APPLICATION) != null;
```

The following example shows how to create `VcapApplication` from the `<VCAP_APPLICATION>` environment variable.

> **Sample Code**

```
public static VcapApplication fromEnvironment() throws
IllegalArgumentException {
       return VcapApplication.from(System.getenv(VCAP_APPLICATION));
}
```

If the environment variable is not set, an empty instance is returned. To check if the environment variable exists, use `getenv()`, as illustrated in the following example:

```
System.getenv().containsKey(VCAP_APPLICATION);
```

The following example shows how to create `VcapServices` from the `<VCAP_SERVICES>` system property:

Sample Code

```
public static VcapServices fromSystemProperty() throws
IllegalArgumentException {
       return VcapServices.from(System.getProperty(VCAP_SERVICES));
}
```

If the system property is not set, an empty instance is returned. To check if the system property exists, use `getProperty()`, as illustrated in the following example:

```
System.getProperty(VCAP_SERVICES) != null;
```

The following example shows how to create `VcapServices` from the `<VCAP_SERVICES>` environment variable:

Sample Code

```
public static VcapServices fromEnvironment() throws IllegalArgumentException {
       return VcapServices.from(System.getenv(VCAP_SERVICES));
}
```

If the environment variable is not set, an empty instance is returned. To check if the environment variable exists, use `getenv()`, as illustrated in the following example:

```
System.getenv().containsKey(VCAP_APPLICATION);
```

audit-java-client-api

SAP HANA includes a client for the SAP HANA XS advanced audit-log service that is used by Web applications to send audit-log messages to a remote persistent storage.

You can configure an application to use the audit-log service to write entries detailing important events in the audit log. For that purpose, you configure an audit log API and then bind the application to the service. To use the client, the following high-level steps are necessary:

1. Include the audit-log API in your Maven project.

   Sample Code

   Define the dependency in `pom.xml`

   ```
   <dependency>
       <groupId>com.sap.xs.auditlog</groupId>
       <artifactId>audit-java-client-api</artifactId>
       <version>1.2.3</version>
       <scope>provided</scope>
   ```

```
</dependency>
```

2. Declare the resource.
   - Integration in Tomcat:

     Sample Code

     Add a new resource in `META-INF/context.xml`.

     ```
     <?xml version='1.0' encoding='utf-8'?>
     <Context>
      <Resource name="audit"
     auth="Container" type="com.sap.xs.audit.api.AuditLogMessageFactory"
     factory="com.sap.xs.XSObjectFactory" singleton="true" />
     </Context>
     ```

   - Integration in TomEE:

     Sample Code

     Add a new resource in `WEB-INF/resources.xml`

     ```
     <?xml version='1.0' encoding='utf-8'?>
     <resources>
      <Resource id="audit"
     type="com.sap.xs.audit.api.AuditLogMessageFactory"
     provider="xs.openejb:XS Audit Log Message Factory Provider"/>
     </resources>
     ```

3. Enable access to the audit-log message factory.
   - Java Naming and Directory Interface (JNDI) look-up:

     ```
     Context ctx = new InitialContext();
     AuditLogMessageFactory auditlogMesageFactory = (AuditLogMessageFactory)
     ctx.lookup("java:comp/env/audit");
     ```

   - Resource injection:

     ```
     @Resource(name="audit")
     private AuditLogMessageFactory mesageFactoryInj;
     ```

Tip

For more information about setting up the client to use the audit-log API, see *Related Information*.

java-container-security

A client library that provides a convenient Java wrapper for the security context of an authentication.

- Add the security library to the `pom.xml`.
- Edit the `spring-security.xml` file to secure the resource end-points and requests.

To maintain the details specified in the Spring security configuration (`spring-security.xml`), for example:

Sample Code

`spring-security.xml`

```
<?xml version="1.0" encoding="UTF-8" ?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:oauth="http://www.springframework.org/schema/security/oauth2"
    xmlns:sec="http://www.springframework.org/schema/security"
...
     <!-- protect secure resource endpoints ================ -->
     <sec:http pattern="/**" create-session="never"
             entry-point-ref="oauthAuthenticationEntryPoint"
             access-decision-manager-ref="accessDecisionManager"
             authentication-manager-ref="authenticationManager"
             use-expressions="true">
             <sec:anonymous enabled="false" />
             <sec:intercept-url pattern="/rest/addressbook/deletedata"
                  access="#oauth2.hasScope('${xs.appname}.Delete')"
method="GET" />
             <sec:intercept-url pattern="/**" access="isAuthenticated()"
method="GET" />
             <sec:custom-filter ref="resourceServerFilter"
before="PRE_AUTH_FILTER" />
             <sec:access-denied-handler ref="oauthAccessDeniedHandler" />
     </sec:http>
...
     <!-- secure the requests ============================== -->
     <oauth:resource-server id="resourceServerFilter"
             resource-id="springsec" token-services-
ref="offlineTokenServices" />
     <bean id="offlineTokenServices"
             class="com.sap.xs2.security.commons.SAPOfflineTokenServices">
      <property name="verificationKey" value="${xs.uaa.verificationkey}" />
      <property name="trustedClientId" value="${xs.uaa.clientid}" />
      <property name="trustedIdentityZone" value="${xs.uaa.identityzone}" />
     </bean>
...
</beans>
```

For example, to enable a Java application to obtain user information related to a security context in XS advanced:

Sample Code

```
try {
    UserInfo userInfo = SecurityContext.getUserInfo();
    logonName = userInfo.getLogonName();
    email = userInfo.getEmail();
    givenName = userInfo.getJsonValue("givenName");
    familyName = userInfo.getJsonValue("familyName");
} catch (UserInfoException e) {
    // handle exception
}
```

java-security

A client library that enables token validation for Java applications in the context of authentication requests on the SAP HANA XS advanced and Cloud Foundry run-time environments and with the XS advanced User Account and Authentication service (XSUAA).

> → Tip
>
> The `java-security` client library is only available on Maven Central and is intended as a replacement for `java-container-security` for Java applications. For more information see *SAP CP Java Security Library (GitHub)* in *Related Information* below.

Configuration

The following example shows how to set up the Maven dependencies for the `java-security` library.

> ‹≡› Sample Code

```
<dependency>
    <groupId>com.sap.cloud.security</groupId>
    <artifactId>java-security</artifactId>
    <version>2.7.5</version>
</dependency>
<dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpclient</artifactId>
</dependency>
```

Usage

> ℹ Note
>
> Changes have been made to the Java client libraries `java-security` and `spring-xsuaa`. If you plan to use these libraries, please note that it is no longer possible to use the parameter `SAP_JWT_TRUST_ACL` to specify a dedicated access control list (ACL) for JWT tokens. Changes also apply regarding the granting of security scopes, which are defined and granted in the application security descriptor (`xs-security.json`). For example, if a business application A wants to call another application B, it is now mandatory that application B grants at least one scope to the calling business application A. For more information about security scopes, see the *Application Security Descriptor Configuration Syntax* in *Related Information* below.

1. Load the service configuration.

   ```
   OAuth2ServiceConfiguration serviceConfig =
   Environments.getCurrent().getXsuaaConfiguration();
   ```

   > → Tip
   >
   > `Environments` detects the following environments automatically: Cloud Foundry or Kubernetes.

2. Set up the validators.
   Configure the `JwtValidatorBuilder` once using the service configuration defined in the previous step

   ```
   CombiningValidator<Token> validators =
   JwtValidatorBuilder.getInstance(serviceConfig).build();
   ```

→ Tip

By default `JwtValidatorBuilder` builds a `CombiningValidator`. For the Signature validation, it needs to fetch the JSON Web Token Keys (`jwks`) from the OAuth server. In case the token does not provide a `jku` header parameter, the Open-ID Provider Configuration is also requested from the OAuth Server to determine the `jwks_uri`. The Apache Rest client can be customized via the `JwtValidatorBuilder` builder.

3. Create a token object.
   This step decodes an encoded JSON Web Token (JWT) and parses both the JSON header and the payload. The Token interface provides a simple access to its JWT header parameters and its claims. You can find the claim constants in the (`TokenClaims`) class.

```
String authorizationHeader = "Bearer eyJhbGciOiJGUzI1NiJ2.eyJhh...";
Token token = new XsuaaToken(authorizationHeader);
```

4. Validate token to check authentication.

```
ValidationResult result = validators.validate(token);
if(result.isErroneous()) {
   logger.warn("User is not authenticated: " + result.getErrorDescription());
}
```

5. Cache the validated token (thread-locally).

```
SecurityContext.setToken(token);
```

6. Retrieve information from the token.

```
Token token = SecurityContext.getToken();
String email = token.getClaimAsString(TokenClaims.EMAIL);
List<String> scopes = token.getClaimAsStringList(TokenClaims.XSUAA.SCOPES);
java.security.Principal principal = token.getPrincipal();
Instant expiredAt = token.getExpiration();
...
```

7. Display information stored in `VCAP_SERVICES`
   If you need further details from the `VCAP_SERVICES` system environment variable, which are not exposed by `OAuth2ServiceConfiguration` interface you can use the `DefaultJsonObject` class for JSON parsing.

```
String vcapServices = System.getenv(CFConstants.VCAP_SERVICES);
JsonObject serviceJsonObject = new
DefaultJsonObject(vcapServices).getJsonObjects(Service.XSUAA.getCFName()).get(
0);
Map<String, String> xsuaaConfigMap = serviceJsonObject.getKeyValueMap();
Map<String, String> credentialsMap =
serviceJsonObject.getJsonObject(CFConstants.CREDENTIALS).getKeyValueMap();
```

ℹ Note

For more information about the `java-security` library, see *SAP CP Java Security Library (GitHub)* in *Related Information* below.

spring-xsuaa

A client library that enables token validation for Java applications in the context of authentication requests.

> → Tip
>
> The client `spring-xsuaa` library is only available on Maven Central and is intended as a replacement for `java-container-security` for Spring Boot applications. or more information see *SAP CP Spring XSUAA Library (GitHub)* in *Related Information* below.

Configuration

The following example shows how to set up the Maven dependencies for the `spring-xsuaa` library.

> Sample Code

```
<dependency> <!-- includes spring-security-oauth2 -->
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-oauth2-jose</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-oauth2-resource-server</artifactId>
</dependency>
<dependency>
    <groupId>com.sap.cloud.security.xsuaa</groupId>
    <artifactId>spring-xsuaa</artifactId> <version>2.7.5</version>
</dependency>
<dependency> <!-- new with version 1.5.0 -->
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-to-slf4j</artifactId>
    <version>2.11.2</version>
</dependency>
```

You can also use the `xsuaa-spring-boot-starter` to set up automatic configuration, as shown in the following example:

```
<dependency>
    <groupId>com.sap.cloud.security.xsuaa</groupId>
    <artifactId>xsuaa-spring-boot-starter</artifactId>
    <version>2.7.5</version>
</dependency>
```

Usage

Spring Boot Applications can use cloud the `spring-xsuaa` client library to perform the offline token validation of a JSON Web Token (JWT). The signature is validated using the token keys received from the XSUAA service.

> i Note
>
> Changes have been made to the Java client libraries `java-security` and `spring-xsuaa`. If you plan to use these libraries, please note that it is no longer possible to use the parameter `SAP_JWT_TRUST_ACL` to specify a dedicated access control list (ACL) for JWT tokens. Changes also apply regarding the granting of security scopes, which are defined and granted in the application security descriptor (`xs-`

`security.json`). For example, if a business application A wants to call another application B, it is now mandatory that application B grants at least one scope to the calling business application A. For more information about security scopes, see the *Application Security Descriptor Configuration Syntax* in *Related Information* below.

To configure Spring security for your Spring Boot application, perform the following steps:

1. Access user or token information.
   In the Java coding, use the token to extract user information:

```
@GetMapping("/getGivenName")
public String getGivenName(@AuthenticationPrincipal Token token) {
    return token.getGivenName(); }
```

2. Check authorization within a method.

```
@GetMapping(@AuthenticationPrincipal Token token)
public ResponseEntity<YourDto> readAll() {
     if (!token.getAuthorities().contains(new
SimpleGrantedAuthority("Display"))) {
         throw new NotAuthorizedException("This operation requires
\"Display\" scope");
     }
}
...
@ResponseStatus(HttpStatus.FORBIDDEN) //set status code to '403'
class NotAuthorizedException extends RuntimeException {
    public NotAuthorizedException(String message) {
        super(message);
    }
}
```

3. Check authorization at the **method** level.
   Spring Security supports authorization semantics at the method level. As a prerequisite, you need to enable global Method Security:

```
@GetMapping("/hello-token")
@PreAuthorize("hasAuthority('Display')")
public Map<String, String> message() {
    ...
}
```

> **i** Note
>
> For more information, see *SAP CP Spring XSUAA Security Library* in *Related Information* below.

java-js-client

This client library is a Java wrapper that enables convenient access to the XS advanced Job Scheduler's REST API end points. The Job Scheduler service enables consumers to create, schedule, manage, and monitor application tasks within the SAP HANA XS advanced run-time, both on-premise and in Cloud Foundry.

Usage

The following code shows how to obtain an instance of the `SchedulerClient` from the factory:

```
Client client = SchedulerFactory.getSchedulerClient();
```

You must set the `jobscheduler` credentials to the client, as illustrated in the following code example:

```
client.setApi("https://host.acme.com:4242");
client.setUserName("a9a32ac0-5522-11e5-adbd-13adbfaf2ac7");
client.setPassword("6081473c-6563-43ca-a8dd-61abb987330f");
```

To enable the client to consume the scheduler API, you must create an instance of the `SchedulerManager`, as illustrated in the following example:

```
SchedulerManager schedulerManager = client.getSchedulerManager();
```

The following example shows how to **update** a scheduled job:

```
newJob.setActive(false);
newJob.setUser("test");
newJob.setPassword("Toor1234");
try {
    newJob = (Job) schedulerManager.update(newJob);
    slf4jLogger.info("Job is updated. Now, the job is " + (newJob.getActive() ?
"active" : "inactive"));
} catch (SchedulerException e) {
    slf4jLogger.error("Failed to update the job");
}
```

The following example shows how to **add** a new schedule to an existing job and **update** a schedule:

```
// Create new schedule entity
Schedule newSchedule = new Schedule();
newSchedule.setDescription("Ping Google every 20 minutes using xscron");
newSchedule.setCron("* * * * * */20 *");
newSchedule.setJob(newJob);
try {
    Schedule createdSchedule = (Schedule) schedulerManager.create(newSchedule);
    slf4jLogger.info("New schedule is created with id " +
createdSchedule.getId());
    // --------------Update schedule----------
    createdSchedule.setStartTime(startTime);
    createdSchedule.setEndTime(endTime);
    Schedule schedule2 = (Schedule) schedulerManager.update(createdSchedule);
    slf4jLogger.info("schedule updated with start time "
      + schedule2.getStartTime().getDate() + " end time "
      + schedule2.getEndTime().getDate());
    newJob = schedulerManager.searchJobById(newJob.getId());
    slf4jLogger.info("Now, the number of schedules is " +
newJob.getSchedules().size());
} catch (SchedulerException e) {
    slf4jLogger.error("Failed to update the schedule");
}
```

The following example shows how to **activate** or **deactivate** all schedules configured for a job:

```
try {
    schedulerManager.activateAllSchedules(newJob);
    Collection<Schedule> schedules = newJob.getSchedules();
    boolean scheduleActive = false;
    for (Iterator<Schedule> scheduleIterator = schedules.iterator();
scheduleIterator.hasNext();) {
```

```
         Schedule schedule2 = (Schedule) scheduleIterator.next();
         scheduleActive = scheduleActive & schedule2.getActive();
    }
    slf4jLogger.info("All the schedules are now " + (scheduleActive == true ?
"active" : "inactive"));
} catch (SchedulerException e1) {
    slf4jLogger.error("Failed to activate the schedules");
}
try {
    schedulerManager.deactivateAllSchedules(newJob);
    newJob = schedulerManager.searchJobById("90009");
    slf4jLogger.info("All schedules are deleted. The number of schedules is " +
newJob.getSchedules().size());
} catch (SchedulerException e1) {
    slf4jLogger.error("Failed to deactivate the schedules");
}
```

The following example shows how to **delete** a scheduled job:

```
try {
    schedulerManager.delete(newJob);
    slf4jLogger.info("The job is deleted");
} catch (SchedulerException e) {
    slf4jLogger.error("Failed to delete the job");
}
```

The following example shows how to **update** a log:

```
    Log log = new Log();
    log.setJobId("1");
    log.setScheduleId("da722278-0a22-440c-a4d6-c599b77617f4");
    log.setId("11765A573A618A79E10000000A61A17B");
    log.setMessage("New log updated");
    log.setSuccess(true);
    Log newLog = new Log();
try {
    newLog = (Log) schedulerManager.update(log);
    slf4jLogger.info("Log is Updated ");
} catch (SchedulerException e2) {
    slf4jLogger.error("Failed to update the log");
}
```

jds

The Java Data Services (JDS) are a set of tools that facilitate the consumption of Core Data Services artifacts in Java applications. More specifically, JDS maps existing CDS data models to corresponding Java Persistence API (JPA) 2.1 data models that may be consumed by Java applications using EclipseLink or Hibernate.

Usage

Run the tool from the command-line in a command shell:

```
jds> java -jar jds-import-1.0.1-full.jar help
usage: java -jar importer-x.y.z-full.jar <command> <parameters>
Commands:
        help - displays this help text
        userguide - display the JSON user guide
        display - display available entities in namespace
        create - create Java classes for entities
```

```
        <config file>.json - process instructions in document (advanced)
```

> → Tip
>
> For more information about using Java Data Services, see *Related Information*.

Support and Limitations

The current release of JDS supports basic CDS entities, CDS types, CDS views, and CDS one-to-one associations. All imported CDS entities can be used with any available JPA technology, including queries using JPQL and the Criteria API.

Note the following limitations:

- nokey entity
  Entities without keys are not supported by JPA:
  - CDS artifact is created.
  - `JpaRepresentationEntity` is not created
  - Java file is not created
- associations to non-primary key fields
  Foreign key associations must not contain foreign keys which are not primary keys:
  - CDS artifact is created.
  - `JpaRepresentationEntity` is created **without** the association
  - Java file is created **without** the association.

gateway.v4

This is client library that enables you to set up an OData (v4) service for use by a Java application in an XS advanced.

An OData v4 service for use with an XS advanced Java application is described in a design-time OData artifact and referenced in the corresponding CDS document (by means of an @OData.publish annotation) that exposes the specified tables or entities in the annotated context. The definition of your OData service should be placed in the java/odata/v4/ folder of your Java application project module. The following example shows the project structure for an XS advanced Java application which consumes an OData version 4.0 service.

> → Tip
>
> For more information about using Java Data Services, see *Related Information*.

sap-java-hdi

The `sap-java-hdi` client library enables Java applications to access the SAP HANA Deployment Infrastructure (HDI). Currently, the library is available in the following versions:

- `sap-java-hdi` version 1.0
  Released with SAP HANA 2.0 SPS 01 and still available in SAP HANA 2.0 SPS 02.

> **i Note**
>
> `sap-java-hdi` version 1.0 is considered feature complete and will not receive any further enhancements in future releases.

- `sap-java-hdi` version 2.0
  Released with SAP HANA 2.0 SPS 02

  > **→ Tip**
  >
  > It is recommended to use version 2.0 of `sap-java-hdi`.

sap-java-hdi (version 1.0)

Version 1.0 of the Java client library for HDI provides the following **methods** to enable access to HDI functionality:

> **→ Tip**
>
> For reasons of support and compatibility, it is recommended to use version 2.0 of `sap-java-hdi`.

- `connect`
- `disconnect`
- `configureDI`
- `configureDIParameters`
- `createContainer`
- `dropContainer`
- `configureContainer`
- `configureContainerParameters`
- `listLibraries`
- `listDeployed`
- `configureLibraries`
- `listConfiguredLibraries`
- `status`
- `read`
- `readDeployed`
- `write`
- `delete`
- `make`
- `makeAsync`
- `grantContainerApiPrivileges`
- `grantContainerApiPrivilegesWithGrantOption`
- `revokeContainerApiPrivileges`
- `grantContainerSchemaPrivileges`
- `revokeContainerSchemaPrivileges`
- `grantContainerSchemaRoles`
- `revokeContainerSchemaRoles`

The HDI class provides two constructors to create a new HDI instance. The first can be used to execute non-container-specific tasks:

Sample Code

```
HDI(String host, String port, String dbName, String user, String password)
```

The second can be used to enable operations on a container, as illustrated in the following example:

Sample Code

```
HDI(String host, String port, String dbName, String user, String password,
String container)
```

An HDI instance created with the first constructor can be upgraded by using the `setContainer()` command:

Sample Code

```
setContainer(String container)
```

The connection to the database is opened and closed with the commands "`connect()`" and "`disconnect()`".

sap-java-hdi (version 2.0)

Version 2.0 of the SAP HANA DI (HDI) client library for Java applications wraps the HDI SQL APIs in Java classes and methods. The following main classes are available:

HDI JAVA Classes

Class	Description
`HDI`	Provides access to the `_SYS_DI` API
`Container`	Provides access to the API of a specific HDI container
`ContainerGroup`	Provides access to the API of a specific HDI container group

→ Tip

For reasons of support and compatibility, it is recommended to use version 2.0 of `sap-java-hdi`.

To use all APIs, the connected user must have the following privileges:

- `EXECUTE`
  The `EXECUTE` privilege is required for the corresponding SQL procedures.
- `SELECT`
  The `SELECT` privilege is required for all table types in `SYS_DI` (`TT*`).

i Note

For all the examples used in this section to illustrate some of the most common usage scenarios for the HDI Java client library `sap-java-hdi`, it is assumed that the `hdiUser` has the privileges required to access

> `_SYS_DI`. In addition, some APIs require a minimum SAP HANA version. For more information about which SAP HANA version is required by a specific class or method, see the `sap-java-hdi` JavaDoc reference.

The following example shows how to use `sap-java-hdi` to instantiate an HDI API wrapper object for the database "`DB1`" in the SAP HANA instance running on a host named "`someHost`" on SQL port `30015` with the user "`hdiUser`", the password "`Abcd1234`" and, in addition, using the user's default schema for temporary tables.

```
HDI hdi = new HDI("someHost", "30015", "DB1", "hdiUser", "Abcd123", "hdiUser");
```

To create a new container group named "`someGroup`", use the following code:

> → Tip
>
> The last argument ("`null`" in this case) is an optional list of key-value parameters for the call.

```
ResultTuple rt = hdi.createContainerGroup("someGroup", null);
System.out.println("Return code: " + rt.getRC());
System.out.println("Messages: " + rt.getMessages());
```

The following example shows how to grant the default container-group administrator privileges for "`someGroup`" to the user "`hdiUser`":

```
List<APIPrivilege> groupAdminPrivileges =
hdi.getDefaultContainerGroupAdminPrivileges("", "hdiUser");
rt = hdi.grantContainerGroupApiPrivileges("someGroup", groupAdminPrivileges,
null);
```

The following code shows how to use the `sap-java-hdi` to instantiate a container-group-API wrapper object for "`someGroup`" in the database "`DB1`" in the SAP HANA instance running on host "`someHost`" with SQL port `30015` with user `hdiUser`, password `Abcd1234` and using the user's default schema for temporary tables:

```
ContainerGroup group = new ContainerGroup("someGroup", "someHost", "30015",
"DB1", "hdiUser", "Abcd123", "hdiUser");
```

The following example shows how to create a new container named "`someContainer`" with the Java HDI API and place it in the container group "`someGroup`" created in the previous example:

> → Tip
>
> The API object "`group`" used in this example was created for the container group "`someGroup`"(`= new ContainerGroup("someGroup", [...])`)

```
rt = group.createContainer("someContainer", null);
```

To grant the default container-administrator privileges for the HDI container "`someContainer`" to the HDI user "`hdiUser`", consider the following functions:

```
List<APIPrivilege> containerAdminPrivileges =
group.getDefaultContainerAdminPrivileges("", "hdiUser");
rt = group.grantContainerApiPrivileges("someContainer",
containerAdminPrivileges, null);
```

The following functions show how to use the HDI container API to instantiate a container-API wrapper object for `someContainer` in database `DB1` in the SAP HANA instance running on host `someHost` with SQL port

`30015` with user `hdiUser`, password `Abcd1234`, and using the user's default schema to store temporary tables:

```
Container container = new Container("someContainer", "someHost", "30015", "DB1",
"hdiUser", "Abcd123", "hdiUser");
```

To list the plug-in libraries configured in `someContainer`, consider the following functions:

```
LibraryInformationResultTuple lrt = container.listConfiguredLibraries(null);
System.out.println("Configured libraries: " + lrt.getResults());
```

Related Information

Download and Consume Java Libraries [page 719]
The SAP HANA XS Advanced Java Run Time [page 716]
Configure an Application to Use the Audit Log Service [page 747]
Enable and Use Java Data Services [page 771]
Defining OData v4 Services for XS Advanced Java Applications [page 580]
SAP CP Java Security Library (GitHub)
SAP CP Spring XSUAA Library (GitHub)
Maven Central Repository
Configure Authentication and Authorization (java-security, spring-xsuaa) [page 754]
Application Security Descriptor Configuration Syntax [page 999]

7.2.2 Tutorials: Setting Up the Java Run-Time Environment in XS Advanced

A selection of tutorials that show you how to set up an application to run in the XS advanced Java run-time environment.

Prerequisites

To perform the tasks described in the following tutorials, the following tools and components must be available:

- SAP HANA XS advanced run time must be installed on your SAP HANA system
- The SAP HANA XS CLI client must be installed on your development machine.

> → Tip
>
> The package containing the XS CLI client tools (`xs.onpremise.runtime.client_<platform>-<version>.zip`) can be downloaded from the SAP HANA server, from the installation DVD, or from the SAP support portal.

- You must have valid user logon credentials for the target system where XS advanced is running

  Change Application Settings for the XS Advanced Java Run Time [page 737]
  You can change the run-time settings in your XS advanced Java application's manifest file.

  Configure a Java Application for Logs and Traces [page 744]
  Configure the collection of log and trace messages generated by a Java application in XS advanced.

  Configure an Application to Use the Audit Log Service [page 747]
  You can configure an application to use the audit-log service to write entries detailing important events in the audit log. For that purpose, you configure an audit log API and then bind the application to the service.

  Configure Authentication and Authorization [page 754]
  You configure authentication and authorization by using the integrated container authentication provided with the SAP Java build pack, the Spring security library, or open-source Cloud security libraries.

  Configure a Database Connection [page 762]
  You can configure your application to use a database connection so that the application can persist its data.

  Debug an XS Advanced Java Application [page 770]
  You can use this procedure only if you can restart your Java application or the application has not yet been pushed.

  Enable and Use Java Data Services [page 771]
  Enable Java Data Services, a native Java client for using Core Data Services functionality in the XS advanced Java run time.

  Run a Java Main Application [page 775]
  You can create a Java application that starts its own run time.

Related Information

Working with the XS Advanced Command-Line Client [page 75]
Getting Started with Application Development in XS Advanced [page 27]
SAP Note SAP Note 2242468: Setting up the XS Advanced CLI

7.2.2.1 Change Application Settings for the XS Advanced Java Run Time

You can change the run-time settings in your XS advanced Java application's manifest file.

Set the Java Virtual Machine (JVM)

Context

You can specify the Java Virtual Machine (JVM) that your application needs by setting the `<JBP_CONFIG_COMPONENTS>` environment variable (for example, in the application's `manifest.yml` file) before pushing the application to the run-time environment. Depending on the Java version your application requires, choose one of the options described in the following sections.

Using Java 8 with the SAP JVM JRE

If your application uses Java 8 (provided by the SAP JVM) and a JRE is sufficient, use the following configuration in the application's `manifest.yml` file:

```
---
applications:
- name: <app-name>
  ...
  env:
    JBP_CONFIG_COMPONENTS: '{ "jres": ["com.sap.xs.java.buildpack.jre.SAPJVM"] }'
  ...
```

If the `<JBP_CONFIG_COMPONENTS>` environment variable is not set, the default JVM for the Java application is the JRE.

Using Java 8 with the SAP JVM JDK

If your application uses Java 8 (provided by the SAP JVM) and requires a JDK, use the following configuration in the application's `manifest.yml` file:

```
---
applications:
- name: <app-name>
  ...
  env:
    JBP_CONFIG_COMPONENTS: '{ "jres":
["com.sap.xs.java.buildpack.jdk.SAPJVMJDK"] }'
  ...
```

Using Java 11 or Java 17 with the SapMachine JRE

If your application uses Java 11 or Java 17 provided by the SapMachine, and a JRE is sufficient, use the following configuration in the application's `manifest.yml` file:

```
---
applications:
- name: <app-name>
  ...
```

```
  env:
    JBP_CONFIG_COMPONENTS: '{ "jres":
["com.sap.xs.java.buildpack.jre.SAPMachineJRE"] }'
    JBP_CONFIG_SAP_MACHINE_JRE: '{ "version": "17.+" }'
  ...
```

Use the environment variable `<JBP_CONFIG_SAP_MACHINE_JRE>` to configure the Java version. You can specify a strict version or use the '+' wildcard to define a major Java version in general, as shown in the example above. If you do not specify this property, the default is Java 11.

Using Java 11 or Java 17 with the SapMachine JDK

If your application uses Java 11 or Java 17 provided by the SapMachine and requires a JDK, use the following configuration in the application's `manifest.yml` file:

```
---
applications:
- name: <app-name>
  ...
  env:
    JBP_CONFIG_COMPONENTS: '{ "jres":
["com.sap.xs.java.buildpack.jdk.SAPMachineJDK"] }'
    JBP_CONFIG_SAP_MACHINE_JDK: '{ "version": "17.+" }'
  ...
```

Use the environment variable `<JBP_CONFIG_SAP_MACHINE_JDK>` to configure the Java version. You can specify a strict version or use the '+' wildcard to define a major Java version in general, as shown in the example above. If you do not specify this property, the default is Java 11.

Set the TomEE or TomEE 7 Run Time

Context

The default Java run-time environment in XS advanced is “Tomcat”. However, you can change your desired Java run-time environment in XS advanced to “TomEE” or “TomEE 7” by setting the `<TARGET_RUNTIME>` environment variable in the Java application's `manifest.yml`, as illustrated in the following examples:

> ‹≡› Code Syntax
>
> For TomEE
>
> ```
> env:
>     TARGET_RUNTIME: tomee
> ```

> ℹ Note
>
> SAP plans to remove the outdated TomEE 1.7 version from the SAP Java Buildpack for SAP HANA XS Advanced. Although TomEE 1.7 will continue to be supported until the end of the deprecation period, it is recommended to migrate to TomEE 7 as soon as possible, as described in the *TomEE Migration Guide* listed in *Related Information* below and in SAP Note 3208765. TomEE 7 is already included in previous versions of the SAP Java Buildpack for SAP HANA XS Advanced.

Code Syntax

For TomEE 7

```
env:
    TARGET_RUNTIME: tomee7
```

Configure Memory Sizes

Context

The HEAP, METASPACE and STACK memory types are configured with a corresponding default size. However, the default values can also be customized, as illustrated in the following example.

Procedure

- Configure the memory sizes during the staging phase.

  Provide the `<JBP_CONFIG_SAPJVM>` (or `<JBP_CONFIG_SAP_MACHINE_[JRE|JDK]>` if you use SapMachine) environment variable in the `manifest.yml` file of the application and then stage the application.

  Example

  Sample `manifest.yml`

  ```
  ---
  applications:
  - name: <app-name>
    memory: 512M
  ...
    env:
      JBP_CONFIG_SAPJVM:  '{ "memory_calculator": { "memory_heuristics":
  { "heap": 85, "stack": 10 } } }'
  ```

  For more information about memory types for the XS advanced Java run-time environment, see *Memory Size Options* in *Related Information*.
- Configure the memory sizes during the application run time.

  This configuration can be changed multiple times after the application is staged and does not require re-staging.

  a. Open the command prompt.
  b. Set custom weights, sizes, and initials.

     For more information, see *Memory Size Options* in *Related Links*.
  c. Restart the application.

Configure Custom JVM Properties

Context

You can add custom JVM properties as this configuration is done once and can be changed only when the application is re-staged.

For more information about the Java run-time options in XS advanced, see *Related Links*.

Procedure

- Use an environment variable in the manifest file.

  ### Sample Code

  Excerpt from a Java `manifest.yml` file

  ```
  ---
  applications:
  - name: <app-name>
    memory: 512M
  ...
    env:
      JBP_CONFIG_JAVA_OPTS: '{ "java_opts": "-Dlog4j2.debug=\"true\"
  -DcustomProperty=\"<custom_value>\"" }'
  ```

  ### Note

  A single quote is used to enclose the `<JBP_CONFIG_JAVA_OPTS>` string in the YML file because the string can contain any one (or combination) of the following characters: *:*, *{*, *}*, *[*, *]*, *,*, *&*, ***, *#*, *?*, *|*, *-*, *<*, *>*, *=*, *!*, *%*, *@*, **.

- Set the environment variable with a the `xs set-env` command.

  If the application is already available on SAP HANA XS advanced and the application developer wants to fine-tune the JVM additionally with an additional or modified property, a new value for the `<JBP_CONFIG_JAVA_OPTS>` environment variable can be specified with the `xs set-env` command. The new values will overwrite the default (or set) values in the corresponding `buildpack` the next time the application is staged.

  ### Example

  ```
  xs set-env myapp JBP_CONFIG_JAVA_OPTS '{ "java_opts": "-
  Dlog4j2.debug=\"true\" -DcustomProperty=\"<custom_value>\"" }'
  ```

  ### Note

  When the Java options are specified on the command line with the `xs set-env` command, the string defining the new values must be enclosed in double quotes, for example, `"<New-config_values>"`.

Using an Agent

Context

You can use any agent with the SAP Java Build pack. The agent must be included in the `.jar` or `.war` archive of your application. The SAP Java Build pack extracts the agent when the application is deployed.

Procedure

- To use a Java agent with the SAP Java Build pack, set the `<JBP_CONFIG_JAVA_OPTS>` environment variable using the -javaagent option as shown in the following example:

  ```
  env:
      JBP_CONFIG_JAVA_OPTS: '{ "java_opts": "-javaagent:<PathToYourAgent>" }'
  ```

  > **i** Note
  >
  > The Java agent is platform-agnostic, but must be compatible to the version of the JVM you are using.

- You can also use a native agent. Since a native agent is a dynamic library, it must be compatible with the architecture of the platform. For Cloud Foundry, this is Linux x86_64. As the application developer you must make sure that the correct agent is used and apply updates whenever they are needed. To use a native agent, set the `<JBP_CONFIG_JAVA_OPTS>` using the -agentpath option as shown in the following example:

  ```
  env:
     JBP_CONFIG_JAVA_OPTS: '{ "java_opts": "-agentpath:<PathToYourAgent>" }'
  ```

Examples

Context

The following example shows how to configure Java 17 with SapMachine JRE, and custom memory calculator and JVM options:

```
---
applications:
- name: <app-name>
  memory: 512M
...
  env:
     JBP_CONFIG_COMPONENTS: '{ "jres":
["com.sap.xs.java.buildpack.jre.SAPMachineJRE"] }'
     JBP_CONFIG_SAP_MACHINE_JRE: '{ "version": "17.+", "memory_calculator":
{ "memory_heuristics": { "heap": 85, "stack": 10 } } }'
     JBP_CONFIG_JAVA_OPTS: '{ "java_opts": "-Dlog4j2.debug=\"true\"
-DcustomProperty=\"<custom_value>\" -javaagent:<PathToYourAgent>" }'
```

Related Information

Memory Size Options [page 742]
Java Memory Options [page 744]
TomEE Migration Guide
SAP Note 3208765

7.2.2.1.1 Memory Size Options

Memory Types

There are three memory types that can be sized and for each memory type there is a command-line option, which is passed to the JVM.

- HEAP
  The initial and maximum sizes of the HEAP memory are controlled by the -Xms and -Xmx options respectively.
- METASPACE
  The initial and maximum sizes of the METASPACE memory are controlled by the -XX:MetaspaceSize and -XX:MaxMetaspaceSize options.
- STACK
  The size of the STACK is controlled by the -Xss option.

Changing Memory Settings with the Memory Calculator

The memory calculator uses calculation techniques that allocate the available memory to the memory types. The total available memory is first allocated to each memory type in proportion to its weighting (this is called 'balancing'). If the resultant size of any memory type lies outside its range, the size is constrained to the range, the constrained size is excluded from the remaining memory, and no further calculation is required for that memory type. After that, the remaining memory is balanced against the memory types that are left, and the check is repeated until no calculated memory sizes lie outside their ranges. These iterations terminate when none of the sizes of the remaining memory types are constrained by their corresponding ranges.

The memory calculator uses the following calculation techniques:

- memory calculations using absolute memory sizes
- memory calculations using relative memory weights

Setting Custom Weights, Sizes, and Initials

You can set custom values in the manifest file before the application is staged by using the following parameters:

- `memory_sizes`
  The data for this parameter specifies the absolute values for the memory properties of the SAP JVM: -Xms, -Xmx, -Xss, -XX:MaxMetaspaceSize, or -XX:MetaspaceSize.
- `memory_heuristics`
  The data for this parameter specifies the relative weights for the different memory types.
- `memory_initials`
  This parameter is used to adjust the initial values for HEAP and METASPACE types.

> ### Sample Code
>
> Setting HEAP memory to 8.5 times larger than the STACK memory.
>
> ```
>   env:
>     JBP_CONFIG_SAPJVM:  '{ "memory_calculator": { "memory_heuristics":
> { "heap": 85, "stack": 10 } } }'
> ```

Furthermore, you can change the sizes during the application run time with the `xs set-env` command, as follows:

- To set custom weights, use the JBP_CONFIG_SAPJVM_MEMORY_WEIGHTS environment variable.

  > ### Sample Code
  >
  > ```
  > xs set-env myapp JBP_CONFIG_SAPJVM_MEMORY_WEIGHTS
  > "heap:5,stack:1,metaspace:3,native:1"
  > ```

- To set custom sizes, use the JBP_CONFIG_SAPJVM_MEMORY_SIZES environment variable.

  > ### Sample Code
  >
  > ```
  > xs set-env myapp JBP_CONFIG_SAPJVM_MEMORY_SIZES "heap: 30m..400m, stack:
  > 2m.., metaspace: 10m..12m"
  > ```

- To set custom initials, use the JBP_CONFIG_SAPJVM_MEMORY_INITIALS environment variable.

  > ### Sample Code
  >
  > ```
  > xs set-env myapp JBP_CONFIG_SAPJVM_MEMORY_INITIALS "heap: 50%, metaspace:
  > 50%"
  > ```

> ### Note
>
> If you specify values for weight and size at the same time, the size values take precedence over the weight values.

Java Options

You can set the Java options by using the following parameters:

- `from_environment`
  This parameter expects a Boolean value which specifies if the value of the `<JAVA_OPTS>` environment variable must be taken into account. By default, it is set to "`false`".
- `java_opts`
  This section is used for additional Java options such as `-Xloggc:$PWD/beacon_gc.log -verbose:gc -XX:NewRatio=2`. By default, no additional options are provided.

7.2.2.2 Configure a Java Application for Logs and Traces

Configure the collection of log and trace messages generated by a Java application in XS advanced.

Prerequisites

- You have installed and configured Maven.
- You use SAP JVM 8 and your `<JAVA_HOME>` environment variable points to this location.
- You do not have any `SLF4J` and logback `JAR` files in the application.

> **i Note**
>
> The log JARs for the SLF4J and logback are already included in the XS advanced Java run-time environment; importing them in the application could cause problems during class loading.

> **i Note**
>
> It is the application's responsibility to ensure that logback is configured in a secure way in the cases when the application overrides the default logback configuration, included in the XS advanced Java run-time environment, for example the logback appenders.

Context

The recommended framework for logging is Simple Logging Facade for Java (SLF4J). To use this framework, you can create an instance of the `org.slf4j.Logger` class. To configure your XS advanced Java application to generate logs and traces, and if appropriate set the logging or tracing level, perform the following steps:

Procedure

1. Instruct Maven that the application should not package the SLF4J dependency; this dependency is already provided by the run time.

   To do this, set the dependency scope to `provided` in the file `pom.xml`:

   ```
   <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.12</version>
    <scope>provided</scope>
    </dependency>
   ```

2. Import the SLF4J classes into the application code (`org.slf4j.Logger` and `org.slf4j.LoggerFactory` libraries) and develop the log handling code.

   Example

   Ensure that `debug` and `info` log messages from the Java application appear in the log files.

   ```
   import org.slf4j.Logger;
    import org.slf4j.LoggerFactory;
    public void logging () {
                   final Logger logger =
   LoggerFactory.getLogger(Log4JServlet.class);
                   logger.debug("this is record logged through SLF4J
   param1={}, param2={}.", "value1", 1);
                   logger.info("This is slf4j info");
   }
   ```

3. If necessary, set a logging level.

   By default, the XS advanced Java application uses the logging level `ERROR`, but you can change this setting with the `xs set-logging-level` command. For more information, see *Logging Levels* in *Related Information* below.

   Note

   To enable logs to be written in the log files, the logging level of the corresponding logger needs to be set to the value that matches the severity of the log entries in the application code. For example, for the sample in the previous step, the severity of the `Log4Servlet` needs to be set to `DEBUG` or `INFO` in order to write the appropriate entry in the corresponding log or trace file.

Related Information

Installing Apache Maven
Logging Levels [page 746]

7.2.2.2.1 Logging Levels

The `Log4J` application supports the following logging levels:

Logging Levels

Logging Level	Description
ALL	All logging levels are displayed in the logs and traces.
DEBUG	This level is used for debugging purposes. When you set this level, DEBUG, INFO, WARN, ERROR and FATAL levels are displayed in the application logs.
INFO	This level is used for information purposes. When you set this level, the INFO, WARN, ERROR and FATAL levels are displayed in the application logs.
WARN	This level is used for warning purposes. When you set this level, the WARN, ERROR and FATAL levels are displayed in the application logs.
ERROR	This level is used to display errors. When you set this level, ERROR and FATAL levels are displayed in the application logs.
FATAL	This level is used to display critical errors. When you set this level, only the FATAL level is displayed in the application logs.
OFF	Use this setting to turn off logging.

Before you use the command to set a logging level, make sure that your application is configured to support the logger location. To set logging levels for a specific location, you need to use the `xs set-logging-level` command, as illustrated in the following example:

```
xs set-logging-level <applications name> <logger path> <log level>
```

For more information about the `xs set-logging-level` command, see *XS CLI: Application Management* in *Related Information*.

> **i Note**
>
> You can change your logger location and messages by editing the `Log4JServlet` class in the `Log4J` application.

The specified logging level is applied during application startup.

Related Information

XS CLI: Application Management

7.2.2.3 Configure an Application to Use the Audit Log Service

You can configure an application to use the audit-log service to write entries detailing important events in the audit log. For that purpose, you configure an audit log API and then bind the application to the service.

Obtain the Audit Log API

Context

You use an audit log API version 1 and version 2 with Web applications by means of standard Java EE mechanisms.

Procedure

1. Include the audit log API in your Maven project.

   Example

   Sample dependency in `pom.xml` for Audit Log version 1 API

   ```
   <dependency>
       <groupId>com.sap.xs.auditlog</groupId>
       <artifactId>audit-java-client-api</artifactId>
       <version>0.0.1-SNAPSHOT</version>
       <scope>provided</scope>
   </dependency>
   ```

   Example

   Sample dependency in `pom.xml` for Audit Log version 2 API

   ```
   <dependency>
       <groupId>com.sap.xs.auditlog</groupId>
       <artifactId>audit-java-client-api</artifactId>
       <version>2.0.X</version>
       <scope>provided</scope>
   </dependency>
   ```

If you are using the community Java buildpack (java-buildpack) you need to include the following libraries:

Sample Code

```
<dependency>
    <groupId>com.sap.xs.auditlog</groupId>
    <artifactId>audit-java-client-api</artifactId>
    <version>2.0.X</version>
</dependency>
<dependency>
    <groupId>com.sap.xs.auditlog</groupId>
    <artifactId>audit-java-client-impl</artifactId>
    <version>2.0.X/version>
</dependency>
<dependency>
    <groupId>com.sap.xs.java</groupId>
    <artifactId>xs-env</artifactId>
    <version>1.6.16</version>
</dependency>
<dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-api</artifactId>
    <version>1.7.25</version>
</dependency>
```

2. Declare the resource.

- Integration in Tomcat
  Add a new resource in `META-INF/context.xml`.

  Sample Code

  Audit Log version 1 API

  ```
  <?xml version='1.0' encoding='utf-8'?>
  <Context>
   <Resource name="audit"
  auth="Container" type="com.sap.xs.audit.api.AuditLogMessageFactory"
  factory="com.sap.xs.XSObjectFactory" singleton="true" />
  </Context>
  ```

  Sample Code

  Audit Log version 2 API

  ```
  <?xml version='1.0' encoding='utf-8'?>
  <Context>
   <Resource name="audit"
  auth="Container" type="com.sap.xs.audit.api.v2.AuditLogMessageFactory"
  factory="com.sap.xs.XSObjectFactory" singleton="true" />
  </Context>
  ```

- Integration in TomEE or TomEE 7
  Add a new resource in `WEB-INF/resources.xml`.

  Sample Code

  Audit Log version 1 API

  ```
  <?xml version='1.0' encoding='utf-8'?>
  <resources>
  ```

```
 <Resource id="audit"
type="com.sap.xs.audit.api.AuditLogMessageFactory"
provider="xs.openejb:XS Audit Log Message Factory Provider"/>
</resources>
```

Sample Code

Audit Log version 2 API

```
<?xml version='1.0' encoding='utf-8'?>
<resources>
 <Resource id="audit"
type="com.sap.xs.audit.api.v2.AuditLogMessageFactory"
provider="xs.openejb:XS Audit Log Message Factory Provider"/>
</resources>
```

- Integration in non Java EE Applications - for Audit Log version 2 API
  If you are using Spring or another non JavaEE framework you can retrieve the `AuditLogMessageFactory` by instantiating it without any arguments.

```
protected AuditLogMessageFactory auditLogMessageFactory;
try {
    auditLogMessageFactory = new AuditLogMessageFactoryImpl();
} catch (AuditLogException e) {
    //TODO: Handle the exception
}
```

3. Access the audit log message factory.

   - Access through Java Naming and Directory Interface (JNDI) look-up

   ```
   Context ctx = new InitialContext();
   AuditLogMessageFactory auditlogMesageFactory = (AuditLogMessageFactory)
   ctx.lookup("java:comp/env/audit");
   ```

   - Access through resource injection

   ```
   @Resource(name="audit")
   private AuditLogMessageFactory mesageFactoryInj;
   ```

Write an Audit Log Message

Prerequisites

You have installed the XS Java client libraries.

For more information about the libraries installation, see SAP Note 2283600.

Context

You use the methods of the audit log interfaces to specify what message to be displayed in the Audit Log UI.

Procedure

- Use the general interface to specify the user that triggers the audit log event.

  Sample Code

  ```
  AuditLogMessage message = new AuditLogMessage();
  message.setUser("John Smith")
  ```

  This user is applicable for all audit log categories.
- Use the audit log message factory to create a message for a specific category.

  Sample Code

  Sample message for the data read access category.

  ```
  AuditLogMessageFactory messageFactory = new AuditLogMessageFactory();
  DataAccessAuditMessage message =
  messageFactory.createDataAccessAuditMessage();
  ```
- Log an event with a transactional logic

  Note

  The transactional logic is applicable only for the configuration change events.

  Sample Code

  A message for a configuration change event that is about to happen.

  ```
  ConfigurationChangeAuditMessage message =
  messageFactory.createConfigurationChangeAuditMessage();
  ...
  message.logPrepare();
  ```
- Log a configuration change event

  Use the `AuditLogMessage`, `AuditLogMessageFactory`, `ConfigurationChangeAuditMessage`, and `TransactionalAuditLogMessage` interfaces to log a configuration change event.

  Sample Code

  Logging for event "User John Smith changed the configuration for the address of object `Library`."

  ```
  ConfigurationChangeAuditMessage message =
  messageFactory.createConfigurationChangeAuditMessage();
  message.setUser("John Smith");
  message.setObjectId("class=com.sap.example.Library”); //a string that
  could uniquely identify the object
  message.addValue("address", “Old address”, “New address”);
  message.log();
  ```
- Log a data read access event

  Use the `AuditLogMessage`, `AuditLogMessageFactory`, and `DataAccessAuditMessage` interfaces to log a data access event.

> ### Sample Code
>
> Logging for event "User John Smith read the `address` attribute of object `Library`"
>
> ```
> DataAccessAuditMessage message =
> messageFactory.createDataAccessAuditMessage();
> message.setUser("John Smith");
> message.setObjectId("class=com.sap.example.Library(id=10)"); //a string
> that could uniquely identify the object
> message.addAttribute("address", true);
> message.log();
> ```

- Log a data modification event.

  Use the `AuditLogMessage`, `AuditLogMessageFactory`, and `DataModificationAuditMessage` interfaces to log a data modification event.

  > ### Sample Code
  >
  > Logging for event "User John Smith modified the `firstname` attribute of object `Person`."
  >
  > ```
  > DataModificationAuditMessage message =
  > messageFactory.createDataModificationAuditMessage();
  > message.setUser("John Smith");
  > message.setObjectId("class=com.sap.example.Person(id=5)"); //a string that
  > could uniquely identify the object
  > message.addAttribute("firstname", true);
  > ```

- Log a security event.

  Use the `AuditLogMessage`, `AuditLogMessageFactory`, and `SecurityEventAuditMessage` interfaces to log a security event.

  > ### Sample Code
  >
  > Logging for the event "User John Smith logged on from IP *10.20.30.40*."
  >
  > ```
  > SecurityEventAuditMessage message =
  > messageFactory.createSecurityEventAuditMessage();
  > message.setUser(“John Smith”);
  > message.setIp(“10.20.30.40”);
  > message.setData("Successful logon");
  > message.log();
  > ```

Bind the Application to the Audit Log Service

Procedure

1. Create a service instance.

   ```
   xs create-service auditlog free <my-service-instance>
   ```

2. Bind the service instance to the application.

- With the command line interface

```
xs bind-service <my-application> <my-service-instance>
```

- In `manifest.yml`

Sample Code

```
---
applications:
- name: java-auditlog-sample
  memory: 512M
  instances: 1
  path: target/java-auditlog-sample.war
  env:
    TARGET_RUNTIME: tomcat
  services:
  - <my-service-instance>
```

View Audit Logs

Context

To view the audit logs, you use an Audit Log UI available as part of the XS Advanced Admin tools.

Related Information

Audit Logs in Java Run Time [page 752]

7.2.2.3.1 Audit Logs in Java Run Time

SAP HANA XS advanced Java run time provides an audit log API that enables applications and the run time to record audit logs.

Audit Log Categories

- Configuration change log
- Data read access log
- Data modification log
- Security event log

Utility Interfaces

- `AuditLogMessage`
  A general interface that applies to all categories and used to persists the created audit log message.
- `ConfigurationChangeAuditMessage`
  A child interface specific for the configuration change category and used to log changes in the configuration data.
- `DataAccessAuditMessage`
  A child interface specific for the data read access category and used to log successful and unsuccessful access to any sensitive personal data.
- `DataModificationAuditMessage`
  A child interface specific for the data modification category and used to log successful and unsuccessful modifications to any sensitive personal data.
- `SecurityEventAuditMessage`
  A child interface specific for the security event category and used to log all security-relevant events. These are events that may impact the confidentiality, integrity, and availability of the system such as server start or stop, failed logins, failed authorization checks, start of critical transactions, and changes of critical system parameters.
- `TransactionalAuditLogMessage`
  The transactional interface is applicable for the *Configuration Change* category and indicates whether the audit log event initiated in a previous step has been successful or not. You can use one of the following methods for such logging:
  - `logPrepare()`
    Indicate that the audit event is about to happen.
  - `logSuccess()`
    Indicate that the audit event has been successful.
  - `logFailure()`
    Indicate that the audit event has failed.

Audit Log Message Format

The audit log message contains the following information:

Parameter	Description
`user`	The user that triggered the audit log event.
`objectId`	The unique ID of the object, which the action was performed on.
`key`	• The key of the data set. • The type of the data access channel.

Parameter	Description
`name`	• The heading name for the attribute that has been read or modified, either successfully or unsuccessfully • The name of an attachment when the event is triggered by the download or display of some attachments.
`id`	The ID of an attachment when the event is triggered by the download or display of some attachments.
`ip`	The source IP address that triggered the event during an external access.
`data`	The detailed data describing the security event
`timestamp`	This is the time at which the action was performed, not the time of the audit log creation.

Related Information

Configure an Application to Use the Audit Log Service [page 747]

7.2.2.4 Configure Authentication and Authorization

You configure authentication and authorization by using the integrated container authentication provided with the SAP Java build pack, the Spring security library, or open-source Cloud security libraries.

Prerequisites

- You have secured your SAP HANA XS advanced run time.
  To configure authorizations for Java run time, you just need to set them for SAP HANA XS advanced. For more information, see *Setting up Security Artifacts* in *Related Information* below.

  > **i** Note
  >
  > A role collection must have already been assigned to the user in SAP HANA. For more information, see *Maintaining Platform Users in XS Advanced* in the *SAP HANA Administrators Guide*.

- You have secured the application router.
  For more information, see *Configure the XS Advanced Application Router* in *Related Information* below.
- You have created an instance of the XS User Account and Authentication (UAA) service.
  You must have created the `xsuaa` service instance and have bound it to the application for which you want to configure authentication. For more information about creating a service instance, see *Create an*

Instance of the xsuaa Service and *Bind the xsuaa Service Instance to the XS Advanced Application* in *Related Information*.

Context

XS advanced enables offline validation of the JSON Web Token (JWT) used for authentication purposes; it does not require an additional call to the User Account and Authentication (UAA) service. The trust for this offline validation is created by binding the `xsuaa` service instance to your application. The information in this topic helps you set up authentication and authorization for your Java application by using the integrated container authentication provided with the SAP Java build pack, the Spring security library, or open-source Cloud security libraries.

> **i** Note
>
> As of SPS05 the `java-container-security` library is deprecated and replaced by the open-source Cloud `java-security` client library. See also Release Notes #1 and #2.

Configure Integrated Container Authentication of the SAP Java Buildpack

Context

The SAP Java build pack includes an authentication method called `XSUAA`. This makes an offline validation of the received JWT token possible. The signature is validated using the verification key received from the service binding to the `xsuaa` service.

To enforce a check for the `$XSAPPNAME.Display` scope, proceed as follows:

Procedure

1. Configure the use of the `XSUAA` authentication method.

   This is done in the `login-config` section of the `web.xml` file.

   Sample `web.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
    xmlns="http://java.sun.com/xml/ns/javaee
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                        http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd
    version="3.0">
    <display-name>sample</display-name>
    <login-config>
        <auth-method>XSUAA</auth-method>
    </login-config>
```

```
</web-app>
```

2. Add a security constraint to a servlet.

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.*;
import javax.servlet.http.*;
/**
 * Servlet implementation class HomeServlet
 */
@WebServlet(“/*”)
@ServletSecurity(@HttpConstraint(rolesAllowed = { “Display” }))
public class HomeServlet extends HttpServlet {
    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        response.getWriter().println(“principal” +
request.getUserPrincipal());
    }
```

Configure Spring Security (`java-container-security`)

Context

Applications using the Spring libraries can use the corresponding Spring security libraries. SAP HANA XS advanced model provides a module for offline validation of the received JSON Web Token (JWT). The signature is validated using the verification key received from the application's binding to the XS UAA service. To configure Spring security for your Java application, perform the following steps:

Procedure

1. Specify the relevant libraries in your build manifest.

   Sample `pom.xml`

```
<dependency>
  <groupId>com.sap.xs2.security</groupId>
  <artifactId>java-container-security</artifactId>
  <version>0.14.5</version>
</dependency>
```

2. Specify the required listeners in your `web.xml`.

   Sample `web.xml`

```
<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</
listener-class>
</listener>
<context-param>
```

```
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/spring-security.xml</param-value>
</context-param>
<filter>
  <filter-name>springSecurityFilterChain</filter-name>
  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-
class>
</filter>
<filter-mapping>
  <filter-name>springSecurityFilterChain</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

3. Configure the Spring beans by adding `spring-security.xml`.

   Sample `spring-security.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/
beans" xmlns:oauth="http://www.springframework.org/schema/
security/oauth2" xmlns:sec="http://www.springframework.org/
schema/security" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://www.springframework.org/schema/
security/oauth2      http://www.springframework.org/schema/security/spring-
security-oauth2-1.0.xsd         http://www.springframework.org/schema/
security         http://www.springframework.org/schema/security/spring-
security-3.2.xsd         http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.1.xsd">
   <!-- protect secure resource endpoints
============================================= -->
   <sec:http pattern="/**" create-
session="never" entry-point-ref="oauthAuthenticationEntryPoint"
access-decision-manager-ref="accessDecisionManager" authentication-manager-
ref="authenticationManager" use-expressions="true">
      <sec:anonymous enabled="false" />
      <sec:intercept-url pattern="/rest/addressbook/deletedata"
access="#oauth2.hasScope('${xs.appname}.Delete')" method="POST" />
      <sec:intercept-url pattern="/**" access="isAuthenticated()"
method="GET" />
      <sec:custom-filter ref="resourceServerFilter"
before="PRE_AUTH_FILTER" />
      <sec:access-denied-handler ref="oauthAccessDeniedHandler" />
   </sec:http>
   <bean id="oauthAuthenticationEntryPoint"
class="org.springframework.security.oauth2.provider.error.OAuth2Authentication
EntryPoint" />
   <bean id="oauthWebExpressionHandler"
class="org.springframework.security.oauth2.provider.expression.OAuth2WebSecuri
tyExpressionHandler" />
   <bean id="accessDecisionManager"
class="org.springframework.security.access.vote.UnanimousBased">
      <constructor-arg>
         <list>
            <bean
class="org.springframework.security.web.access.expression.WebExpressionVoter">
               <property name="expressionHandler"
ref="oauthWebExpressionHandler" />
            </bean>
            <bean
class="org.springframework.security.access.vote.AuthenticatedVoter" />
         </list>
      </constructor-arg>
   </bean>
   <sec:authentication-manager alias="authenticationManager" />
   <oauth:resource-server id="resourceServerFilter" resource-id="springsec"
token-services-ref="offlineTokenServices" />
   <bean id="offlineTokenServices"
class="com.sap.xs2.security.commons.SAPOfflineTokenServices">
```

```
      <property name="verificationKey" value="${xs.uaa.verificationkey}" />
      <property name="trustedClientId" value="${xs.uaa.clientid}" />
      <property name="trustedIdentityZone" value="${xs.uaa.identityzone}" />
   </bean>
   <bean id="oauthAccessDeniedHandler"
class="org.springframework.security.oauth2.provider.error.OAuth2AccessDeniedHa
ndler" />
   <!-- define properties file
============================================================ -->
   <bean
class="com.sap.xs2.security.commons.SAPPropertyPlaceholderConfigurer">
      <property name="location" value="classpath:/application.properties" />
   </bean>
</beans>
```

In this example, the most important line in the service context is highlighted in **bold**. The contents of this file specify which parts of the micro service are to be made secure. For example, an `HTTP POST` request to `/rest/addressbook/deletedata` requires the `Delete` scope.

4. Configure the security properties of the service instance in `application.properties`.

   Sample `application.properties` file of a Java application

   ```
   # parameters of hello-world-java
   xs.appname=java-hello-world
   ```

5. Update your Java code.

   The following code snippet shows that not only the requests are authenticated but also that the user principal is available.

   ```
   UserInfo userInfo = SecurityContext.getUserInfo();
   String name = userInfo.getLogonName();
   String email = userInfo.getEmail();
   String[] attribute = userInfo.getAttribute("my attribute");
   boolean hasDeleteScope = userInfo.checkLocalScope("Delete");
   ```

Configure Spring Security for Web Applications (`java-security`)

Context

Applications using the deprecated Spring Security OAuth can use the Cloud `java-security` client library with the Spring Security Adapter. SAP HANA XS advanced model provides a module for offline validation of the received JSON Web Token (JWT). The signature is validated using the verification key received from the XS UAA service. To configure Spring security for your Java Web application, perform the following steps:

Procedure

1. Specify the relevant libraries in your build manifest.

Sample `pom.xml`

```
<!-- take the latest version of security dependencies -->
<dependency>
     <groupId>org.springframework.security.oauth</groupId>
     <artifactId>spring-security-oauth2</artifactId>
</dependency>
<dependency>
     <groupId>com.sap.cloud.security.xsuaa</groupId>
     <artifactId>api</artifactId>
     <version>2.7.5</version>
</dependency>
<dependency>
     <groupId>com.sap.cloud.security</groupId>
     <artifactId>java-security</artifactId>
     <version>2.7.5</version>
</dependency>
```

2. Specify the required listeners in your `web.xml`.

   Sample `web.xml`

```
<listener>
  <listener-class>org.springframework.web.context.ContextLoaderListener</
listener-class>
</listener>
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/spring-security.xml</param-value>
</context-param>
<filter>
  <filter-name>springSecurityFilterChain</filter-name>
  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-
class>
</filter>
<filter-mapping>
  <filter-name>springSecurityFilterChain</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

3. Configure the Spring beans by adding `spring-security.xml`.

   Sample `spring-security.xml`

```
<?xml version="1.0" encoding="UTF-8" ?>
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:oauth="http://www.springframework.org/schema/security/oauth2"
 xmlns:sec="http://www.springframework.org/schema/security"
 xsi:schemaLocation="http://www.springframework.org/schema/security/oauth2
   https://www.springframework.org/schema/security/spring-security-oauth2.xsd
   http://www.springframework.org/schema/security
   https://www.springframework.org/schema/security/spring-security.xsd
   http://www.springframework.org/schema/beans
   https://www.springframework.org/schema/beans/spring-beans.xsd">
   <!-- protect secure resource endpoints
============================================= -->
   <sec:http pattern="/**" create-
session="never" entry-point-ref="oauthAuthenticationEntryPoint"
access-decision-manager-ref="accessDecisionManager" authentication-manager-
ref="authenticationManager" use-expressions="true">
      <sec:anonymous enabled="false" />
      <sec:intercept-url pattern="/rest/addressbook/deletedata"
access="#oauth2.hasScope('Delete')" method="POST" />
      <sec:intercept-url pattern="/**" access="isAuthenticated()"
method="GET" />
```

```
      <sec:custom-filter ref="resourceServerFilter"
before="PRE_AUTH_FILTER" />
      <sec:access-denied-handler ref="oauthAccessDeniedHandler" />
   </sec:http>
   <bean id="oauthAuthenticationEntryPoint"
class="org.springframework.security.oauth2.provider.error.OAuth2Authentication
EntryPoint" />
   <bean id="oauthWebExpressionHandler"
class="org.springframework.security.oauth2.provider.expression.OAuth2WebSecuri
tyExpressionHandler" />
   <bean id="accessDecisionManager"
class="org.springframework.security.access.vote.UnanimousBased">
      <constructor-arg>
         <list>
            <bean
class="org.springframework.security.web.access.expression.WebExpressionVoter">
               <property name="expressionHandler"
ref="oauthWebExpressionHandler" />
            </bean>
            <bean
class="org.springframework.security.access.vote.AuthenticatedVoter" />
         </list>
      </constructor-arg>
   </bean>
   <sec:authentication-manager alias="authenticationManager" />
   <oauth:resource-server id="resourceServerFilter" resource-id="springsec"
token-services-ref="offlineTokenServices" />
   <bean id="offlineTokenServices"
class="com.sap.cloud.security.adapter.spring.SAPOfflineTokenServicesCloud">
     <property name="localScopeAsAuthorities" value="true" />
   </bean>
   <bean id="oauthAccessDeniedHandler"
class="org.springframework.security.oauth2.provider.error.OAuth2AccessDeniedHa
ndler" />
   <!-- define properties file
============================================================ -->
   <bean
class="com.sap.xs2.security.commons.SAPPropertyPlaceholderConfigurer">
      <property name="location" value="classpath:/application.properties" />
   </bean>
</beans>
```

The contents of this file specify which parts of the micro service are to be made secure. For example, an `HTTP POST` request to `/rest/addressbook/deletedata` requires the `Delete` scope.

4. Configure the security properties of the service instance in `application.properties`.

   Sample `application.properties` file of a Java application

   ```
   xs.appname=java-hello-world
   ```

5. Update your Java code.

   The following code snippet shows that not only the requests are authenticated but also that the user principal is available.

   ```
   AccessToken token = SecurityContext.getAccessToken();
   String userName = token.getClaimAsString(TokenClaims.USER_NAME);
   String email = token.getClaimAsString(TokenClaims.EMAIL);
   boolean hasDeleteScope = token.hasLocalScope("Delete");
   ```

6. Optional: Use the `XSUserInfo` interface.

You can use the `XSUserInfoAdapter` from `java-security` to access the token data by means of the `XSUserInfo` interface, as illustrated in the following code example:

```
AccessToken token = SecurityContext.getAccessToken();
XSUserInfo userInfo = new XSUserInfoAdapter(token);
String hdbToken = userInfo.getHdbToken();
```

→ Tip

For information about using `java-security` to set up basic token validation in plain Java, for example, in cases where applications are not running in a Web context but still need to perform authentication and authorization checks with JWT tokens, see *Configure Authentication and Authorization Checks for Java applications* below.

Note that the token interface used in `java-security` is not compatible with the `XSUserInfo` interface. If you want to use `XSUserInfo`-specific methods such as `getHdbToken`, use the `XSUserInfoAdapter`.

Configure Spring Security for Spring Boot Applications (`spring-xsuaa`)

Context

Spring Boot Applications can use the Cloud `spring-xsuaa` client library, which provides a module for offline token validation of the received JSON Web Token (JWT). The signature is validated using the token keys received from the XS UAA service. To configure Spring security for your Spring Boot application, perform the following steps:

Procedure

1. Include the `spring-xsuaa` library.

   The library is available on the Maven Central Repository.

   ```
   <groupId>com.sap.cloud.security.xsuaa</groupId>
   <artifactId>spring-xsuaa</artifactId>
   ```

2. Read the instructions in the source documentation.

   For more information, see https://github.com/SAP/cloud-security-xsuaa-integration on GitHub.

3. Authenticate requests with JSON Web tokens (JWT).

   For instructions and examples showing how to authenticate requests with JWTs, see the following sample application:

   Sample application for Spring security

Configure Authentication and Authorization Checks for Java applications (`java-security`)

Context

Native Java Applications can use the Cloud `java-security` library to perform basic offline token validation in plain Java. This can be useful for applications that are not running in a Web context but still need to perform authentication and authorization checks using JWT tokens.

Procedure

1. Include the `java-security` library.

   The library is available on the Maven Central Repository.

   ```
   <groupId>com.sap.cloud.security.xsuaa</groupId>
   <artifactId>java-security</artifactId>
   ```

2. Read the instructions in the source documentation.

   For more information, see https://github.com/SAP/cloud-security-xsuaa-integration on GitHub.

3. Authenticate requests with JSON Web tokens (JWT).

   For instructions and examples showing how to authenticate requests with JWTs, see the following sample application:

   Sample application for Java security

Related Information

Set up Application Security in XS Advanced [page 978]
Configure the XS Advanced Application Router [page 913]
Create an Instance of the XS UAA Service [page 1013]
Bind the XS UAA Service Instance to the XS Advanced Application [page 1016]
Maintaining Platform Users in XS Advanced

7.2.2.5 Configure a Database Connection

You can configure your application to use a database connection so that the application can persist its data. This configuration is applicable for Tomcat or TomEE run times.

Related Information

Configure a Database Connection for the Tomcat Run Time [page 763]
Configure a Database Connection for the TomEE Run Time [page 764]
Database Connection Configuration Details [page 766]
SAP HANA HDI Data Source [page 769]

7.2.2.5.1 Configure a Database Connection for the Tomcat Run Time

Context

The steps below demonstrate how data sources can be configured in such a way that a developer or operator can change the SAP HANA service used for creating the data source without having to rebuild or release a new version of the application archive.

Procedure

1. Create a `context.xml` file.

   The `context.xml` file has to be inside the `META-INF/` directory of the application's WAR file and has to contain information about the data source to be used.

   > Example
   >
   > Sample `context.xml`
   >
   > ```
   > <?xml version='1.0' encoding='utf-8'?>
   > <Context>
   >  <Resource name="jdbc/DefaultDB"
   >    auth="Container"
   >    type="javax.sql.DataSource"
   >    factory="com.sap.xs.jdbc.datasource.tomcat.TomcatDataSourceFactory"
   >    service="${service_name_for_DefaultDB}"/>
   > </Context>
   > ```

   In this example a placeholder is used for the service instance name.

2. Add the default keys and their values.

   You need to include the data source information in `META-INF/sap_java_buildpack/config/resource_configuration.yml` of the WAR file to define a default value for the placeholder from the previous step.

> ### Example
>
> Defining a default service in `resource_configuration.yml`
>
> ```
> ---
>  tomcat/webapps/ROOT/META-INF/context.xml:
>   service_name_for_DefaultDB: di-core-hdi
> ```

This configuration ensures that the service used for the creation of data sources is determined while staging, and *di-core-hdi* is used. To use this method to change the data sources without having to rebuild or make a new release of the application, developers or operators need to perform an additional deployment of the same application archive.

3. Add the key values to be updated.

   To use a different service instance, you need to provide an environment variable in the `manifest.yml` file and bind the application to that service instance in the services section of the `manifest.yml` file.

> ### Example
>
> Defining a new service for the look-up of the data source
>
> ```
> env:
>   JBP_CONFIG_RESOURCE_CONFIGURATION: "['tomcat/webapps/ROOT/META-INF/
> context.xml': {'service_name_for_DefaultDB' : 'my-local-special-di-core-
> hdi'}]"
> ```

Using this configuration, when the application starts, the factory named `com.sap.xs.jdbc.datasource.TomcatDataSourceFactory` takes the parameters bound to service *my-local-special-di-core-hdi* from the environment, creates a data source, and binds it under `jdbc/DefaultDB`. The application then uses the Java Naming and Directory Interface (JNDI) to look up how to connect with the database.

Related Information

Configure a Database Connection [page 762]
Database Connection Configuration Details [page 766]
SAP HANA HDI Data Source [page 769]

7.2.2.5.2 Configure a Database Connection for the TomEE Run Time

Context

The following steps demonstrate how data sources can be configured in such a way that a developer or operator can change the SAP HANA service used for creating the data source without having to rebuild or release a new version of the application archive.

Procedure

1. Create a `resources.xml` file.

   - If the data source is to be used from a Web application, you have to create the file inside the `WEB-INF/` directory.
   - If the data source is to be used from Enterprise JavaBeans (EJBs), you have to create the file inside the `META-INF/` directory.

   The `resources.xml` file has to be inside the application's WAR file and has to contain information about the data source to be used.

   > **Example**
   >
   > Sample `resources.xml`
   >
   > ```
   > <?xml version='1.0' encoding='utf-8'?>
   > <resources>
   >  <Resource id="jdbc/DefaultDB" provider="xs.openejb:XS Default JDBC
   > Database" type="javax.sql.DataSource" >
   >     service=${service_name_for_DefaultDB}
   >   </Resource>
   > </resources>
   > ```

   In this example a placeholder is used for the service instance name.

2. Add the default keys and their values.

   You need to include the data source information in `META-INF/sap_java_buildpack/config/resource_configuration.yml` of the WAR file to define a default value for the placeholder from the previous step.

   - > **Example**
     >
     > Defining a default service in `resource_configuration.yml` for a Web application
     >
     > ```
     > ---
     >  tomee/webapps/ROOT/WEB-INF/resources.xml:
     >   service_name_for_DefaultDB: di-core-hdi
     > ```

   - > **Example**
     >
     > Defining a default service in `resource_configuration.yml` for an EJB
     >
     > ```
     > ---
     > tomee/webapps/ROOT/META-INF/resources.xml:
     >   service_name_for_DefaultDB: di-core-hdi
     > ```

   With this configuration the service used for creation of data sources is determined while staging and *di-core-hdi* is used. To use this method to change the data sources without having to rebuild or make a new release of the application, developers or operators need to perform an additional deployment of the same application archive.

3. Add the key values to be updated.

   To use a different service instance, you need to provide an environment variable in the `manifest.yml` file, and bind the application to that service instance in the services section of the `manifest.yml` file.

- Example

  Defining a new service for the look-up of the data source in a Web application

  ```
  env:
    TARGET_RUNTIME: tomee
    JBP_CONFIG_RESOURCE_CONFIGURATION: "[
      'tomee/webapps/ROOT/WEB-INF/resources.xml':
        {'service_name_for_DefaultDB' : 'my-local-special-di-core-hdi'}
       ]"
  ```

- Example

  Defining a new service for the look-up of the data source in an EJB

  ```
  env:
    TARGET_RUNTIME: tomee
    JBP_CONFIG_RESOURCE_CONFIGURATION: "[
      'tomee/webapps/ROOT/META-INF/resources.xml':
        {'service_name_for_DefaultDB' : 'my-local-special-di-core-hdi'}
       ]"
  ```

With this configuration, when the application starts, the XS factory takes the parameters bound for service *my-local-special-di-core-hdi* from the environment, creates a data source, and binds it under `jdbc/DefaultDB`. The application then uses the Java Naming and Directory Interface (JNDI) to look up how to connect with the database.

Related Information

Configure a Database Connection [page 762]
Database Connection Configuration Details [page 766]
SAP HANA HDI Data Source [page 769]

7.2.2.5.3 Database Connection Configuration Details

Define details of the database connection used by your Java application in XS advanced.

Configuration Files

To configure your XS advanced application to establish a connection to the SAP HANA database, you must specify details of the connection in a configuation file. For example, you must define the data source that the application will use to discover and look up data. The application then uses a Java Naming and Directory Interface (JNDI) to look up the specified data in the file.

The easiest way to define the required data source is to declare the keys for the data source in a resource file.

Example

For the Tomcat run time, you can create a `context.xml` file in the `META-INF/` directory with the following content:

```
<?xml version='1.0' encoding='utf-8'?>
<Context>
 <Resource name="jdbc/DefaultDB"
    auth="Container"
    type="javax.sql.DataSource"
    factory="com.sap.xs.jdbc.datasource.tomcat.TomcatDataSourceFactory"
    service="di-core-hdi"/>
</Context>
```

For the TomEE run time, you can create a `resources.xml` file in the `WEB-INF/` directory with the following content:

```
<?xml version='1.0' encoding='utf-8'?>
<resources>
 <Resource id="jdbc/DefaultDB" provider="xs.openejb:XS Default JDBC Database"
type="javax.sql.DataSource" >
    service=di-core-hdi
 </Resource>
</resources>
```

The problem with this simple approach is that your `WAR` file cannot be signed, and any modifications can only be made in the `WAR` file. For this reason, it is recommended that you do not use the method in a production environment; use modification settings in `resource_configuration.yml` and `manifest.yml` instead, as illustrated in the following examples:

The `resource_configuration.yml` is used for the default “key” settings.

Example

Defining a default service in `resource_configuration.yml`

```
---
 tomcat/webapps/ROOT/META-INF/context.xml:
  service_name_for_DefaultDB: di-core-hdi
```

Specifying a default name for a service is useful (for example, for automation purposes) only if you are sure there can be no conflict with other names. For this reason, it is recommended that you include a helpful and descriptive error message instead of a default value. That way the error message will be part of an exception triggered when the data source is initialized, which helps troubleshooting.

Example

Sample `resource_configuration.yml`

```
---
 tomcat/webapps/ROOT/META-INF/context.xml:
  service_name_for_DefaultDB: Specify the service name for Default DB in
manifest.yml via "JBP_CONFIG_RESOURCE_CONFIGURATION"..
```

Placeholders

The generic mechanism `JBP_CONFIG_RESOURCE_CONFIGURATION` basically replaces the key values in the specified files. For this reason, if you use placeholders in the configuration files, it is important to ensure that you use unique names for the placeholders.

Example

Sample `context.xml`

```
<?xml version='1.0' encoding='utf-8'?>
<Context>
 <Resource name="jdbc/DefaultDB"
  auth="Container"
  type="javax.sql.DataSource"
  description="Datasource for general functionality"
  factory="com.sap.xs.jdbc.datasource.tomcat.TomcatDataSourceFactory"
  service="${service_name_for_DefaultDB}"
  maxActive="${max_Active_Connections_For_DefaultDB}"  />
 <Resource name="jdbc/DefaultXADB"
  auth="Container"
  type="javax.sql.XADataSource"
  description="Datasource for functionality requiring more than one
transactional resource"
  factory="com.sap.xs.jdbc.datasource.tomcat.TomcatDataSourceFactory"
  service="${service_name_for_DefaultXADB}"
  maxActive="${max_Active_Connections_For_DefaultXADB}" />
</Context>
```

The names of the defined placeholders are also used in the other resource files.

Example

Sample `resource_configuration.yml`

```
---
tomcat/webapps/ROOT/META-INF/context.xml:
  service_name_for_DefaultDB: di-core-hdi
  max_Active_Connections_For_DefaultDB: 100
  service_name_for_DefaultXADB: di-core-hdi-xa
  max_Active_Connections_For_DefaultXADB: 100
```

Sample `manifest.yml`

```
---
env:
  JBP_CONFIG_RESOURCE_CONFIGURATION: "['tomcat/webapps/ROOT/META-INF/
context.xml': {'service_name_for_DefaultDB' : 'my-local-special-di-
core-hdi' , 'max_Active_Connections_For_DefaultDB' : '30',
'service_name_for_DefaultXADB' : 'my-local-special-xa-di-core-hdi' ,
'max_Active_Connections_For_DefaultXADB' : '20'  }]"
```

Related Information

Configure a Database Connection [page 762]

SAP HANA HDI Data Source [page 769]

7.2.2.5.4 SAP HANA HDI Data Source

To use a SAP HANA HDI container from your Java run time, you must create a service instance for the HDI container and bind the service instance to the Java application. To create a service instance, use the `xs create-service` command, as illustrated in the following example:

Sample Code

```
xs create-service hana hdi-shared my-hdi-container
```

To bind the service instance to a Java application, specify the service instance in the Java application's deployment manifest file (`manifest.yml`).

Sample Code

```
services:
   - my-hdi-container
```

Next, add the resource reference to the `web.xml` file, which must have the name of the service instance:

Sample Code

```
<resource-ref>
  <res-ref-name>jdbc/my-hdi-container</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
</resource-ref>
```

Look up the data source in your code; you can find the reference to the data source in the following ways:

- Annotations

  ```
  @Resource(name = "jdbc/my-hdi-container")
  private DataSource ds;
  ```

- Java Naming and Directory Interface (JNDI) look-up

  ```
  Context ctx = new InitialContext();
  return (DataSource) ctx.lookup("java:comp/env/jdbc/my-hdi-container");
  ```

7.2.2.6 Debug an XS Advanced Java Application

You can use this procedure only if you can restart your Java application or the application has not yet been pushed.

Procedure

1. Add an environment variable to the Java application's manifest file (`manifest.yml`).

   Open the `manifest.yml` file and add the `<JBP_CONFIG_JAVA_OPTS>` variable, as illustrated in the following example:

   Example

   ```
   ---
   applications:
   - name: test-java
     memory: 256M
     instances: 1
     path: target/test-java.war
     env:
       TARGET_RUNTIME: tomcat
       JBP_CONFIG_JAVA_OPTS: [java_opts: '-
   agentlib:jdwp=transport=dt_socket,address=8000,server=y,suspend=y']
   ```

   Note

   With `<JBP_CONFIG_JAVA_OPTS>`, you set options to the JVM, which will activate the Java Debug Wire Protocol (JDWP) agent. You could change the desired options to your liking. For more information, about the available options, see *-agentlib:jdwp and -Xrunjdwp sub-options* in *Related Information* below.

2. Push the application.

   To push the application to the XS advanced Java run-time, use the following command:

   ```
   xs push -f manifest.yml
   ```

   Before the application starts, it waits for a debugger to connect to port 8000.

3. Attach the debugger to the specified port.

   Caution

   Your application will fail to start if you do not attach a debugger in time.

Related Information

-agentlib:jdwp and -Xrunjdwp sub-options

7.2.2.7 Enable and Use Java Data Services

Enable Java Data Services, a native Java client for using Core Data Services functionality in the XS advanced Java run time.

Context

The Java Data Services (JDS) for XS advanced provides a set of tools that facilitate the consumption of Core Data Services (CDS) artifacts in Java applications. JDS enables you to map existing CDS data models to the corresponding Java Persistence API (JPA) 2.1 data models so that they can be consumed by Java applications. To use Java Data Services in XS advanced, perform the following actions:

Procedure

1. Start the JDS tool using the following command in a command shell.

   ```
   $:/jds> java -jar jds-import-1.0.1-full.jar help
   ```

2. Have a look at the tools provided in JDS.

   The `help` option lists the top-level functions available.

   Output Code

   ```
   usage: java -jar importer-x.y.z.jar <command> <parameters>
   Commands:
        help - displays this help text
        userguide - display the json userguide
        display - display available entities in namespace
        create - create Java classes for entities
        <config file>.json - process instructions in document (advanced)
   ```

   → Tip

   Further information is displayed concerning how to set and use parameters.

Related Information

Java Data Services in XS Advanced [page 772]

7.2.2.7.1 Java Data Services in XS Advanced

Java Data Services provide a native Java client for using Core Data Services functionality in the XS advanced Java run time container.

The Java Data Services (JDS) are a collection of tools which help to enable the consumption of Core Data Services (CDS) artifacts in Java applications. JDS maps existing CDS data models to corresponding Java Persistence API (JPA) 2.1 data models that can then be consumed by Java applications, for example, using EclipseLink or Hibernate. The JDS tools are available by default from the command line, as illustrated in the following example:

Sample Code

```
java -jar jds-import-1.0.1-full.jar help
```

In addition to providing all the parameters of `jds-tool` on the command line or inside the `jds.ini` file it is also possible to use a JavaScript Object Notation (JSON) document instead. For more advanced features such as providing information about back links (one-to-many associations) or “n:m” associations, a JSON document is required. The meaning of all possible flags or parameters is equal to the command-line options.

Note

When using a JSON document, the contents of any `jds.ini` properties file are ignored.

The following example shows the basic structure of a JSON document that describes the parameters available for the JDS tools:

Code Syntax

```
{
  "connection" : {
    /* connection parameter */
  },
  "<command>" : {
    /* either create or display - as with the commandline tool */
  },
  "contexts" : {
    /* optional additional metadata for the "create" command.
       used to provide information for
       - backlinks
       - m:n connections
       and list all entities if you don't want to get JPA classes
       for all CDS entities in the namespace */
  }
}
```

The generated Java source files can be copied into your Java application. Remember to regenerate the files if your underlying CDS data model changes.

Caution

Note that JDS is for use in consuming existing CDS data models; it is strongly recommended **not** to use `drop-create-tables` with JDS.

Connection Parameters

For the `"connection"` object, all properties except `proxy` are mandatory; if the mandatory properties are omitted, they and their assigned value are queried by the JDS tool.

Code Syntax

```
"connection" : {
   "password": "<yourPassword>",
   "schema": "<yourSchema>",
   "host": "<your host>:<your port>",
   "user": "<yourUser>",
   "proxy": "<yourproxy>"
}
```

Commands

For the `"command"` object, some parameters (for example, `namespace`) are mandatory and some optional, as illustrated in the following example:

Code Syntax

```
"create": {
   "namespace":"<namespace of CDS entities>", //mandatory
   "package": "<packageName>", // optional
   "targetDirectory": "<targetDirectory>", //optional
   "flags": ["continueOnError", "simulate","overwrite",
             "includeSchema", "listedEntitiesOnly"], //optional
   "jpaFormat": { //optional
       "embeddedKeySuffix": "<MyPK>",
       "indentationDepth": 8,
       "quotationCharacter": "\\\""
   }
}
"display": {
   "namespace": "<your namespace>",
   "flags": ["withoutDependencies"]      // currently only one flag
}
```

You can use `"contexts"` to add additional metadata to `create` objects. This might be necessary if there is no support for one-to-many (back link) and many-to-many associations. The following example shows a simple way to use contexts to supply additional metadata:

Code Syntax

```
"contexts" : {
             "post": {
                 "Comments": {
                      "target": "comment",
                       "viaBacklink": "Post"
                 },
                 "Tags": {
                      "target": "tag",
                      "viaEntity": "tags_by_post",
                      "sourceField": "Tag.tid",
```

```
                    "targetField": "Post.pid"
                }
            },
            "comment": {},
            "tag": {} // no need for reference to 'post'
 }
```

! Restriction

The `"contexts"` object in combination with the flag `"listedEntitiesOnly"` can also be used to create JPA classes only for a limited selection of entities instead of the full namespace.

Command line Properties

The most commonly used command-line arguments, for example, connection properties, can be placed in a `jds.properties` file, as illustrated in the following example:

Code Syntax

```
#---JDS connection properties---
password=<secretPassword>
schema=<mySchema>
host=<myHost>:<myPort>
user=<myUserName>
```

CDS Support

This release of Java Data Services supports the following CDS elements:

- CDS entities
- CDS types
- CDS views
- CDS associations (one-to-one, only)

Related Information

Enable and Use Java Data Services [page 771]
Download and Install JavaScript Data Services [page 689]

7.2.2.8 Run a Java Main Application

You can create a Java application that starts its own run time.

Prerequisites

You are not using any resource configurations.

The resource configurations needed for the database connection are not applicable for the Java Main application. For more information about database connections, see *Related Information*.

Context

The SAP Java buildpack provides an Apache Tomcat, Apache TomEE or an Apache TomEE 7 run time. However, you can create a Java Main application to use other frameworks and Java run times such as *Spring Boot*, *Jetty*, *Undertow*, or *Netty*.

> ⚠ Caution
>
> The Tomcat, TomEE and TomEE 7 run times provided by the SAP Java buildpack have the following functionality:
>
> - Solution manager end-to-end tracing
> - Java EE web container security
>   Java Main applications can only use Spring Security.
> - Management of exit codes
>   The exit codes are used by XS advanced to monitor started processes. If a process exits with a *0* exit code, it will not be started automatically.
>
> When you create an application to use its own run time, you have to take into consideration that the above functionality might not be available.

Procedure

1. Make sure your built JAR archive is configured properly.

   Regardless of the tool you use to build your Java application, you have to make sure that the following tasks are performed:

   a. You have built a JAR archive.
   b. You have specified a main class in the `META-INF/MANIFEST.MF` file of the JAR archive.

Sample Code

Sample `MANIFEST.MF`

```
Manifest-Version: 1.0
Archiver-Version: Plexus Archiver
Built-By: p123456
Created-By: Apache Maven 3.3.3
Build-Jdk: 1.8.0_45
Main-Class: com.companya.xs.java.main.Controller
```

c. You have packaged all your dependent libraries in the JAR file, also known as creating an uber JAR or a fat JAR.

If you are using Maven as your build tool, you can use the *maven-shade-plugin* to perform the above tasks.

Sample Code

An example configuration for the *maven-shade-plugin*

```
<build>
  <finalName>java-main-sample</finalName>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-shade-plugin</artifactId>
      <version>2.4.3</version>
      <configuration>
        <createDependencyReducedPom>false</createDependencyReducedPom>
        <filters>
          <filter>
            <artifact>*:*</artifact>
            <excludes>
              <exclude>META-INF/*.SF</exclude>
              <exclude>META-INF/*.DSA</exclude>
              <exclude>META-INF/*.RSA</exclude>
            </excludes>
          </filter>
        </filters>
      </configuration>
      <executions>
        <execution>
          <phase>package</phase>
          <goals>
            <goal>shade</goal>
          </goals>
          <configuration>
            <transformers>
              <transformer
implementation="org.apache.maven.plugins.shade.resource.ServicesResourceTra
nsformer" />
              <transformer
implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTra
nsformer">
                <manifestEntries>
                  <Main-Class>com.sap.xs.java.main.Controller</Main-Class>
                </manifestEntries>
              </transformer>
            </transformers>
          </configuration>
        </execution>
      </executions>
    </plugin>
  </plugins>
</build>
```

2. Configure `manifest.yml`.

   To be able to push the Java Main application, you need to specify the path to the `.jar` archive in the `manifest.yml` file.

   Sample Code

   Sample manifest.yml

   ```
   ---
   applications:
   - name: java-main
     memory: 128M
     path: ./target/java-main-sample.jar
     instances: 1
     services:
     - java_main_hana
   ```

3. Push the application to XS advanced run time.

   For that purpose, use the `xs push` command.

Example

Donna Moore would like to create a Java Main application to use its own run time. For that purpose she performs the following steps:

1. She downloads her *sample_main* application from the company's server and extracts it into a local folder.
2. She navigates in the command line tool to the *sample_main* directory of the application and builds it. Because Maven is installed on her computer, she can perform this operation with the command line `mvn clean install`.
3. After the build is successful, she checks that her folder contains a `sample_main.jar` file, which is located in the `target` directory.
4. She opens the `sample_main.jar` file and checks that the META-INF/MANIFEST.MF file contains the *Main-Class* header with a value the name of the main class.
5. She adds the path to the JAR file in the `manifest.yml` file.
   She needs to do that to make sure that the application can be pushed to the XS advanced run time. For that purpose, she adds the following line in the manifest file:

   ```
   path: ./target/sample_main.jar
   ```

6. Finally, she pushes the Java Main application with the following command line:

   ```
   xs push sample_main
   ```

Related Information

Configure a Database Connection [page 762]

7.3 The SAP HANA XS Advanced Python Run Time

SAP HANA XS advanced provides a Python run time to which you can deploy your Python applications.

SAP HANA XS advanced provides a build pack for you to deploy your Python code, however unlike Java and Node.js, SAP HANA XS advanced does not package pre-built Python run-time binaries. This means that you have to deploy your own Python run-time version prior to application deployment. During application deployment, the build pack ensures that this run time is provided to the application and that the correct certificates, configuration, and environment is set for your application.

> **i** Note
>
> The XS advanced run-time platform makes no assumptions about which frameworks and libraries to use to implement the Python micro service. The configuration scripts included in the latest Python build pack for XS advanced require at least Python 3 to be installed on the target system.

XS advanced does, however, provide the following components to help build a Python micro service:

- Setup of the SAP HANA client for communication with SAP HANA
- A Python library to support JSON Web Tokens (JWT) validation
  Used for communication between the application router, micro services, and the database
- A Python client for communication with the Audit Log service
- A Python library for creating and deleting service instances per tenant within an application during run time

Related Information

Set Up the Python Run Time for XS Advanced [page 778]
Download and Consume Python Libraries [page 782]
Tutorials: Setting Up the Python Run-Time Environment in XS Advanced [page 792]

7.3.1 Set Up the Python Run Time for XS Advanced

Find information about how to set up the Python run time for your XS advanced Python applications.

Prerequisites

- The Python run time is pre-built from sources by the administrator. The XS Advanced administrator has added the Python run time versions via the `xs create-runtime` command prior to application deployment. For more information, see *Building the Python Run Time from Sources* in the *Related Information* below.

- The Python run time environment must be version 3.* or higher.
- The pre-built Python run time environment must include `setuptools` and `pip`.

Procedure

1. Set up the application.
   a. Specify the Python version in the application using the `runtime.txt` or `Pipfile.lock` file included in the root folder of the deployed application.
   b. Format the Python version as shown in the following example:

      ```
      major.(minor|x)|(.patch|x)
      ```
   c. Declare the application dependencies either in `requirements.txt`, in `setup.py`, or in `Pipfile.lock`. Both `pip` and `pipenv` are supported for dependencies installation.
   d. Download application dependencies by putting them in the `vendor` directory at the root of the application.

      The build pack will execute `pip install` using the provided `requirements.txt` file (or generate a `requirements.txt` file from the provided `Pipfile.lock` file). If the vendor directory exists, it will be passed to the `pip install` command as an additional parameter.
   e. Provide a start command for the application, for example, with one of the following options:
      - A `Procfile` for cases where no start command is provided for the build pack. For more information see *Related Information* below.
      - The `start` command in the `manifest.yml` file
      - Provide a Python start script named `server.py` in the root folder of your application. If none of the above options is available, the build pack will start the application with the command `python server.py`.
      - The `start` command provided in `xs push` with the -c option.
2. Deploy the Python application.

   > **i** Note
   >
   > The Python build pack reports an error if there is no run time matching the requested version.

   During application staging, the Python build pack copies the certificates provided by the XS advanced run time in the application's local folder. The locations of the certificates are available at run time as a colon-separated list in the `XS_CACERT_PATH` environment variable.

Related Information

Building from Sources and Deploying the Python Run Time to XS Advanced [page 780]

Procfile

Python Programming (SAP HANA Client Interface Reference)

7.3.1.1 Building from Sources and Deploying the Python Run Time to XS Advanced

Building Python from sources requires several development packages to be installed on the operation system. The repositories providing these packages need to be configured and differ according to the specific Linux distributions.

> ⚠ Caution
>
> The content of this guide has been tested on the operating systems and architecture supported by the XS advanced run time. If you are using other untested versions, differences may occur.

The following table lists the packages and sample commands for installation on Suse Linux Enterprise Server (SLES) and Red Hat Enterprise Linux (RHEL):

Commands for Python Package Installation

Package	Installation on SLES	Installation on RHEL
`tk-devel`	`zypper install tk-devel`	`yum install tk-devel`
`tcl-devel`	`zypper install tcl-devel`	`yum install tcl-devel`
`libffi-devel`	`zypper install libffi-devel`	`yum install libffi-devel`
`openssl-devel`	`zypper install openssl-devel`	`yum install openssl-devel`
`readline-devel`	`zypper install readline-devel`	`yum install readline-devel`
`sqlite3-devel`	`zypper install sqlite3-devel`	`yum install sqlite3-devel`
`ncurses-devel`	`zypper install ncurses-devel`	`yum install ncurses-devel`
`xz-devel`	`zypper install xz-devel`	`yum install xz-devel`
`zlib-devel`	`zypper install zlib-devel`	`yum install zlib-devel`
`libbz2-devel`	`zypper install libbz2-devel`	`yum install libbz2-devel`

After completion, the `make` command prints a list of missing packages. Check this list carefully, install any missing packages, and re-run the `make` command.

> **Note**
>
> If you are building on SLES, the output will be similar to the following:
>
> ```
> Failed to build these modules:
> _curses_panel
> ```
>
> The error is not critical for the Python run time because the skipped module is for console UI building. However, if you want to fix the problem, you can create a symbolic link to the `/usr/include/ncurses/panel.h` header in `/usr/include/`. For more information, see *Related Information* below.

Building Python from sources is explained in the Python documentation. For more information about Python version 3.6.x, see *Related Information* below.

> **Tip**
>
> For productive scenarios, it can be helpful to run a build for Python with the `--enable-optimizations` flag. Bear in mind, however, that this option can sometimes lead to problems. If in doubt, do not use this option.
>
> In older Python versions *setuptools* and *pip* are not installed by default, so the additional option `--with-ensurepip=install`, should be provided to the configuration script.
>
> It is recommended to build older Python versions with UTF-32 internal strings representation, for example, with the configuration option `--enable-unicode=ucs4`.

Examples

Latest Python 3

1. Download and extract the sources from the *python downloads* link in *Related Information* below.
2. Run a build; the sample is for version 3.6.2.
3. Change the paths in the following example to suit the needs of your environment.

> **Sample Code**
>
> ```
> ./configure --prefix=/home/<myuser>/tmp/builds/Python-3.6.2 --exec-prefix=/
> home/<myuser>/tmp/builds/Python-3.6.2 --enable-optimizations
> make -j4
> make altinstall
> ```

The build `results` directory should look like the following example:

> **Output Code**
>
> ```
> .
> ├── bin
> ├── include
> ├── lib
> └── share
> ```

> **Note**
>
> If there is a `lib64` directory (which may occur if you are building on something other than PPC, RHEL and SLES) you should manually copy its contents to the `lib` directory to avoid build errors.

To use the built Python run time to run Python applications on XS advanced, you need to deploy the Python run time to XS advanced using the `xs create-runtime` command, as illustrated in the following example, which also shows how to use the example build directories:

> **Sample Code**
>
```
xs create-runtime -p /home/<myuser>/tmp/builds/Python-x.x.x
```

Related Information

https://bugs.python.org/issue12271
https://docs.python.org/3.7/using/unix.html#building-python
python downloads

7.3.2 Download and Consume Python Libraries

A selection of SAP-specific and ready-to-use Python client libraries is available for download from the SAP Service Marketplace.

Context

To download the SAP Python Client libraries, perform the following steps:

Procedure

1. Download the ZIP archive `XS_PYTHON` from SAP Service Marketplace.

   The `XS_PYTHON` ZIP archive is available for download on SAP Service Marketplace at the following location for those people with the required S-User ID:

   Service Marketplace > *Products* > *Software download* > *SUPPORT PACKAGES & PATCHES* > *Search* > *XS PYTHON 1*
2. Extract the contents of the `XS_PYTHON` archive into a folder in your local file system, for example `sap_dependencies`.

The overall recommendation for applications is for them to be deployed as self-contained. For this reason, applications need to include all of their dependent packages so that the staging process does not need to make any network calls. The Python build pack provides a mechanism for this; applications can make provision for their dependencies by creating a `vendor/` directory in the application's root directory and executing the following command to download (-d) the dependent packages and save them in the `vendor/` directory:

```
pip download -d vendor -r requirements.txt –-find-links sap_dependencies
```

> **i** Note
>
> The `pip download` command performs the same dependency-resolution and downloading steps as `pip install`. Instead of installing the dependencies, however, `pip download` collects the downloaded distributions in the specified directory (the default is the current directory). To facilitate offline or locked-down package installation, this directory can later be passed as a value to `pip install --find-links`. For more information, see **pip download** in *Related Information* below.

Make sure you are always downloading packages for the correct platform. Your application must package the dependencies for the platform it will run on. For example, if you are packaging your application in a Windows OS environment, but the XS advanced run time where the application will be deployed is running on SLES, you must package the Python dependencies for SLES, for example, by including the *--platform* option in the `pip download` call.

You should also always ensure the correct dependencies for the corresponding Python run-time version, that is; the run-time version that your Python application will run on. Either execute the `pip download` command with the same Python version, or pass the *--python-version* flag.

For more information, see **pip download** in the *Related Information* below.

Related Information

SAP Software Center
pip download
The Python Package Index (PyPI)

7.3.2.1 Standard Python Packages for XS Advanced

SAP includes a selection of Python packages, which are available for download and use, for customers and partners who have the appropriate access authorization, from the SAP Service Marketplace (SMP). For more information about how to download and use Python packages from the SAP Service Marketplace, log on to the SMP and search for the software component XS_PYTHON, which is an archive that contains the SAP packages. The following table lists the SAP Python packages that are currently available. For more details about the contents of each Python package as well as any configuration tips, see the README file in the corresponding package.

Python Packages

Package	Description
`sap_instance_manager`	Python package for creating and deleting service instances per tenant within an application at run time
`sap_audit_logging`	Provides audit logging functionality for Python applications
`sap_xssec`	XS Advanced Container Security API for Python
`sap_cf_logging`	This is a collection of support libraries for Python applications running on Cloud Foundry provide the following features: • Generate structured application log messages • Enable web applications in your application stack to collect request metrics **Note** `sap_cf_logging` is not included in SAP's `XS_PYTHON` archive; it is an open-source library that is developed by SAP and published on the Python Package Index (Pypi).
`hdbcli`	The SAP HANA database client provides tools that enable connections to the SAP HANA database.

`sap_instance_manager`

This package provides a client for the Instance Manager - a component that creates and deletes service instances (via REST API) for a specified service key. The Instance Manager can be used in the context of multi-tenant applications, where the tenant ID is the key that an instance is associated with. Python application can dynamically create and delete service instances per tenant at run time. An instance of a managed service of the desired type is created first and is then bound to the application. For an SAP HANA database the managed service is called *managed-hana*. This service binding only provides parameters (HTTP endpoints and credentials) which can later be used by the application for creating and deleting service instances of the desired type for each tenant.

Application Programing Interface (API)

The following example shows how to use the instance manager API to manage service instances for the XS advanced application used by multiple tenants:

Sample Code

XS Advanced Service - Instance Manager API

```
from sap import instance_manager
# create instance manager client
```

```
client = instance_manager.create(options)
# create instance
instance = client.create('my-tenant')
print(instance)
# ...
# get already created instance
instance = client.get('my-tenant')
print(instance)
# ...
# delete instance
client.delete('my-tenant')
```

Options

The managed service bound to the application (for example `managed-hana`) provides credentials as well as REST endpoints of the Instance Manager - the component that handles creation and deletion of services. These credentials and endpoints are mandatory.

The create and delete operations are executed asynchronously on server side. To provide an easier interface, this library also implements polling until an operation is finished. Developers can tune polling via some optional properties.

Since operations involve network activity (thus, can be considered relatively slower) the package also caches the created instances. Cache options can also be provided by developers.

Instance Manager Configuration Options

Property	Mandatory	Type	Details
user	Yes	string	User for authentication.
password	Yes	string	Password for the user.
post_managed_instance_url	Yes	string	REST endpoint used for creating a new service instance for a tenant.
get_managed_instance_url	Yes	string	REST endpoint.
get_all_managed_instances_url	Yes	string	REST endpoint used for getting details about all instances (for all tenants).
delete_managed_instance_url	Yes	string	REST endpoint used for deletion of a service instance.
polling_interval_millis	No	integer	States how many milliseconds to wait between requests in the polling phase. Defaults to 300 milliseconds. Minimum value is 0.

Property	Mandatory	Type	Details
polling_timeout_seconds	No	integer	Sets a limit for the amount of time (in seconds) that can be spent in polling. Defaults to 120 seconds. Minimum value is 1.
cache_max_items	No	integer	States the capacity of the cache. Default value is 500. Minimum value is 1.
cache_item_expire_seconds	No	integer	Number of seconds after which a cache entry expires. Defaults to 600 seconds. Minimum value is 1.

> **i** Note
>
> A managed service binding contains all of the mandatory properties mentioned above. It is recommended to have a single instance-manager client Python object for each managed service that is bound to the application.

Methods

The following table lists the methods available with the instance-manager API:

Instance-Manager API Methods

Method	Description
`create(tenant, optional_parameters)`	Creates a service instance for the provided tenant. The method polls until the instance is successfully createdk. Reports error if an instance for this tenant already exists. The second argument is optional.
`get(tenant)`	Retrieves the corresponding service instance for the specified tenant either from the cache or from the server. A returned value of `None` indicates that no service instance exists for the specified tenant. **!** Restriction The `get` method only polls if the service instance is in status `CREATION_IN_PROGRESS`. In all other cases it returns the service instance as it is on server. There is no guarantee that the `credentials` property on the `instance` object is included in the callback.
get_all()	Retrieves the service instances for all tenants as an array of objects. Filtering of the instances according to their status (e.g. `CREATION_SUCCEEDED`, `CREATION_IN_PROGRESS`) does not take place. Thus, having the credentials property on each of the instances provided is not guaranteed. This method updates the cache.

Method	Description
`delete`	Removes the service instance for the provided tenant. The method polls until the instance is successfully deleted and then invokes the callback. An error is reported if no instance exists for the specified tenant.

> → Tip
>
> If the callback of a method is invoked with an error and the error is caused by an unexpected HTTP response code received from the server, then the error object will have a `statusCode` property with the same status code as the received HTTP response.

sap_audit_logging

Provides audit logging functionality for Python applications.

Audit logging concerns writing entries in a specific format to a log storage. The entries written to the audit log concern any events of significant importance, for example, security events which may have an impact on the confidentiality, the integrity, or the availability of a system. Access to sensitive personal data (both reading and altering) such as bank accounts, political opinion, health status is also an event that is written to the audit log. Audit logs are not the same as ordinary log files. Ordinary log files are used by the system administrator who need to keep track of the state of a system; audit logs are read by an auditor. Audit logs are subject to legal requirements, which in some countries are stricter than in others. In general, the events that are supposed to be recorded in the audit log can be grouped into the following categories:

- Changes to system configurations (which may have significant effect on the system itself)
- Access to sensitive data (related to data privacy)
- General security events (for example, starting or stopping a system, failed authorization checks, etc.)

> → Tip
>
> For more detailed information about the `sap_audit_logging` package, see the `README.rst` file included in the `sap_audit_logging` package.

sap_xssec

The client security library includes the XS advanced container security API for Python.

Authentication for Python applications in XS advanced relies on a special usage of the OAuth 2.0 protocol, which is based on central authentication at the SAP HANA User Authentication & Authorization (UAA) server that then vouches for the authenticated user's identity by means of a so-called OAuth Access Token. The current implementation uses as access token a JSON web token (JWT), which is a signed text-based token formatted according to the JSON syntax.

In a typical deployment scenario, your Python application will consist of several parts, which appear as separate application packages in your manifest file, for example:

- Database artifacts
  This application module (`<myAppName>/db/`) describes and defines the SAP HANA database content
- Application logic
  This application module (`<myAppName>/js/`) contains the application logic: code written in Python. This module can make use of this *XS Advanced Container Security API* for Python).
- UI client
  This application module (`<myAppName>/web/`) is responsible for the UI layer; this module can make use of the application router functionality (defined in the file `xs-app.json`).

> **Note**
>
> The application logic written in Python and the application router should be bound to one and the same UAA service instance so that these two parts use the same OAuth client credentials.

To use the capabilities of the XS Advanced container security API, add the package "`sap_xssec`" to the dependencies section of your application's `package.json` file. To enable tracing, set the environment variable `DEBUG` as follows: `DEBUG=xssec:*`.

Usage

To use the XS advanced container security API, it is necessary to provide a JWT token. The examples below rely on users and credentials that you should substitute with the ones in your context. The typical use case for calling this API lies from within a container when an HTTP request is received. The authorization header (with keyword "`bearer`") already contains an access token; you can remove the prefix bearer and pass the remaining string to the API as "`access_token`", as illustrated in the following example:

> **Sample Code**

```
from sap import xssec
from cfenv import AppEnv
env = AppEnv()
uaa_service = env.get_service(name='<uaa_service_name>').credentials
security_context = xssec.create_security_context(access_token, uaa_service)
```

The example above uses the Python module `cfenv` to retrieve the configuration of the UAA service instance.

The creation function `xssec.create_security_context` is used for an end-user token (for example, with "grant_type" `password` or `authorization_code`) where user information is expected to be available within the token and, as a consequence, within the security context. `create_security_context` also accepts a token of grant type `client_credentials`, which enables the creation of a limited security context where certain functions are not available, as described in more detail in the Security API documentation below.

Container Security API

The complete details of the XS advanced Container Security API including all parameters and options are available in the README file included in the `sap_xssec` package.

Container Security API

API	Description
`create_security_context`	Creates the "security context" by validating the received access token against credentials put into the application's environment via the UAA service binding

API	Description
`get_logon_name`	Returns the user's logon name *
`get_given_name`	Returns the user's given (first) name *
`get_family_name`	Returns the user's family name *
`get_email`	Returns the user's email address *
`check_local_scope`	Checks a scope that is published by the current application in the `xs-security.json` file. Returns "True" if the checked scope is included in the user's authorization scopes (otherwise "False")
`check_scope`	Checks a scope that is published by an application. Returns "True" if the checked scope is included in the user's authorization scopes (otherwise "False")
`get_token`	Returns a token that can be used to connect to the SAP HANA database. If the token that the security context has been instantiated with is a foreign token (meaning that the OAuth client contained in the token and the OAuth client of the current application do not match), "`None`" is returned instead of a token.
`get_hdb_token`	Returns a token that can be used to connect to the SAP HANA database. If the token that the security context has been instantiated with is a "foreign" token (the OAuth client contained in the token and the OAuth client of the current application do not match), "`None`" is returned instead of a token.
`request_token_for_client`	Requests a token with `grant_type=user_token` from another client. The requesting client must also have `grant_type=user_token`, and the current user token must include the authorization scope `uaa.user`.
`has_attributes`	Returns "`True`" if the token contains any XS advanced user attributes; otherwise "`False`".
`get_attribute`	Returns the attribute exactly as it is contained in the access token. If no attribute with the given name is contained in the access token, "`None`" is returned. If the token that the security context has been instantiated with is a foreign token (the OAuth client contained in the token and the OAuth client of the current application do not match), "`None`" is returned regardless of whether the requested attribute is contained in the token or not.
`get_additional_auth_attr`	Requires the `name` of the additional authentication attribute requested. Returns the requested attribute exactly as it is contained in the access token. If no attribute with the given `name` is contained in the access token, `None` is returned.

API	Description
`is_in_foreign_mode`	Returns "`True`" if the token, that the security context has been instantiated with, is a foreign token that was not originally issued for the current application, otherwise "`False`".
`get_subdomain`	Returns the subdomain that the access token has been issued for.
`get_clientid`	Returns the client id that the access token has been issued for.
`get_identity_zone`	Returns the identity zone that the access token has been issued for
`get_expiration_date`	Returns the expiration date of the access token as a Javascript "Date" object
`get_clone_service_instance_id`	Returns the id of the cloned service instance, if the XSUAA broker plan is used
`get_grant_type`	Returns the grant type of the JWT token, for example: `authorization_code`, `password`, `client_credentials`, etc.

> **i** Note
>
> * Not available for tokens of grant type `client_credentials`

sap_cf_logging

This is a lightweight library for Python applications that is compatible with the Python logging module, emits JSON log and supports request instrumentation.

Installation

> **i** Note
>
> The `sap_cf_logging` library is not included in SAP's `XS_PYTHON` archive; it is an open-source library that is developed by SAP and published on the Python Package Index (Pypi). To install "`sap_cf_logging`", you must use the standard Python tools.

To install the packages run the following command in your command line interface:

```
pip install sap_cf_logging
```

Usage

To set up your application, the logging library needs to be initialized. Depending on you application type, different initialization is used. You should usually do this in your application entrypoint.

For CLI applications you just need to call `cf_logging.init()` once to configure the library. The library will try to configure future loggers to emit logs in JSON format.

After installation is complete you need to perform the following actions:

1. Import the `cf_logging` library and set up Flask logging on the application.

   Sample Code

   Flask

   ```
   from sap.cf_logging import flask_logging
   app = flask.Flask(__name__)
   flask_logging.init(app, logging.INFO)
   ```

2. Use Python's logging library.

   Sample Code

   ```
   @app.route('/')
   def root_route():
       logger = logging.getLogger('my.logger')
       logger.info('Hi')
       return 'ok'
   ```

3. Note the logs generated by the application.
4. Set up Sanic on the application.

   Sample Code

   ```
   import sanic
   import logging
   from sanic.response import HTTPResponse
   from sap.cf_logging import sanic_logging
   from sap.cf_logging.core.constants import REQUEST_KEY,
   app = sanic.Sanic('test.cf_logging')
   sanic_logging.init(app)
   @app.route('/')
   async def two(request):
       extra = {REQUEST_KEY: request}
       logging.getLogger('my.logger').debug('Hi', extra = extra)
       return HTTPResponse(body='ok')
   ```

   Note

   With Sanic you need to pass the request with an extra parameter in the logging API. This is necessary to obtain the *correlation_id* generated at the beginning of the request or fetched from the HTTP headers.

hdbcli

The library implements PEP 249 - The Python Database API Specification v2.0. For more information about connecting to the SAP HANA database with hdbcli, see *hdbcli* in the *Related Information* below.

Related Information

hdbcli
pip download
The Python Package Index (PyPI)

7.3.3 Tutorials: Setting Up the Python Run-Time Environment in XS Advanced

Tutorials that show you how to set up Python applications for XS advanced.

This section includes a selection of tutorials that show you how to set up various basic but important aspects of the application that you plan to deploy on the XS advanced Python run time.

> **i Note**
>
> The tutorials in this section focus on tools provided with the XS advanced command-line client.

Prerequisites

Before you start the tutorials in this section, bear in mind the following prerequisites:

- SAP HANA XS advanced model must be installed on your SAP HANA system and you can log in to it using the XS advanced command-line client. For more information, see *Related Information* below.
- Python is installed locally. If you use a proxy to connect to the Internet, you might need to configure it. For more information about proxy configuration settings for pip, see *pip Documentation* in the *Related Information* below.
- SAP Python packages
  The archive (`XS_PYTHON<version>.SAR`) is available on the SAP HANA media and the SAP Marketplace (SMP); it includes a selection of Python packages developed by SAP for use with the Python applications running in the XS advanced run time.

Related Information

Working with the XS Advanced Command-Line Client [page 75]
pip Documentation
Standard Python Packages for XS Advanced [page 783]
Download and Consume Python Libraries [page 782]

7.3.3.1 Set up the Python Run Time

This topic guides you through setting up the Python run time in XS advanced.

Context

To run Python applications in XS advanced, you have to upload a Python runtime for the application to run on. This is one major difference to the approach taken so far with Node.js and Java. To check if a Python runtime is already set up and see which versions of Python this runtime supports, run the command `xs runtimes` in an XS CLI shell. If no entry for Python is visible in the list of available run-time environments, you must set up a Python run time environment, for example, by uploading a prebuilt Python and ensuring that *pip* and `setuptools` are included in the build result.

> **i** Note
>
> For more detailed information about building Python, see *Building the Python run Time from Sources* in *Related Information* below.

Procedure

1. Download the latest sources for Python 3.x from the "download sources" link in *Related Information* below and extract the downloaded sources into a local directory.
2. From the source's directory, run the `configure` command.

   ```
   ./configure --prefix=python_dir --exec-prefix=python_dir
   ```

   > **i** Note
   >
   > `python_dir` is an empty directory where the Python run time will be installed. The path to `python_dir` must be absolute.

   To complete the build you will need to install the *devel* versions of the missing packages and run the `./configure` command again.
3. Run the `make` command in a command shell:

   ```
   make –j4 && make altinstall
   ```

4. After a successful build, add the new Python run time to XS advanced, for example, by running the `xs create-runtime` command.

   ```
   xs create-runtime -p python_dir
   ```

   > **i** Note
   >
   > The `xs create-runtime` command requires XS advanced administrator permissions (for example, user `XSA_ADMIN`).

5. Check if the Python run time has been deployed correctly.

   ```
   xs runtimes
   ```

   The `xs runtimes` command displays a list of the currently available application run-time environments.

Related Information

Building from Sources and Deploying the Python Run Time to XS Advanced [page 780]
https://devguide.python.org/setup/
https://www.python.org/downloads/source/

7.3.3.2 Create a Python Application

This topic will guide you through creating a simple Python application.

Procedure

1. Create a new directory called `pyapp`.
2. Specify the run time version your application needs, by creating a `runtime.txt` file in the `pyapp` directory, with the Python version you have deployed in XS advanced.

   Sample Code

   `runtime.txt`

   ```
   python-3.6.3
   ```

   You can check the Python version with the xs CLI, for example, using the command `xs runtimes`.
3. Specify Flask as a dependency for the Python application.

   Since the Python application is a Web server utilizing the Flask Web framework, you must specify Flask as a dependency, for example, by creating a `requirements.txt` file in the `pyapp` directory, with the following content:

   Source Code

   Flask Dependency in `requirements.txt`

   ```
   Flask==0.12.2
   ```
4. Create a `manifest.yml` file in the `pyapp` directory, which describes the Python application and how to deploy and run it in XS advanced.

Source Code

manifest.yml

```
---
applications:
- name: pyapp
  host: pyapp
  path: .
  command: python server.py
```

Caution

Cutting and pasting from this code block will probably cause white-space formatting issues in the `yaml` file. After pasting, remember to adjust the indenting to conform to valid yaml.

5. Create a `server.py` file in the `pyapp` directory, which will contain the application logic.

   In this example, the server returns a `Hello World` message, when requested.

Source Code

server.py

```
import os
from flask import Flask
app = Flask(__name__)
port = int(os.environ.get('PORT', 3000))
@app.route('/')
def hello():
    return "Hello World"
if __name__ == '__main__':
    app.run(port=port)
```

6. Push the Python application to XS advanced.

   ```
   xs push
   ```

7. Check the URL of the deployed Python application.

   After application deployment is complete, use the command `xs apps` to display a list of all deployed applications and the corresponding URL at which each application is reachable. Locate the application `pyapp` and note the assigned URL.

8. Open a new Web browser and paste the URL of `pyapp` into it.

   `pyapp` displays the following message:

   ```
   Hello World
   ```

7.3.3.3 Consume XS Advanced Services

Procedure

1. Create an instance of the SAP HANA service with the following command:

   ```
   xs create-service hana hdi-shared myhana
   ```

2. Bind the service instance to the application. Modify the `manifest.yml`:

Source Code

`manifest.yml`

```
---
applications:
- name: pyapp
    host: pyapp
    path: .
    command: python server.py

    services:
        - myhana
```

Caution

Cutting and pasting from this code block will probably cause white-space formatting issues in the `yaml` file. After pasting, remember to adjust the indenting to conform to valid yaml formatting.

3. Add a module that will help read the service settings from the environment. Modify `requirements.txt`:

Source Code

`requirements.txt`

```
Flask==0.12.2
cfenv==0.5.3
```

4. Consume the XS advanced service `hana`.

   Modify the `server.py` file to `get` the service `hana`, as illustrated in the following example:

Source Code

`server.py`

```
import os
from flask import Flask
from cfenv import AppEnv
app = Flask(__name__)
env = AppEnv()
port = int(os.environ.get('PORT', 3000))
hana = env.get_service(label='hana')
@app.route('/')
def hello():
    return "Hello World"
if __name__ == '__main__':
app.run(port=port)
```

Results

The SAP HANA service instance `myhana` is now bound to the application, and you can connect to it.

Related Information

Connect to SAP HANA [page 797]

7.3.3.4 Connect to SAP HANA

Use the `hdbcli` module to connect to SAP HANA.

Procedure

1. Modify the `requirements.txt` file:

   Source Code

   `requirements.txt`

   ```
   Flask==0.12.2
   cfenv==0.5.3
   hdbcli
   ```

2. Use the module to connect to SAP HANA and execute a simple query.

   Modify `server.py`:

   Source Code

   `server.py`

   ```
   import os
   from flask import Flask
   from cfenv import AppEnv
   from hdbcli import dbapi
   app = Flask(__name__)
   env = AppEnv()
   port = int(os.environ.get('PORT', 3000))
   hana = env.get_service(label='hana')
   @app.route('/')
   def hello():
       conn = dbapi.connect(address=hana.credentials['host'],
   port=int(hana.credentials['port']), user=hana.credentials['user'],
   password=hana.credentials['password'])

       cursor = conn.cursor()
       cursor.execute("select CURRENT_UTCTIMESTAMP from DUMMY", {})
       ro = cursor.fetchone()
       cursor.close()
       conn.close()

       return "Current time is: " + str(ro["CURRENT_UTCTIMESTAMP"])
   if __name__ == '__main__':
       app.run(port=port)
   ```

> **Note**
>
> For more information about the API, see *Related Information* below.

3. Provide the application package dependencies.

   Create a folder called `vendor` in the root folder of the application. This is the target location for the dependent Python package that are downloaded with `pip download`. For more information about the `pip download` command and the `vendor` folder, see *Download and Consume Python Libraries* in the *Related Information* below.

4. Push the Python application to XS advanced and request its URL.

   ```
   xs push <pyapp>
   ```

   ```
   xs apps
   ```

   Locate the URL of the Python application and paste it into a Web browser; you should see the current SAP HANA time.

   ```
   Current time is: 14:38:25
   ```

Related Information

Python Support
Download and Consume Python Libraries [page 782]

7.3.3.5 Set up Authentication

To set up authentication, you use the application router and the XS User Account and Authentication (UAA) service.

Procedure

1. Create a service instance of the XS UAA service, with the following command:

   ```
   xs create-service xsuaa default myuaa
   ```

2. Create a new directory for the application router.

   Inside the new application-router directory, create the following files:

   - `manifest.yml`
     The application's manifest file.
   - `package.json`
     The list of packages that `pyapp` requires.
   - `xs-app.json`

The application descriptor which is used to configure the application router.

a. Create and configure the application manifest file `manifest.yml`:

   In this example, you bind the UAA service to the application router and create a destination for the Python application specifying you want an authentication token to be sent.

   Source Code

   `manifest.yml`

```
---
applications:
- name: web
  path: .
  env:
    destinations: >
      [
        {
          "name":"pyapp",
          "url":"https://host:port_of_python_app",
          "forwardAuthToken": true
        }
      ]
  services:
    - myuaa
```

   Caution

   Cutting and pasting from this code sample will probably cause white-space formatting issues in the `yaml` file. After pasting, remember to adjust the indenting to conform to valid yaml formatting.

b. Create and configure the package-dependencies file `package.json`.

   Source Code

   `package.json`

```
{
  "name": "myapp-entry-point",
  "scripts": {
    "start": "node node_modules/@sap/approuter/approuter.js"
  },
  "dependencies": {
    "@sap/approuter": "^3.0.1"
  }
}
```

   Note

   You list the application router as a dependency and use it for the authentication.

c. Create and configure the application descriptor `xs-app.json` for the Python application `pyapp`:

   Source Code

   `xs-app.json`

```
{
  "routes": [
    {
```

```
      "source": "^/pyapp/(.*)$",
      "target": "$1",
      "destination": "pyapp"
    }
  ]
}
```

In this example, you redirect all traffic to `/pyapp/` to the Python application `pyapp`. For more information about routes and destinations in XS advanced, see *Maintaining XS Advanced Application Routes and Destinations* in the *Related Information* below.

3. Push the Python application `pyapp` to XS advanced.

   ```
   xs push pyapp
   ```

4. Check the URL of the application router:

   After deployment is complete, use the command `xs apps` to display a list of all deployed applications and the corresponding URL at which each application is reachable. Locate the URL of the application router for the `pyapp` application and note the assigned URL.

5. In a Web browser, navigate to the URL of the application router: `<URL_of_approuter>`/pyapp/.

   You should be prompted to log in, and then you should see the current SAP HANA time displayed by the Python application.

6. Enter the credentials of a valid user. If no custom identity provider (IDP) is configured, you should use the credential of a known SAP HANA user.

   You should see a message similar to the following in the browser:

   - *Application user:* `<login user>`
     The name of the user who logs on to the application.
   - *HANA user:* `<db user>`
     The technical user used to connect to SAP HANA.

> **i Note**
>
> An application user is not the same as a database user.
>
> Since the application `pyapp` and the application `web` are both bound to the same UAA service instance, authentication is handled by the `web` application.

Related Information

Maintaining XS Advanced Application Routes and Destinations

7.3.3.6 Set up Authorization

To control access to the content you serve, use the `xssec` security library.

Procedure

1. Modify the `requirements.txt` file to include the `xssec` library:

   Source Code

   `requirements.txt`

   ```
   Flask==0.12.2
   cfenv==0.5.3
   hdbcli
   xssec==1.0.0
   ```

2. Modify the `manifest.yml` file and bind the same UAA service:

   Source Code

   `manifest.yml`

   ```
   ---
   applications:
   - name: pyapp
     host: pyapp
     path: .
     command: python server.py
     services:
       - myhana
       - myuaa
   ```

   Caution

   Cutting and pasting from this code sample will probably cause white-space formatting issues in the resulting `yaml` file. After pasting, remember to adjust the indenting to conform to valid yaml formatting.

3. Modify the `server.py` file to use the security library:

   Source Code

   server.py

   ```
   import os
   from flask import Flask
   from cfenv import AppEnv
   from flask import request
   from flask import abort
   import xssec
   from hdbcli import dbapi
   app = Flask(__name__)
   env = AppEnv()
   port = int(os.environ.get('PORT', 3000))
   ```

```
hana = env.get_service(label='hana')
uaa_service = env.get_service(name='myuaa').credentials
@app.route('/')
def hello():
    if 'authorization' not in request.headers:
        abort(403)
    access_token = request.headers.get('authorization')[7:]
    security_context = xssec.create_security_context(access_token,
uaa_service)
    isAuthorized = security_context.check_scope('openid')
    if not isAuthorized:
        abort(403)
    conn = dbapi.connect(address=hana.credentials['host'],
port=int(hana.credentials['port']), user=hana.credentials['user'],
password=hana.credentials['password'])
    cursor = conn.cursor()
    cursor.execute("select CURRENT_UTCTIMESTAMP from DUMMY", {})
    ro = cursor.fetchone()
    cursor.close()
    conn.close()
    return "Current time is: " + str(ro["CURRENT_UTCTIMESTAMP"])
if __name__ == '__main__':
    app.run(port=port)
```

4. Modify the application's package dependencies to include the security library in the download to the `vendor` folder.

   Follow the steps described in *Download and Consume Python Libraries* in the *Related Information* below.

5. Try to access the Python application directly, for example, using the application URL displayed by the `xs app pyapp --urls` command; you should see an HTTP 403 error.

   If you however access the Python application through the application router, you should see the current SAP HANA time, provided you have the security scope `openid` assigned to XS advanced user who is calling the URL and logging on.

Related Information

Download and Consume Python Libraries [page 782]

7.3.3.7 Set up Logging

To add logging support, use the SAP Python library `sap_cf_logging`.

Procedure

1. Modify the `requirements.txt` file to include the logging module:

Source Code

requirements.txt

```
Flask==0.12.2
cfenv==0.5.3
hdbcli
xssec==1.0.0
sap_cf_logging==3.0.0
```

2. Modify `server.py` file to use the logging library and log some messages when the application is accessed:

Source Code

server.py

```
import os
import logging
from flask import Flask
from cfenv import AppEnv
from flask import request
from flask import abort
import xssec
from hdbcli import dbapi
from cf_logging import flask_logging
app = Flask(__name__)
env = AppEnv()
flask_logging.init(app, logging.INFO)
port = int(os.environ.get('PORT', 3000))
hana = env.get_service(label='hana')
uaa_service = env.get_service(name='myuaa').credentials
@app.route('/')
def hello():
    logger = logging.getLogger('route.logger')
    logger.info('Someone accessed us')
    if 'authorization' not in request.headers:
        abort(403)
    access_token = request.headers.get('authorization')[7:]
    security_context = xssec.create_security_context(access_token,
uaa_service)
    isAuthorized = security_context.check_scope('openid')
    if not isAuthorized:
        abort(403)
    logger.info('Authorization successful')
    conn = dbapi.connect(address=hana.credentials['host'],
port=int(hana.credentials['port']), user=hana.credentials['user'],
password=hana.credentials['password'])
    cursor = conn.cursor()
    cursor.execute("select CURRENT_UTCTIMESTAMP from DUMMY", {})
    ro = cursor.fetchone()
    cursor.close()
    conn.close()
    return "Current time is: " + str(ro["CURRENT_UTCTIMESTAMP"])
if __name__ == '__main__':
    app.run(port=port)
```

3. Modify the application's package dependencies to include the logging library in the download to the `vendor` folder.

   Follow the steps described in *Download and Consume Python Libraries* in the *Related Information* below.

4. Push the application and access its URL.

```
$ xs push
```

```
$ xs app pyapp --urls
```

5. Verify the message you logged is part of the application logs:

   Display the most recent messages logged in the `pyapp` application's logs:

   ```
   xs logs pyapp --recent
   ```

Related Information

Download and Consume Python Libraries [page 782]

7.3.3.8 Troubleshoot XS Advanced Python Applications

Many issues can be analyzed using information stored in the application's log files. You can use the following methods to change the logging levels of the application:

- Set the environment variable `XS_APP_LOG_LEVEL` to "debug".

  > **i Note**
  >
  > You must restart the application for this to take effect.

- Use the `xs set-logging-level` command, as illustrated in the following example:
  `xs set-logging-level <application-name> '*' debug`

  > **i Note**
  >
  > You do not need to restart the application for this configuration change to take effect.

Related Information

Tutorials: Setting Up the Python Run-Time Environment in XS Advanced [page 792]

7.4 Custom Run-Time Environments in SAP HANA XS Advanced

Create a custom run-time environment for an application written in a language that is not supported by default by XS advanced.

SAP HANA XS advanced includes a list of standard predefined run-time environments (for example Java, Node.js, Tomcat, TomEE) that can be downloaded from a run-time “store” included with XS advanced during

the compilation phase of the corresponding buildpack. These standard default run-time environments are primarily intended for use with the buildpacks included by default with XS advanced.

However, you can also create and run a custom run-time environment of your own in XS advanced, so that you can deploy applications written using languages such as Python or PHP, which are not supported by any of the default run-time environments included in the XS advanced run-time store.

> ⚠ Caution
>
> SAP does not provide support for custom language, buildpack, or run-time scenarios.

Applications deployed to a custom run-time environment in XS advanced do not have automatic access to (or use of) some important features that are built into and supported by the XS advanced framework, including (but not limited to): authentication and security, logging and auditing, and connections to the database. If you deploy an application to a custom run-time environment in XS advanced, you must configure these components manually for the custom application.

If you want to deploy an application written in a language for which XS advanced does not include a corresponding run-time environment (for example, Python or PHP), you must provide a suitable run-time "buildpack". Buildpacks are a convenient way of packaging framework, dependencies, and run-time support for an application at deployment time. The buildpack defines what happens to your application when it is deployed and, in addition, how it is executed after deployment.

Buildpacks are stored in the central blob store for XS advanced along with run-time environments and applications. To upload a buildpack to the central XS advanced blob store, you can use the XS command-line interface (CLI), for example, the `xs create-buildpack` command.

> ! Restriction
>
> Access to the blob store is restricted to users with administrator rights.

Related Information

Custom Buildpacks in XS Advanced [page 808]
XS CLI: Buildpacks

7.4.1 Best Practices for Deploying Applications in a Custom Run-Time Environment

A list of things to bear in mind when coding an application that you plan to deploy in a custom XS advanced run-time environment.

Overview

If you plan to develop an application that will be deployed to a non-standard run-time environment in SAP HANA XS advanced, for example, Python or PHP, it is important to develop a strategy that facilitates the

creation (or migration) and deployment of the application to the custom run-time environment. For example, application logic that cannot be easily ported to one of the supported languages (Node.js, XS JavaScript, Java) should be contained in its own module.

You also need to understand the nature of the custom language you want to use and support:

- Compiled language (for example, C++, Swift)
- Interpreted language (for example, JavaScript, PHP, Perl)
- Runs in a Virtual Machine (for example, Java)
- Requires run-time support (for example, ASP.NET)

The primary function of your custom application module is to service HTTP requests that come exclusively from the application router module. The work that these requests require should be a natural fit for the programming language that the buildpack is supporting. If your module needs to access other services, for example, a data store, it can find details required for connecting to the service by inspecting the environment that the deployment process provides.

> → Tip
>
> Keep in mind that your application module is executing in isolation; the primary interface to the outside world is through the HTTP server interface it provides.

Deployment Environment

It is important to understand how the custom application will be deployed, for example, which parts of the application need to scale or be able to respond to a fail over. Since details of the deployment environment are often only available at run time, consider the following questions and points:

- External dependencies
  Which, if any, external services are required?
- Host names and/or ports
  Each module must be identified uniquely in the system. Typically, this is done by either assigning a port or a host name to the module. Since it is not always possible to obtain details of the deployed environment before deployment (or if details of the deployed environment change), it is essential to avoid hard-coding any host names or ports in your application; all ports and host names must be read from the environment.

  > i Note
  >
  > In a multi-tenant PaaS environment (for example, SAP Cloud Platform) where many customers have multiple accounts and many deployed MTAs, host names are the only way to insure uniqueness.
- Configuration details
  Since configuration information is provided as run-time environment variables, the application-programming language must be able to read the run-time environment and parse JSON in order to obtain this configuration information.
- Security and encryption
  The application language must also include the libraries and code required to provide an HTTPS server, which can be used to respond to requests from the application router. In addition, the logic provided must be able to handle security certificates, encryption, and uncompressing data payloads.

The application language must also be able to decrypt, parse, and verify JSON Web Tokens (JWT), for example, so that an application can refuse to respond to any request that does not provide a JWT in the request header (or provides an invalid one).
If the custom application needs to communicate with other modules, it might be necessary to provide an HTTPS client, as well.

- Environment variables
  Your application module must be able to read and parse execution environment variables. The application-deployment process initializes the environment in such a way as to make available any required information so that the module can make connections to other services and interact with them. For example, JDBC is primarily used to connect to the data store, but other services are also available, for example, the job scheduler. An application module can also use HTTP to communicate with other modules, however, it will need to pass on the JWT in the request header, if necessary, so that the receiving module can also perform the same verification checks. If an application module needs to interact with an external service, it can perform outbound HTTP connections as well.
- Logging and tracing
  An application module does not usually have access to a writeable file system, so it is not recommended to assume that one will be available. This is especially important in a multi-tenant environment where your applications are running inside your own account. Often logging and (or) tracing service is provided by the system, so an application module should at least be able to log its activity, for example, to record when it encounters conditions that produce a warning or an error.

Best Practices

To understand which components and dependencies are required to ensure the successful deployment of applications to a custom run-time environment in XS advanced, for example, Python, or PHP, bear in mind the following important points:

- Task focus
  Try and organize your custom application's modules into single (or related) tasks that require little or no communication with other modules. Choose a programming language that is best suited for the selected tasks.
- Portability
  Keep deployment and portability in mind as you develop. Since it is often not possible to know if the target system for deployment will be a dedicated system configured for port assignment or a multi-tenant system configured to use host names to distinguish module identity, it is essential to ensure that any configuration details and settings data is held in deployment descriptors so that it can be read dynamically during module start up.
- Database access
  Try to confine access to the persistence store (database) to one module and ensure this dedicated module fields all database requests from other modules, if needed. This keeps the data access unified in one place. Since these modules are all running in the same process space, the overhead is minimal anyway.
- Target deployment environment
  For compiled languages (for example, C++ or Swift) or languages that have dependencies to specific static or shared libraries, try to develop your custom application in an environment that, in configuration terms, is as close as possible to the target deploy environment.

> → Tip
>
> This is less important for interpreted languages (for example, PHP or Perl) or languages such as Java that run in a Virtual Machine (VM).

- Testing and debugging
  If you have full control of the deployed environment (for example, in an on-premise scenario), consider setting up an HTTP server and a language run-time environment so that you can test logic and debug locally.
- Build/Deployment environment
  If possible, re-create as closely as possible a local environment for the language you are working in.
- Mock data/services
  Code your module in a way that it can be tested on the command line with mock data representing the services the module will eventually connect to. In this way, you can iterate quickly and focus on the logic of your module and avoid the time-consuming edit-deploy-test cycle.
- Routing and authentication
  Test each module **without** routing or any requirement for User Account and Authentication (UAA) services, for example, by using mock data that is provided directly in the target environment. If possible perform the tests locally and work out all the logical bugs (parsing JSON, string manipulation, iterating over data, and so on). Only after successful tests for the individual modules, should you consider connecting them to other modules, at which point you can set up the MTA's deployment descriptor (`mtad.yaml`) to configure and coordinate inter-module dependencies.
- Buildpacks
  When developing buildpacks, use `chroot` to help resolve any dependencies between environment and libraries. The Unix/Linux `ldd` command can be used to display a list of dependent libraries for a particular binary.
- Deployment target
  Test deployment of the custom application in multiple spaces and know where to find deployment logs and how to read and interpret them.

Related Information

Custom Run-Time Environments in SAP HANA XS Advanced [page 804]

7.4.2 Custom Buildpacks in XS Advanced

Custom buildpacks define the conditions for the deployment of an application to a custom run-time environment in XS advanced.

buildpacks are a convenient way of configuring the packaging framework and run-time support for an application. The buildpack defines what happens to your application when deployed in XS advanced and, in addition, how it is executed after deployment.

SAP HANA XS advanced includes a list of standard predefined run-time environments (for example Java, Node.js, Tomcat) that can be downloaded from a run-time “store” during the compilation phase of the corresponding buildpack. These standard run-time environments are primarily intended for use with the

buildpacks included by default with XS advanced. However, you can also create and run a custom run-time environment in XS advanced, so that you can deploy applications written using languages such as PHP, which is not supported by any of the default run-time environments included with XS advanced.

> **Note**
>
> Applications deployed to a custom run-time environment in XS advanced cannot by default use some of the integrated features built into and supported by the XS advanced framework, including (but not limited to): authentication and security, logging and auditing, and connections to the database, all of which would need to be configured manually for the custom application. This would all need to be implemented in the specific programming language and (or) supplied by the run-time environment provided by the buildpack.

If you want to deploy an application written in a language for which XS advanced does not included a corresponding run-time environment (for example, PHP), you must provide a suitable run-time buildpack. A simple buildpack contains the following scripts, which are typically located in the directory `/bin/`.

- Build-pack detection
  The `bin/detect` script is used to find a suitable buildpack to use to deploy an application.
- Compilation
  The `bin/compile` script assembles all the components that are required to deploy and run the application in the custom run-time environment.
- Deployment and execution
  The `bin/release` script is used to provide metadata for XS advanced; the metadata describes how and where the deployed application should be executed.

Buildpack Detection

The `bin/detect` script is used to find a suitable buildpack to use to deploy an application. If more than one buildpack is available, XS advanced uses a detection priority to find the appropriate buildpack, where priority 1 (one) is the highest priority and indicates the first buildpack to check for suitability. The `detect` script requires one argument that specifies the location of the build directory for the custom application. The build directory is the location to which the pushed (application) content is copied by the `xs push` command during deployment. If the `detect` script finds a suitable buildpack for the specified application, it returns the exit code "`0`" (success). If multiple buildpacks are installed for a particular language and a buildpack with a lower priority is desired, the required buildpack can be explicitly identified by name during application deployment, as illustrated in the following command:

> **Sample Code**

```
xs push myAPP -b <BUILDPACK_NAME>
```

> **Note**
>
> If the returned exit code is not "`0`", XS advanced attempts to find and use other buildpacks by calling their corresponding `bin/detect` scripts.

You can find out what buildpacks are installed in your deployment system and what priority is assigned to each buildpack during the dectection phase by running the `xs buildpacks` command, as illustrated in the following example:

Output Code

```
xs buildpacks
Getting buildpacks...
buildpacks            version     position   enabled   locked
-------------------------------------------------------------
sap_java_buildpack    <unknown>   1          true
nodejs_buildpack_op   3.0.0       2          true
python_buildpack      1.1.0       9          true
phpbp_buildpack       1.0.0       10         true
```

Buildpack Compilation

The `bin/compile` script builds the droplet run by the execution agent; the droplet contains all the components that are required to deploy and run the application in the custom run-time environment. The `compile` script requires a value for the following arguments:

- The build location (a directory path/name) for the application
- The location of the cache directory that the buildpack uses for temporary storage during the build process

> **Note**
>
> The platform provides the flexibility required to handle the execution of architecture-dependent applications.

If the architecture where the buildpack compiles the droplet is the same as the architecture where the instances run, you can configure the custom buildpack to detect the architecture during compile phase and do the compilation accordingly. For example, the buildpack could take an architecture-specific run time that is included a “local” archive (the archive you are putting together in this process). The buildpack could also fetch the required run time from an external repository. It is also possible for the buildpack to assemble different application binaries for different architectures. If necessary, you can configure the buildpack to compile the application sources, too.

Buildpack Deployment and Execution

The `bin/release` script is used to provide metadata for XS advanced; the metadata describes how the deployed application should be executed. The `release` script requires a value for the build location of the application (which was prepared in the compile step). The content of the file generated by the `release` script

be formatted according to YAML rules and, in addition, contain at least the information illustrated in the following example:

Sample Code

Buildpack Application Run-time Metadata

```
config_vars:
    name: <value>
default_process_types:
    web: <commandLine>
```

- `config_vars`
  A list of environment variables that will be set in the run-time environment where the application is deployed
- `<command>`
  The command used to start the deployed application; the command is run in the `work` directory prepared in the buildpack's compile phase.

Related Information

Custom Run-Time Environments in SAP HANA XS Advanced [page 804]

7.4.3 Create a PHP Buildpack for XS Advanced

A high-level overview of the steps required to build a custom buildpack for use in deploying a custom application (for example PHP) to the XS advanced run-time environment.

Prerequisites

- This example assumes that the target deployment environment is SUSE 12.3 or similar.
- The project directory in this example is called "`buildpacks`". The `buildpack` contains a subdirectory called `rootdir`. The `rootdir` directory contains the files associated with the use of the `chroot` command (for example, to determine build and library dependencies).

Context

The run-time environment is like an isolated file system and an execution environment that hosts the components of the buildpack. The application files deployed are added to the environment that the buildpack provides, and the instructions in the buildpack's compile script are run. In this example, we use PHP as the programming language.

> **! Restriction**
>
> The information included in this example is provided for illustration purposes only; it is intended to help you put together your own buildpack for use in your own custom environment.

Procedure

1. Set up the new buildpack components.

   Have a look at an existing (working) buildpack to get an example of the detect/compile/release scripts that you'll need to modify to accommodate your custom language. The existing buildpack will also provide much of the file-system files that the deployment run-time environment supplies.

   > **i Note**
   >
   > A buildpack typically includes a minimal set of system files. For this reason, it is highly likely that you will need to add any specific shared libraries that are required to support the execution of your programming language.

2. Build the executables.

   If your language is a compiled language, you will either need to build the executables on your local development machine or provide a script to build them on the target machine during application deployment (for example, using the buildpack's "compile" script). Since PHP is an interpreted language, the module's logic is expressed in various PHP scripts that are included in the main "execution" script (`index.php`) that is executed by the buildpack during application deployment. The most important component that our buildpack needs to provide is a functional PHP interpreter.

3. Build the PHP interpreter.

   It is possible to try to use a binary version of the PHP interpreter, but you will need to ensure that all the required shared libraries are available to it.

   Depending on the language, you may have to implement it's packager as well. For example, both Perl and Node.js have a dedicated packager you can use. PHP, however, uses the concept of modules that provide various services or enable various libraries. When building the PHP interpreter for the sample buildpack, it makes sense to start by trying to build the interpreter as statically as possible and only include those features that are needed.

   > **i Note**
   >
   > This involve getting the source code from PHP.net and compiling it with the desired features enabled.

4. Test the creation of the custom buildpack.

   If possible, try to create the custom buildpack on the target server environment, where the build environment is as close as possible to the deployed run-time environment.

   > **i Note**
   >
   > If you log in to the run-time system remotely, your login shell has access to the remote system's entire environment including the system `<path>` that is used for locating files and libraries. Bear in mind that files and libraries found in the build environment (because they are in locations configured in the

system <path>) might not necessarily be found in the “real” but more isolated context of the buildpack running in the XS advanced run time. Dependencies to shared libraries that work in the target server environment might not work when you execute the buildpack.

In Unix and Linux environments, you can use the `chroot` command to simulate the isolated execution environment of the buildpack. To ensure you have all the components required to create your PHP buildpack, perform the following steps:

a. Check the version of the PHP binary available in the `chroot` environment.

```
sudo chroot rootdir

cd bin/php/bin

./php -v

PHP 5.6.22 (cli) (built: Jun  7 2016 21:43:46)
Copyright (c) 1997-2016 The PHP Group
Zend Engine v2.6.0, Copyright (c) 1998-2016 Zend Technologies
```

The output of the `php -v` command should provide version information only; it any error messages are displayed indicating dependencies to missing shared libraries, you will need to investigate and resolve the errors as described in the following steps.

b. Exit the `chroot` shell.

c. Run the `ldd` command on the PHP binary to list any shared dependencies, as illustrated in the following example:

```
cd rootdir/bin/php/bin/php
ldd ./php

  linux-vdso.so.1 (0x00007ffe3b3ca000)
  libcrypt.so.1 => /lib64/libcrypt.so.1 (0x00007f5c23302000)
  libresolv.so.2 => /lib64/libresolv.so.2 (0x00007f5c230eb000)
  libpng12.so.0 => /usr/lib64/libpng12.so.0 (0x00007f5c22ec0000)
  libz.so.1 => /lib64/libz.so.1 (0x00007f5c22caa000)
  libcurl.so.4 => /usr/lib64/libcurl.so.4 (0x00007f5c22a44000)
  [...]
```

d. Copy the missing shared libraries into the correct location in the `chroot` buildpack environment, for example, `/usr/lib64`.

```
cp /usr/lib64/libpng12.so.0 rootdir/usr/lib64
```

e. Check the PHP binary again

5. Package the directory structure containing the components required for your buildpack.

Package your directory in such a way that, on deployment, the directory contents are extracted into the target run-time environment based on the instructions in the buildpack's `compile` script. You can then install the buildpack, for example, using the `xs create-buildpack` command.

Sample Code

```
cd buildpacks/rootdir

tar czvf ../php-buildpack/runtime/PHP.TGZ .

cd buildpacks
```

6. Install the custom buildpack in XS advanced.

   You can install the buildpack using the following command:

   ```
   xs create-buildpack <myBuildpackName> <Path-to-BuildPack> <position> --enable
   ```

   For example, to create a buildpack called `phpbp` using a package called `php-buildpack.zip` with position "10" and make the buildpack available, use the following command:

   ```
   xs create-buildpack phpbp php-buildpack.zip 10 --enable
   ```

   → Tip

   If multiple buildpacks are available, they are sorted for detection purposes according to `<position>` (priority), where position 1 (one) is the highest priority. A buildpack with position 1 (one) is inspected first during any detection process.

   Output Code

   ```
   xs buildpacks

   Getting buildpacks...

   buildpacks            version     position   enabled   locked
   ------------------------------------------------------------
   sap_java_buildpack    <unknown>   1          true
   nodejs_buildpack_op   3.0.0       2          true
   phpbp                 1.0.0       10         true
   ```

   → Tip

   If a buildpack with the same name already exists, you can use the `xs update-buildpack` command to modify and overwrite it, for example:

   Sample Code

   ```
   xs update-buildpack phpbp -p php-buildpack
   ```

7. Create and push PHP application modules to the new PHP run-time environment you have created in XS advanced.

   When the PHP buildpack is installed in XS advanced, you can start creating PHP modules and pushing them to the deployed environment. During the deployment operation, your `index.php` file is detected and the PHP buildpack is used for your run-time environment.

   Sample Code

   ```
   xs push phpapp -p php-test
   ```

   i Note

   `phpapp` is the name of the application to deploy and `-p php-test` specifies the location of the (ZIP) archive or directory containing the application to deploy.

Related Information

Custom Run-Time Environments in SAP HANA XS Advanced [page 804]
Best Practices for Deploying Applications in a Custom Run-Time Environment [page 805]
XS CLI: Application Management

7.4.3.1 Buildpack Scripts for PHP Run-Time Environment in XS Advanced

Example scripts for a custom PHP buildpack.

This topic provides some simple examples of the three basic scripts required by a buildpack (in this case PHP) for use when deploying PHP application modules to the XS advanced run-time environment:

- `bin/compile` [page 815]
- `bin/detect` [page 816]
- `bin/release` [page 816]

> **Note**
>
> The scripts here are provided for illustration purposes only; you might need to adjust them to suit your specific requirements.

bin/compile

The following code snippets illustrates a simple `compile` script for use in a PHP buildpack:

> **Sample Code**
>
> Compile Script for a PHP Buildpack

```
#!/bin/bash
# bin/compile <build-dir> <cache-dir>
BUILD_DIR=$1
BIN_DIR=`dirname $0`
BUILD_PACK_DIR=`dirname ${BIN_DIR}`
echo "Extracting PHP into..."
echo ${BUILD_DIR}
cd ${BUILD_DIR}
tar xzvf ${BIN_DIR}/../runtime/PHP.TGZ
echo "Extracting PHP Done."
```

bin/detect

The following code snippets illustrates a simple `detect` script for use in a PHP buildpack:

Sample Code

Detect Script for a PHP Buildpack

```
bin/detect
#!/bin/bash
# bin/detect <build-dir>
if [ -f $1/index.php ];
    then exit 0
fi
exit 1
```

bin/release

The following code snippets illustrates a simple `release` script for use in a PHP buildpack:

Sample Code

Release Script for a PHP Buildpack

```
#!/bin/bash
# bin/release
cat <<EOF
---
config_vars:
addons:
default_process_types:
  web: ./bin/php/bin/php -S 127.0.0.1:\$VCAP_APP_PORT
EOF
```

Related Information

Create a PHP Buildpack for XS Advanced [page 811]
Best Practices for Deploying Applications in a Custom Run-Time Environment [page 805]
Custom Run-Time Environments in SAP HANA XS Advanced [page 804]

7.4.4 Available Build Packs

Display a list of all available build packs in your XS advanced system.

A selection of build packs are available for use in XS advanced, both on-premise and in Cloud Foundry on the SAP Cloud Platform. You can also create your own build packs for use on-premise, for example, for PHP.

> **Note**
>
> It is strongly recommended to specify the appropriate build pack for the application in the application manifest. This ensures that the application is handled by the build pack for which it is intended and tested. Specifying the appropriate build pack also speeds up application deployment, since only the specified build pack needs to be downloaded for use during the staging process.

To display the build packs available in your installed system, use the `[xs | cf] buildpacks` command in an XS CLI shell, as illustrated in the following example. The command prefix you use depends on the installed platform, for example, on-premise (`xs`) or Cloud Foundry (`cf`).

```
$ xs buildpacks
Getting buildpacks...
buildpacks              version   position   enabled   locked
-------------------------------------------------------------
sap_java_buildpack      1.4.1     1          true      false
sap_nodejs_buildpack    3.2.3     2          true      false
sap_python_buildpack    0.2.3     2          true      false
```

The following build packs are typically available for Cloud Foundry scenarios:

> **Note**
>
> Build pack versions differ depending on the your account configuration and release details.

```
$ cf buildpacks
Getting buildpacks...
buildpacks             position  enabled  locked   filename
-------------------------------------------------------------------------------
staticfile_buildpack   1         true     false    staticfile-buildpack-v1.4...
java_buildpack         2         true     false    java-buildpack-v4.8.zip
ruby_buildpack         3         true     false    ruby-buildpack-v1.7.9.zip
nodejs_buildpack       4         true     false    nodejs-buildpack-v1.6.14.zip
go_buildpack           5         true     false    go-buildpack-v1.8.15.zip
python_buildpack       6         true     false    python-buildpack-v1.6.6.zip
php_buildpack          7         true     false    php-buildpack-v4.3.47.zip
binary_buildpack       8         true     false    binary-buildpack-v1.0.15.zip
dotnet_core_buildpack  9         true     false    dotnet-core-buildpack-v1.0...
sap_java_buildpack     10        true     false    sap_java_buildpack-v1.6.15...
```

The following table lists the standard build packs available with XS advanced on-premise and in the Cloud:

Build Pack Availability in XS Advanced

	Target Platform	
Build Pack	**XS Advanced**	**Cloud Foundry**
`binary_buildpack` *	-	Yes
`dotnet_core_buildpack` *	-	Yes
`go_buildpack` *	-	Yes
`java_buildpack` *	-	Yes
`nodejs_buildpack` *	-	Yes

Build Pack	Target Platform	
	XS Advanced	Cloud Foundry
`php_buildpack` *	-	Yes
`python_buildpack`*	-	Yes
`ruby_buildpack` *	-	Yes
`sap_java_buildpack`	Yes	Yes
`sap_nodejs_buildpack`	Yes	-
`sap_python_buildpack`	Yes	-
`staticfile_buildpack` *	-	Yes

> **i** Note
>
> * Community-supported build packs.

Related Information

Create a PHP Buildpack for XS Advanced [page 811]
Custom Run-Time Environments in SAP HANA XS Advanced [page 804]
XS CLI: Buildpacks

7.5 Create the XS Advanced Application Package Descriptor

The package descriptor defines a JavaScript application's design-time and run-time dependencies in SAP HANA XS Advanced.

Prerequisites

- File name:
  `package.json`
- Location:
  Application source-file folder, for example, `package.json`

- Format
  JavaScript Object Notation (JSON)

Context

The build, deployment, and run-time dependencies of a JavaScript application are described in the `package.json` file. The `package.json` file is mandatory for JavaScript applications and it must be located in the JavaScript application's source-file folder, for example, `js/`.

To create an application package description for your new application project, perform the following steps:

Procedure

1. Create the application resource-file structure.

   For example, `/appname1/`
2. Create a sub-folder for the JavaScript source-files in the application root folder.

   For example, `/appname1/js`
3. Create the application package description file.

   The application descriptor file `package.json` must be located in the JavaScript source-files folder, for example, `/appname1/js`.
4. Define details of the application packages and any dependencies:

   The contents of the `package.json` file must follow the required syntax.

Related Information

The XS Advanced Application Package Descriptor [page 819]
XS Advanced Application Package Descriptor Configuration Syntax [page 822]

7.5.1 The XS Advanced Application Package Descriptor

A file whose contents describe the prerequisites and dependencies that apply to a JavaScript application in SAP HANA XS Advanced.

The dependencies of a JavaScript application are described in the `package.json` file. The `package.json` file is mandatory for JavaScript applications and must be located in the JavaScript application's source-file folder, for example, `/path/appname/js`.

Typically, the `package.json` file is created with the node.js command `npm init`, where you provide general information about the JavaScript application. Note that for XS JavaScript applications, you must also provide the following additional information:

- Dependency to XS JavaScript module (the XS compatibility layer)
- Dependency to the application router (see example below)
- Start command to specify node options

> **i Note**
>
> For illustration purposes, the following example includes dependencies to a ficticious github account (on "acme.com"); you must adapt any URLs to suit the local development environment and dependencies.

> **Sample Code**
>
> Application's Package Description (`package.json`)

```
{
  "name": "<application_name>",
  "description": "<application_description>",
  "version": "<application_version>",
  "repository": {
    "url": "git@github.acme.com:xs-samples/nodejs-hello-world.git"
  },
  "dependencies": {
    "@sap/xsjs": "^1.1.9"
    "@sap/approuter": ">=1.0.1 <2.0.1"
    "@sap/hdbext": "^1.0.2"
    "@sap/xssec": "^1.0.5"
  },
  "engines": {
    "node": ">=0.10.x <0.12"
  },
  "scripts": {
    "start": "node --max-old-space-size=400 --expose-gc main.js"
  }
}
```

A package description is required for both the `web/` and `db/` modules. In the `web/` module, the information in the `package.json` file is enable the startup of the application router; the contents of the `package.json` file located in the `db/` module are used to start the SAP HANA HDI deployment application.

SAP HANA Database Connections

To establish a connection to the database, an application must first look up the connection properties from the bound services and create an "hdb" client. The easiest way to do this is to add a dependency to `@sap/hdbext` to the application's `package.json` file, as illustrated in the following example:

> **Sample Code**

```
"dependencies": {
"@sap/hdbext": "1.0.2"
  },
```

Next, the application has to create a connection, as illustrated in the following example:

Sample Code

```
var xsConnection = require("@sap/hdbext");
var hdbclient = xsConnection.createConnection(function(error) { ... });
```

Connection Pools

The `@sap/hdbext` module includes a simple method for pooling connections. To use the connection-pool feature, you must first create the connection pool, as illustrated in the following example:

```
var pool = hdbext.getPool(hanaConfig, poolConfig);
```

Then you can acquire a connection client from the pool; the client is delivered in a callback, as illustrated in the following example:

```
pool.acquire(function(err, client) {});
```

If the connection client is no longer needed, it should be released back to the pool:

```
pool.release(client);
```

Security

For Node.js, the client security library is an `npm` module called `@sap/xssec`. If you want your application to use the security library `@sap/xssec`, you must add the dependency to `@sap/xssec` in the `dependencies` section of your application's package description file, `package.json`, as illustrated in the following example:

Sample Code

```
"dependencies": {
    "@sap/xssec": "^1.0.5"
  },
```

Outbound Connectivity

If a Node.js application needs to call external applications or services, it must do so by performing HTTP requests to the required services. The service can either be located in the same system (for example, an SAP HANA XS advanced system) or on an external system on the Internet. For connections to remote systems, it is recommended for Node.js applications to make use of the "`request`" module. For outbound connections, you need to consider the following:

- Proxy connections
  If your Node.js applications runs behind an HTTP proxy, the details of the proxy connection must be provided to the `request` module. Request can discover this information either from environment variables

set in the application's manifest file (`manifest.yml`) or directly from the options object of the HTTP request.

- Authentication
  If the application needs to call a service that performs authentication by means of Security Assertion Markup Language (SAML) and JWT (authentication), and the calling service does so as well, then the SAML token can be taken from the incoming request and attached to the outgoing request.

Sample Code

```
var options = {
    url: remoteURL,
    headers: {
        'authorization': req.headers.authorization
    }
};
```

Note

For this so called “principal propagation” to work, both the calling and the called application must be bound to the same User Account and Authentication (UAA) service. Otherwise the SAML token attached to the request will not be recognized and the request cannot be authenticated.

Related Information

Create the XS Advanced Application Package Descriptor [page 818]
XS Advanced Application Package Descriptor Configuration Syntax [page 822]
Maintaining Application Security in XS Advanced [page 957]

7.5.2 XS Advanced Application Package Descriptor Configuration Syntax

The package descriptor defines the prerequisites and dependencies that apply to packages in a JavaScript application deployed in the SAP HANA XS advanced run time.

The contents of the XS advanced application package descriptor, `package.json`, must be formatted according to the JSON rules. The required syntax for the properties and their corresponding keys is described below:

Sample Code

```
{
  "name": "<application name>",
  "description": "<application description>",
  "private": true,
  "version": "1.3.1",
  "repository": {
    "url": "git@github.acme.com:xs-samples/nodejs-hello-world.git"
  },
```

```
  "dependencies": {
    "@sap/xsjs": "^1.0.5",
    "@sap/approuter": "^2.3.1"
  },
  "engines": {
    "node": ">=0.10.x <0.12"
  },
  "scripts": {
    "start": "node --max-old-space-size=400 --expose-gc main.js"
  }
}
```

name

The name of the JavaScript application whose package prerequisites and dependencies are defined in this package description.

Sample Code

```
"name": "<application name>",
```

description

A short description the JavaScript application whose package prerequisites and dependencies are defined in this package description.

Sample Code

```
"description": "<application description>",
```

private

Use the `private` property to indicate if access to the package specified in `name` is restricted. Private packages are not published by `npm`.

Sample Code

```
"private": true,
```

version

The version of the JavaScript application whose package prerequisites and dependencies are defined in this package description.

The version value must follow the standard format for semantic versioning: `<major>.<minor>.<patch>`. The restriction ensures a reliable contract between a development environment and deployed system components.

Sample Code

```
"version": "<Major_Version>.<Minor_Version>.<Patch_Version>"
```

→ Tip

When defining `dependencies` and run-time `engines` in the package description, you can specify a range of versions, for example: `>1.0.3`, `<=1.2.5`, `^1.0.5` (compatible with version 1.0.5), or `1.2.x` (any 1.2 version), or `1.1.0 - 1.3.12` (any version including or between the specified versions).

repository

The absolute path to the repository used by the JavaScript application whose package prerequisites and dependencies are defined in this package description.

Sample Code

```
"repository": {
    "url": "<repository>@<host.name>:</path/to/your/repository>"
},
```

Sample Code

```
 "repository": {
    "url": "git@github.acme.com:samples/nodejs-hello-world.git"
  },
```

dependencies

A list of dependencies that apply to the JavaScript application whose package prerequisites and dependencies are defined in this package description.

Sample Code

```
"dependencies": {
    "@sap/xsjs": "^1.2.5",
    "@sap/approuter": "^2.3.1",
```

```
    "@sap/cds": "^1.1.7",
    "@sap/xssec": "^1.3.6",
    "@sap/xsenv": "^1.2.2"
},
```

> **i** Note
>
> In the package description, you can define a range of versions, for example: `>1.0.3`, `<=1.2.5`, or `1.2.x` (any 1.2 version), and `1.1.0 - 1.3.12` (any version including or between the specified versions).

engines

The run-time engines used by the application specified in the `name` property.

> **⁖** Sample Code

```
"engines": {
    "node": >=0.10.x <0.12"
},
```

> **i** Note
>
> In the package description, you can define one or a range of versions, for example: `>1.0.3`, `<=1.2.5`, or `1.2.x` (any 1.2 version), and `1.1.0 - 1.3.12` (any version including or between the specified versions).

scripts

The script containing the command used to start the JavaScript application along with any additional (required) options and parameters.

> **⁖** Sample Code

```
"scripts": {
    "start": "node --max-old-space-size=400 --expose-gc main.js"
  },
```

Related Information

Create the XS Advanced Application Package Descriptor [page 818]

8 Maintaining Application Services in XS Advanced

In XS advanced, applications can make use of services managed by a service broker.

In SAP HANA XS Advanced, application developers can make use of a catalog of services managed by a service broker, for example, for job schedules or user accounts and OAuth clients. To make use of a service, an instance of the service must be created and an the XS advanced application must be bound to the specified service instance. Services are either installed with the XS Advanced run-time platform, or, if required, deployed manually on the platform at a later point in time. The following XS advanced services are available by default:

- File-system storage service *
- SAP HANA Deployment Infrastructure (HDI) services *
- User Account and Authentication (UAA) service *
- Job Scheduler service
- Audit-log service *
- Instance manager *
- Portal services
- SAP UI5 service

> **i** Note
>
> Services are installed from different packages on the SAP HANA media or a defined component location. Some default services (*) are included in the core XS advanced component package `SAP EXTENDED APP SERVICES 1`; other services such as the Job scheduler, portal services, and SAPUI5 services are available in separate, additional, packages. For more information about core and additional services, see *Related Information* below.

Services brokers are advertised in the marketplace. To check which services are available, you can use the `[xs|cf] marketplace` command, as illustrated in the following example:

> **⥱** Sample Code
>
> Services in the Service Marketplace

```
$ xs marketplace
Getting services from marketplace in org acme / space devel as admin...
OK
service          plans                        description
------------------------------------------------------------------------------
fs-storage       free                         Environment variable denotes the
                                              root of client app's file system
hana             hdi-shared, sbss, schema     SAP HANA database
                 securestore
xsuaa            default, devuser, space      XS UAA Service Broker for user
                                              authentication & authorization...
auditlog         free                         Audit log broker on XS advanced
portal-services  site-host, site-content,     Service broker for creating and
                 admin-cockpit                accessing portal DB
jobscheduler     default                      Job Scheduler on XS advanced
```

```
sapui5_sb        sapui5-1.44, sapui5-1.38    SAPUI5 1.1.0-SNAPSHOT
```

Services that are manually deployed are pushed as applications (for example, using the `xs/cf` command) and exposed as services. Services that provide their own persistence, for example, HDI services, must be installed with the platform. This is because deployed services (pushed as applications) run in containers that do not provide persistent access to the corresponding file-system. In addition, services that are required by the SAP HANA platform itself, for example, the UAA service for user account management and authentication, are installed with the platform to avoid issues at boot time. Each instance of a service is assigned a “service plan”.

→ Tip

The service plan is mapped to a corresponding application “resource type” that is typically specified in the `resources` section of the application's MTA deployment descriptor (`mtad.yaml`). For example, the resource type “`com.sap.xs.hana-sbss`” is mapped to the “`hana`” service plan “`sbss`”; the resource type “`com.sap.xs.uaa-devuser`” is mapped to the “`xsuaa`” service plan “`devuser`”.

Services that bind to other services for their persistence and are only consumed by applications (for example, the job-scheduler service) can be deployed on the platform by pushing them (and the corresponding service broker) as applications. Note that the job-scheduler service is deployed and available by default.

→ Tip

The `<VCAP_SERVICES>` variable is typically used to provide applications with configuration details of a service, for example: the name of the service, any logon credentials required, the host name and port to use for connections, and the URL for the service end-point.

Related Information

Create a Service Instance [page 861]
Core Services in XS Advanced [page 827]
Additional Services in XS Advanced [page 846]
Service Plans and Resources in XS Advanced [page 856]

8.1 Core Services in XS Advanced

In XS advanced, a selection of core services are installed by default with the run-time platform.

The core services installed by default with XS advanced are included in the package `SAP EXTENDED APP SERVICES 1`, and service brokers are used to manage instances of the installed service. The following **core** services are available by default after installation of XS advanced:

Default XS Advanced Service Brokers

Service Broker	Service	Description	Package Name
auditlog	auditlog	Audit log service broker on the XS advanced platform	`SAP EXTENDED APP SERVICES 1`
fs-storage	fs-storage	Provides storage Volume Services, which define a special type of service broker that can be used to make persistent file systems available to any bound applications	
hdi-broker	hana	The service broker for the SAP HANA database and deployment infrastructure (HDI) service component	
instance-manager	managed-hana	The service broker for the dynamically managed **tenant-based** HDI service component	
uaa-security	xsuaa	The XS UAA Service Broker for user account and authentication services in XS advanced	

> → Tip
>
> Services that are not part of the core XS advanced package, `SAP EXTENDED APP SERVICES 1`, can be deployed during installation of XS advanced, if the service packages are in a location known to the SAP HANA installation tool, `hdblcm`.

For example, if the installation package of an "additional" service such as the job scheduler is located in the component directory of the SAP HANA installation tool `hdblcm` during installation of XS advanced, then the additional service will be displayed in the list of services available for installation and, if selected, will be installed along with the default services contained in the core XS advanced services package.

Related Information

Additional Services in XS Advanced [page 846]
Service Plans and Resources in XS Advanced [page 856]
Service Types in XS Advanced [page 859]
SAP Software Download Center (Logon required)

8.1.1 Audit Log Services in XS Advanced

The `auditlog` service allows you to perform audit-log tasks within the SAP HANA XS advanced run-time environment.

The SAP HANA XS advanced audit-log service is intended for use by Web applications that want to send audit log messages to a remote persistent storage. SAP HANA includes a Java client that can be used by these Web applications.

The SAP HANA XS advanced Java runtime provides an audit-log application programming interface (API) that enables applications and run-time environments to comply with the regulations on audit logging of the PIL Security standard. There are three main categories of audit logs:

- Security events
- Data access events
- Configuration change events

The audit-log API provides different interfaces that can be used for these different log types.

Service Configuration

You can use the XS command-line interface (CLI) to check that the audit-log service is displayed in the list of services registered in the Service Marketplace:

```
$ xs marketplace
Getting services from marketplace in org acme / space devel as admin...
OK
service          plans                       description
-------------------------------------------------------------------------
fs-storage       free                        Environment variable denotes the
                                             root of client app's file system
hana             hdi-shared, sbss, schema    SAP HANA database
                 securestore
xsuaa            default, devuser, space     XS UAA Service Broker for user
                                             authentication & authorization...
auditlog         free                        Audit log broker on the XSA Platform
portal-services  site-host, site-content,    Service broker for creating and
                 admin-cockpit               accessing portal DB
jobscheduler     default                     Job Scheduler on the XSA platform
```

To deploy a client application, you can use the XS CLI `xs push` command, as illustrated in the following example, which assumes the client is called `auditlog-client`

```
xs push auditlog-client
```

To create an instance of the audit-log service, use the XS CLI `xs create-service` command, as illustrated in the following example:

```
xs create-service auditlog free auditlog-instance
```

> → Tip
>
> The service plan `free` enables unlimited access to the functionality provided by the audit-log service.

To bind the audit-log service instance to the client application that you deployed, use the `xs bind-service` command:

```
xs bind-service auditlog-client auditlog-instance
```

Check that the deployed application (`auditlog-client`) has been successfully bound to the instance by checking its `<VCAP_SERVICES>` environment variable, for example, with the `xs env` command:

```
xs env auditlog-client
```

The audit-log service broker provisions these credentials for the client application. The environment variable configuration should looks like the following example:

Sample Code

VCAP_SERVICES Environment Variable

```
"VCAP_SERVICES" : {
  "auditlog" : [ {
    "name" : "auditlog-instance",
    "label" : "auditlog",
    "tags" : [ "auditlog", "audit log", "xsa" ],
    "plan" : "free",
    "credentials" : {
      "password" : "Aa_93080244...221922102809335986",
      "user" : "SBSS_6592728326...602494781306527846",
      "url" : "http://localhost:4243"
    }
  } ]
}
```

Service Maintenance

To unbind your application from the audit-log service instance, use the `xs unbind-service` command in an XS CLI command shell, as illustrated in the following example, which assumes that the application name is `auditlog-client` and the name of the service instance is `auditlog-instance`:

```
xs unbind-service auditlog-client auditlog-instance
```

To delete the service instance, use the `xs delete service` command, as follows:

```
xs delete-service auditlog-instance
```

Audit Log Storage Location

SAP HANA XS advanced stores the logs in the SAP HANA audit-logging system instead of separate database tables.

Note

SAP HANA audit logging is disabled by default and no audit logs are persisted. To ensure the persistency of audit-log events, you must enable SAP HANA audit logging.

To enable SAP HANA audit logging, create an audit-log policy for the following actions: `SECURITY EVENT`, `PERSONAL DATA ACCESS`, `PERSONAL DATA MODIFICATION`, and `CONFIGURATION CHANGE`, for example, with SAP HANA studio or by executing the following SQL query:

Sample Code

```
create audit policy xsa_audit auditing all SECURITY EVENT, PERSONAL DATA
ACCESS, PERSONAL DATA MODIFICATION, CONFIGURATION CHANGE level info;
ALTER AUDIT POLICY xsa_audit ENABLE;
```

To see the stored audit logs in the SAP HANA audit-logging system in the `xsa_audit_log` view, execute the following statement in the database with administrator credentials:

Sample Code

```
select * from xsa_audit_log;
```

This opens a view with columns as described in *Audit Trail Layout for Trail Target Database Table* in *Related Information* below.

Related Information

Configure an Application to Use the Audit Log Service [page 747]
Service Plans and Resources in XS Advanced [page 856]
Auditing Activity in SAP HANA
Audit Trail Layout for Trail Target Database Table
ALTER AUDIT POLICY Statement (Access Control)
CREATE AUDIT POLICY Statement (Access Control)

8.1.2 File-System Storage Services in XS Advanced

Configure file-system services for XS advanced applications.

Since an XS advanced application runs inside a container that might be stopped and discarded at any time, the file system of a running XS advanced application is ephemeral. To provide support for applications that require a persistent POSIX file system to function, XS advanced provides Volume Services, which enable you to define a special type of service broker that makes persistent file systems available to any bound applications.

Service Configuration

The file system services available with SAP HANA XS, advanced model, are implementations of a volume service. With XS advanced on-premise you can create a service instance by using the service `fs-storage` with the service plan "`free`", as illustrated in the following examples:

Sample Code

```
xs create-service fs-storage free <service-instance-name>
```

Note

The file system service is not available on SAP Cloud Platform.

You can display a list of the available service plans using the `xs marketplace` command, as illustrated in the following example:

Sample Code

```
xs marketplace -s fs-storage
Getting services from marketplace...
Getting plans for service "fs-storage"...
plan   description                    free/paid
-----------------------------------------------
free   A plan not incurring any costs  free
```

Service Maintenance

You can the bind the service instance to your XS advanced application and use it by inspecting the bound service instance entry in the `<VCAP_SERVICES>` environment variable of your application. This entry will contain a `volume_mounts` field that contains an array of length 1 of `volume_mounts` that have the following properties:

- `mode`
  Defines the access mode to the named file system; currently only `"rw"` (read-write) is allowed
- `device_type`
  Defines the type of file-system access; currently only `"shared"` is allowed
- `container_dir`
  Defines the path to the location where the persistent file system is mounted and made available, and where it can be consumed using regular file-IO operations.

Note

The properties `"storage-path"` and `"container_dir"` are set by the file-system service; it is not possible to set either of these properties manually, for example, to a specific path or directory.

Sample Code

VCAP_SERVICES for `fs-storage`

```
"VCAP_SERVICES" : {
  "fs-storage" : [ {
    "name" : "volume1",
    "label" : "fs-storage",
    "tags" : [ ],
    "plan" : "free",
    "credentials" : {
        "storage-path" : "/path/to/resource/18b48853-888f-4109-9730-
dbf5041c6a24"
    },

    "volume_mounts" : [ {
      "mode" : "rw",
      "device_type" : "shared",
      "container_dir" : "/path/to/directory/"
    } ]
  } ]
}
```

File-System-Service Backup

When creating and maintaining instances of the file-system service, it is important to bear in mind that the instances of the file-system service that you create are not included in any automatic, platform-wide, backup operation. If you want to ensure that instances of the file-system service are saved to a backup copy that can be restored at some point if necessary, you must perform a manual backup operation, for example, by using the XSA command-line interface, which includes the following administrator tools:

Note

To use the XSA command, you must log on to the XS advanced platform in a command shell as the operating-system user `<SID>adm`.

- `XSA backup-fss`
  Create a backup copy of the file-system service instances in the database
- `XSA restore-fss`
  Restore a backup copy of the file-system service instance from the database

The standard platform-wide operation to replicate the system does not automatically include the data maintained in an instance of the file-system service; you must perform a manual replication of the file-system data. For more information, see the section describing *Table Replication* in the *System Administration* section of the *SAP HANA Administration Guide*.

Note

If an application uses several different types of backing-services at the same time (for example, a file-system service instance and an SAP HANA service instance), the system cannot ensure a consistent backup of the respective service instances. For this reason, the applications themselves are responsible for dealing with service instances that are recovered from backups taken at different points in time.

Related Information

Core Services in XS Advanced [page 827]
Maintaining Application Services in XS Advanced [page 826]
The XSA Command Reference (SAP HANA Administration Guide)
Table Replication (SAP HANA Administration Guide)

8.1.3 SAP HANA and HDI Services in XS Advanced

The SAP HANA XS Deployment Infrastructure service (HDI) is the central infrastructure component of XS advanced model for application-container management.

The SAP HANA Deployment Infrastructure (HDI) is a service layer of the SAP HANA database that simplifies the deployment of SAP HANA database objects by providing a declarative approach for defining database objects (as design-time artifacts) and ensuring a consistent deployment into the database. The SAP HANA DI service enables you to create and manage not only HDI containers but also plain schemata, and the Secure Store on shared SAP HANA XS advanced systems.

The deployment of database objects with HDI is based on a container model where each container corresponds to a database schema. Containers can be used for multiple deployments of the same database artifacts, and for development sandboxes, too. Containers are isolated from each other; each database schema with its deployed database objects is owned by a per-schema technical database user. By default, a cross-container access at the database level is not possible. However, such cross-container access could be granted by means of database privileges.

In XS advanced, database service instances are created by a database-specific service broker, for example, the SAP HANA database service broker. In the context of the XS advanced run time, a database service instance corresponds to a single database schema and an assigned technical database user. The HANA database service broker delegates the creation of schemas to HDI in such a way that HDI can create its required metadata, database users, and database roles inside the database.

In XS advanced, an instance of the "`hana`" service corresponds to an HDI container or a plain schema in the database. When such a service is bound to an application, the bind operation “injects” into the application an application user (for use by the application code) and a deployment user (for user by the database artifact deployer) to enable access to the container or schema.

SAP HANA Service Plans

In XS advanced, SAP HANA is exposed as a managed service with the service name “`hana`”. If an application is bound to an instance of the HANA service, the credentials required to access SAP HANA are defined in the application's `<VCAP_SERVICES>` environment variable. A tenant database of an SAP HANA instance can be mapped to an Organization or Space in XS advanced, and all SAP HANA service instances created within such an Organization or Space relate to a schema in the mapped tenant database.

Output Code

```
$ [xs | cf] marketplace -s hana
```

```
Getting services from marketplace...
Getting plans for service "hana"...

service plan   description                                        free/paid
-------------------------------------------------------------------------
hdi-shared     An HDI container on an SAP HANA database           free
sbss           User with permissions to use the SBSS              free
schema         A schema on an SAP HANA database                   free
securestore    User with permissions to use the secure store      free
```

HANA Service Plans and Availability

HANA Service Plan	Description	On-Premise	Cloud Foundry
hdi-shared	An SAP HANA Deployment Infrastructure (HDI) container in an SAP HANA database	□	□
sbss	User with permissions to use the Service Broker Security Support (SBSS) component	□	□
schema	A schema in an SAP HANA database	□	□
securestore	User with permissions to maintain encrypted data in the SAP HANA Secure Store	□	□

The “hdi-shared” Service Plan

When you create and bind a service instance with the service plan `hdi-shared`, an application receives the credentials required for access to an HDI container, which is basically a database schema that is equipped with additional metadata.

HDI containers ensure isolation, and within an SAP HANA database you can define an arbitrary number of HDI containers. The same objects can be deployed multiple times into different HDI containers in the same SAP HANA database, for example, to install several instances of the same software product in the same SAP HANA database. HDI containers are isolated from each other by means of schema-level access privileges. Cross-container access at the database level is prevented by default, but can be enabled by explicitly granting the necessary privileges, for example, using synonyms.

Database objects (tables, views, procedures, and so on) have an owner: the user who created the object. When the owner of a database object is deleted, all objects owned by the deleted user are removed from the database, too. In addition, if application objects are created by end-users, the objects are deleted when the end-user is deleted, for example when the employee leaves the organization. HDI ensures that during deployment all database objects are created by a container-specific technical user, which is never deleted as long as the container exists.

In HDI, database schema content (for example, tables, views, procedures, etc.) is defined in corresponding design-time files as part of a development project. These definition artifacts are pushed to the platform as part of a special Node.js application, the HDI Deployer application `@sap/hdi-deploy`, which is part of the XS advanced platform. This deployer application binds to an SAP HANA service instance and, on start up, creates the set of database objects that correspond to the pushed definition files, for example: `myTable.hdbtable`, `myView.hdbview`, or `myProcedure.hdbprocedure`.

Sample Code

Service-Creation Parameters:

```
{"schema": "<schema-name>", "database_id": "<database_id>"}
```

Note

If no `"schema"` parameter is used, a random schema name is created and made accessible through `<VCAP_SERVICES>` after the service instance is bound to an application. The parameter `"database_id"` must be used if multiple tenant databases are mapped to the same XS advanced organization or space.

Random schema names is the recommended way to create a service instance; setting a **fixed** schema name creates a global singleton in the database and means that it is not possible to create a service instance with the same schema name.

Sample Code

Service-Binding Parameters:

```
{"roles": "<list of user roles>"}
```

Every time the service instance is bound to an application, the service broker creates a technical application user who is named `"user"` in the service instance's credentials. The bound application must provide the technical user's credentials when accessing the HDI container's run-time schema. This user is equipped with the service instance's global access role (`<schema-name>::access_role`), where `<schema-name>` is the name of the HDI container schema.

In order to assign roles from the HDI content to the technical application binding user, the HDI deployer implements an automatic assignment of the `"default_access_role"`, if the role is included in the deployed content. By using the service-binding parameter roles when binding an HDI container to an application, it is possible to declare any role which will be assigned to the technical binding `"user"`, as illustrated in the following example:

```
$> [xs | cf] bind-service my-app my-hdi-container -c {"roles":
"sap.myapp.roles::read_access_role" }
```

In the application deployment descriptor (`mtad.yaml`), this would look like the following code:

Sample Code

Application Deployment Descriptor

```
ID: com.sap.xs2.samples.my-app
version: 0.1.0
modules:
  - name: my-app1
    type: javascript.nodejs
    requires:
      - name: my-hdi-container
        parameters:
          config:
            roles: sap.myapp.roles::read_access_role
...
```

The “sbss” Service Plan

The `sbss` service plan (for "Service Broker Security Support") enables you to ensure that access to a service instance is protected. For example, when an application uses a service instance based on the `sbss` service plan, the application needs at least the credentials (for example, user name and password) of a technical user that the service must authenticate before allowing the application to access the service's functionality.

> ### Note
>
> This does not apply to the "business (end) user" who accesses the application.

Since each service broker needs the ability to create, validate and delete credentials, we offer these three distinctive functions within this Service Broker Security Support (SBSS) reuse component:

- Credential generation
  Credentials can be generated by the service broker either when a service instance is created or when the created service instance is bound to an application. It is recommended to use application-specific credentials in order to allow for the revoking of credentials of one application without influencing any other bound applications. Credentials are generated and made available to the bound application during the application-bind process.
- Credential validation
  A service instance needs to validate the credentials supplied by the application requesting access. Access to the service is only be granted if the provided credentials are valid and match the credentials required by the particular service instance.
- Credential de-provisioning
  When a service instance is unbound from an application, the credentials need to be revoked in order to prevent unauthorized access attempts (for example, if the application stored the service access credentials).

The “schema” Service Plan

The `schema` service plan creates a plain schema, which you need to manage manually or with application code. No automated deployment or schema-management services are provided.

You should consider using the `schema` service plan if your application uses Object-Relational Mapping (ORM) tools, for example, those offered by Hibernate or SQLAlchemy, and a framework is available that creates the necessary database resources on demand. The `schema` service plan could also make sense where applications create and manage their database objects using conventional DDL.

> ### Sample Code
>
> Service Creation Parameters:
>
> ```
> {"schema": "<schema-name>", "database_id": "<database_id>"}
> ```

> ### Note
>
> If no schema parameter is used, a random schema name is created and accessible via `<VCAP_SERVICES>` after the service instance is bound to an application. Random schema names is the recommended way

to create a service instance; setting a **fixed** schema name creates a global singleton on the database and means that it is not possible to create a service instance with the same schema name.

The “securestore” Service Plan

The `securestore` service plan enables a bound application to make use of stored procedures that provide access to the SAP HANA Secure Store. This functionality is available when using the XS JavaScript `$.security.store` API and allows the bound application not only to insert encrypted values **into** the Secure Store but also to retrieve (or delete) encrypted data **from** the Secure Store.

An application that is bound to a `hana` service with the service plan `securestore` can make use of the following stored procedures, which enable access to the SAP HANA Secure Store:

- `SYS.USER_SECURESTORE_INSERT`
  Insert an encrypted entry into the SAP HANA Secure Store.
- `SYS.USER_SECURESTORE_RETRIEVE`
  Retrieve an encrypted entry from the SAP HANA Secure Store.
- `SYS.USER_SECURESTORE_DELETE`
  Delete an encrypted entry from the SAP HANA Secure Store.

→ Tip

For more information about the parameters required when calling the stored procedures that provide access to the Secure Store, for example, `STORE_NAME`, `KEY`, or `VALUE`, see *Related Information* below.

Retrieving Values from the SAP HANA Secure Store

When calling the stored procedure `SYS.USER_SECURESTORE_RETRIEVE`, you need to provide the name of the target secure store (`storename`) containing the entry to be retrieved, a `key` (name) for the retrieved entry, and a binary `value` for the retrieved key. You also need to specify if the retrieved value is stored for the XS advanced application user (“`true`”) or the technical user (“`false`”). For more information about the parameters required when calling the procedures that provide access to the Secure Store, see *Related Information* below.

The following code example shows how a Node.js application could make use of the stored procedure `SYS.USER_SECURESTORE_RETRIEVE` to request and obtain encrypted entries from the SAP HANA Secure Store:

i Note

To avoid authorization problems when a Node.js application connects to the database, make sure that the resource type for the database connection specified for the MTA's Node.js module (in the `mta.yaml`) is `com.sap.xs.hana-securestore`.

‘≒, Sample Code

Call SYS.USER_SECURESTORE_RETRIEVE (Node.js Application)

```
//Secure Store Retrieve
app.get("/:key?", async(req, res) => {
```

```
     const hdbext = require("@sap/hdbext");
     const key = req.params.key;
     let inputParams = "";
     if (typeof key === "undefined" || key === null) {
            res.type("text/plain").status(500).send("ERROR: No Secure Store
Key Input Parameter Specified");
            return;
     } else {
             inputParams = {
                     KEY: key };
     }
     inputParams.STORE_NAME = "MY_SECURE_STORE";
     inputParams.FOR_XS_APPLICATIONUSER = true;
     const client = await getSecureStore();
     //(client, Schema, Procedure, callback)
     hdbext.loadProcedure(client, "SYS", "USER_SECURESTORE_RETRIEVE", (err,
sp) => {
          if (err) {
               res.type("text/plain").status(500).send(`ERROR: $
{err.toString()}`);
               return;
          }
          //(Input Parameters, callback(errors, Output Scalar Parameters,
[Output Table Parameters])
          sp(inputParams, (err, parameters, results) => {
               if (err) {
                         res.type("text/plain").status(500).send(`ERROR: $
{err.message.toString()}`);
                         return;
               }
               console.log(`Results: ${parameters.VALUE.toString("utf8")} `);
               res.type("application/json").status(200).send(`Entry in secure
store successsfully retrieved for key: ${key} and value: $
{parameters.VALUE.toString("utf8")} `);
          });
     });
});
```

Bear in mind that, in the Node.js example above, the `KEY` parameter returns a `VALUE` of type `VARBINARY`, which might need to be converted to a character or numeric data type, for example, using `TO_NVARCHAR` on the database side or by using a JavaScript Buffer. The following example code snippet shows how to use a body-parsing module to set the body content used to fill the `VALUE` parameter to `raw`, which ensures that no conversion is required either way (IN or OUT):

Sample Code

Process Request Body and Return a JavaScript Buffer

```
module.exports = () => {
    const app = express.Router();
    const bodyParser = require("body-parser");
    app.use(bodyParser.raw({
        type: "text/plain"
    }));
```

The following code example shows how a Java application could make use of the stored procedure `SYS.USER_SECURESTORE_RETRIEVE` to request and obtain encrypted entries from the SAP HANA Secure Store:

Sample Code

Call SYS.USER_SECURESTORE_RETRIEVE (Java Application)

```
@Service
public class SecureStoreServiceImpl implements SecureStoreService {
    private static final Logger LOGGER =
                        LoggerFactory.getLogger(SecureStoreService.class);
    @Autowired
    @Qualifier("secureStoreDataSource")
    private DataSource dataSource;
    @Override
    public void getValue(String storeName, String key) throws SQLException {
       LOGGER.info("secure store read value: " + storeName + " / " + key);
       try (Connection connection = dataSource.getConnection()) {

       CallableStatement callableStatement = connection
                       .prepareCall(
                           "{call SYS.USER_SECURESTORE_RETRIEVE(?,?,?,?)}");
       callableStatement.setString(1, storeName);
       callableStatement.setBoolean(2, false);
       callableStatement.setString(3, key);
       callableStatement.registerOutParameter(4, Types.VARBINARY);

       callableStatement.executeUpdate();
       byte[] bytes = callableStatement.getBytes(4);

       if (null == bytes) {
              LOGGER.info("value is null");
              return null;
       } else {
              String returnValue = new String(bytes);
              LOGGER.info("return value is :" + returnValue);
              return returnValue; 68 }
       }
    }
}
```

In the Java example above, the `KEY` parameter for the retrieved entry expects a `VALUE` (4) with the `VARBINARY` data type. In this case, we ensure the data type is correctly set with `registerOutParameter(4, Types.VARBINARY)`.

Related Information

Create a Service Instance [page 861]
Core Services in XS Advanced [page 827]
Maintaining Application Services in XS Advanced [page 826]
SAP HANA Secure-Store Interface Procedures and Parameters [page 1022]

8.1.4 Instance Manager Services in XS Advanced

Enable applications that are multi-tenant aware to manage service instances for individual tenants.

In XS advanced, service instances (for example, HDI containers) are statically bound to an application at deployment time. In contrast, applications that are multi-tenant aware and leverage service instances for tenant separation (where each tenant database stores its data in a separate HDI container) need to be able to create additional service instances dynamically at run time, for example, whenever a new tenant database is added. Applications that can recognize individual tenant databases also need to be able to connect to any one of these service instances when processing a request for a specific tenant database. To support this requirement, a set of Instance Manager services is used to make application-managed instances available.

An Instance Manager service is, itself, a service (`managed-hana`) that can be used to manage the life cycle of instances of a dedicated service (`hana`) for applications. An application must be bound to an instance of an Instance Manager service which has been created based on a supported service plan. For the `managed-hana` service the available service plans include: `hdi-shared`, `schema`, and `securestore`, which are all inherited from the `hana` service.

When the application is started, it reads the credentials for the Instance Manager instance from `<VCAP_SERVICES>`. The application uses these credentials to invoke the services provided by the Instance Manager and create new actual application-managed instances (for example, HDI containers), whenever a new tenant is added to the application. When a tenant uses the application and access to this application-managed instance is needed, the application invokes the Instance Manager. The application receives the credentials for the tenant-specific service instance based on a tenant identifier that was provided when the service instance was originally created. With these credentials it can then connect to the application-managed instance and dynamically switch between the available instances as required in order to process requests from specific tenants.

Related Information

Set up and Use an Application-Managed Instance [page 867]
Core Services in XS Advanced [page 827]
Service Types in XS Advanced [page 859]
Service Plans and Resources in XS Advanced [page 856]

8.1.5 User Account and Authentication Services in XS Advanced

The SAP HANA XS User Account and Authentication service (UAA) is the central infrastructure component of XS advanced model for user authentication.

The UAA uses OAuth 2.0 for authentication to the XS advanced application and database containers. In the context of the OAuth flow, the UAA provides scopes, role templates, and attributes to applications deployed in the XS advanced run time. If a business user has a certain scope, an access token with this scope is issued, which enables a user to run the application: the scopes are used to define the model that describes who has the authorization to start an application running in the XS advanced run time.

> **i Note**
>
> The UAA service supports either an SAML 2.0 identity provider (IDP) or an SAP HANA database user store as external user stores.

The XS advanced model's UAA service provides a programming model for developers; it enables developers to define templates that can be used to build role and authorization models for business users of applications deployed to the XS advanced run time. In the OAuth 2.0 work flow, the UAA acts as an OAuth server, which includes both token and revocation end points.

Security Services in the Application Router

In the XS advanced run time, the application router is the single point-of-entry for an application running in SAP HANA XS advanced; the application router is part of the XS advanced application instance and triggers the user-authentication process in the UAA. In the OAuth work flow, the XS application instance (including the application router and any bound containers) is the OAuth 2.0 client, which handles user authentication and authorization.

Container Security APIs

In the context of OAuth 2.0, the security APIs for the XS advanced run time container act as resource servers; the APIs provide the security context for user authentication in XS advanced. When the user authentication process is triggered, the XS advanced container security APIs receive an HTTP header with "`Authorization: Bearer`" and a JSON Web token (JWT) from the application router.

The token contains information describing the user and any authorization scopes, which must be validated by the XS advanced container security APIs using the security libraries provided by the UAA. Applications deployed in the XS advanced run time can use the security API to check if scope values have been assigned to the user or application. The XS container security API provides the security **context** for XS advanced applications (for example, scopes, attributes, token information); the JWT token initializes the XS advanced security context. A security API is available for the following application run time containers:

- Node.js API
- XS JavaScript (compatibility layer) API
- Java API

Every XS advanced application's run time container must use the XS advanced container security API to provide the security context. The XS advanced security context is initialized with the JWT token and enables you to perform the following functions:

- Get user information
- Check an authorization scope
- List authentication attributes
- Get a token

XS UAA Service Plans

The XSUAA service plan you use determines the impact on the `xsappname` that is written into the credentials section of the environment from the application bound to the service (`[xs | cf] env <appname>`. Each of the XSUAA service plans is designed for use with a particular kind of user-security scenario, and service availability depends on platform, for example, on-premise (`xs`) and SAP Cloud Plaftorm @ Cloud Foundry (`cf`), as illustrated in the following examples:

Sample Code

SAP HANA Managed Service

```
xs marketplace -s xsuaa

Getting services from marketplace...
Getting plans for service "xsuaa"...

service plan   description                                  free/paid
-------------------------------------------------------------------
default        Production                                   free
devuser        Development                                  free
space          Space-specific name adoption                 free
```

Sample Code

SAP HANA Managed Service (SAP Cloud Platform/Cloud Foundry)

```
cf marketplace -s xsuaa

Getting services from marketplace...
Getting plans for service "xsuaa"...

service plan   description                                  free/paid
-------------------------------------------------------------------
application    Plan for business applications               free
```

The XS User Account and Authentication service provides the following plans for XS advanced:

XSUAA Service Plans and Availability

XSUAA Service Plan	Description	On-Premise	Cloud Foundry
default	Production use	□	-
devuser	Development user; required if you want to run applications for different developers in the same XS advanced “space”	□	-
space	Space-specific name adoption, which is required if you want to run the same XS advanced applications in different XS advanced “spaces”	□	-
application	PaaS-enabled service plan only available in Cloud-Foundry landscapes	-	□

XSUAA Service Plan: “default”

If you use the XSUAA service plan `"default"`, you must ensure that the name of the XS advanced application specified in `xsappname` in the application's security descriptor `xs-security.json` is unique in the XS advanced landscape in which the application is deployed. This means that it is not possible to deploy an application more than once with service plan `"default"` unless you modify the ““xsappname”” property in the corresponding `xs-security.json` file. You would get an error message during the second deployment.

> → Tip
>
> The `xsappname` that is written into the credentials section of the environment is identical to the application name specified in the application's `xs-security.json` file.

The obvious disadvantage of this plan is that it is not possible to find out which application names are already in use in the XS advanced landscape (defined in `“xsappname”`). If a duplicate application name is used, the deployment of the application with the duplicate name will fail. For this reason, it is recommended to use the `default` service plan only in smaller XS advanced landscapes, for example, your private test landscape.

XSUAA Service Plan: “devuser”

The service plan `"devuser"` requires that the value of the `"xsappname"` property in the the application descriptor `xs-security.json` is unique in the XS advanced landscape for the development user that deploys the application. This means that an application can be deployed by several development users at the same time but not twice by the **same** development user (unless the developer modifies the `"xsappname"` property in the corresponding application's `xs-security.json`file). A duplicate application name would result in an error message during the second deployment.

> → Tip
>
> The `"xsappname"` that is written into the credentials section of the environment is enhanced with a suffix and has the following format: `"<xsappname>!u<user index>"`, where `<user index>` is an incrementing number assigned per user that the XS UAA maintains internally and differs from landscape to landscape, for example, `myapp!u2`.

The advantage of `“devuser”` plan is that you know which `xsappname` your development user has already deployed in the XS advanced landscape where you want to deploy your application. It is recommended to use the `"devuser"` plan for XS advanced on-premise landscapes where multiple developers are working independently on the same application.

XSUAA Service Plan: “space”

The service plan `"space"` requires that the value of the `"xsappname"` property specified in the `xs-security.json` file is unique in the XS advanced landscape within the space in which the application is deployed. This means that an application may be deployed in several spaces at the same time but not twice in the same space (unless you modify the value of `"xsappname"` in the second application's

`xs-security.json` security descriptor). Failure to adjust the XS advanced application name would result in an error message during the second deployment.

> → Tip
>
> The `"xsappname"` that is written into the credentials section of the environment is enhanced with a suffix and has the following format: `"<xsappname>!i<space index>"`, where the `<space index>` is a number that the UAA maintains internally and differs from landscape to landscape, for example, `myapp!i3`.

The advantage of the `"space"` service plan compared to the `"default"` service plan is that it is possible to see which `xsappnames` are already in use in the XS advanced space where you want to deploy your application. it is recommended to use the `"space"` plan for XS advanced on-premise landscapes.

XSUAA Service Plan: “application”

For platform applications that need to integrate with the platform User Account and Authentication (UAA) service for **user-related** operations (for example, business users of applications), the `xsuaa` service broker provides the `"application"` service plan.

The service plan `"application"` requires that the value of the `"xsappname"` property specified in the `xs-security.json` file is unique in the Cloud Foundry landscape within the tenant in which the application is deployed. This means that an application may be deployed in several tenants at the same time but not twice in the same tenant (unless you modify the value of `"xsappname"` in the second application's `xs-security.json` security descriptor). Failure to adjust the application name results in an error message during the second deployment.

> → Tip
>
> The `"xsappname"` that is written into the credentials section of the environment is enhanced with a suffix and has the following format: `"<xsappname>!t<tenant index>"`, where the `<tenant index>` is a number that the UAA maintains internally and differs from landscape to landscape, for example, `myapp!i5`.

The PaaS-enabled service plan `"application"` is only available in Cloud Foundry landscapes. It is highly recommended to use the `"application"` service plan for all standard Cloud Foundry deployments.

Related Information

Create a Service Instance [page 861]
Core Services in XS Advanced [page 827]
Maintaining Application Services in XS Advanced [page 826]
Maintaining Application Security in XS Advanced [page 957]
Application Security Descriptor Configuration Syntax [page 999]

8.2 Additional Services in XS Advanced

In XS advanced, the administrator can deployed additional services on the XS advanced run-time platform manually.

Additional services that are not part of the core XS advanced package, `SAP EXTENDED APP SERVICES 1`, are available for download from the SAP Service Marketplace and can be installed separately as individual packages. However, additional services can also be deployed during the installation of XS advanced, it the installation packages for the additional services are located in a folder known to the SAP HANA installation tool, `hdblcm`, at installation time.

For example, if the installation package for the XS advanced job scheduler is located in the component directory of the SAP HANA installation tool, `hdblcm`, during installation of XS advanced, then the job scheduler will be displayed in the list of services available for installation and, if selected, will be installed along with the default services included in the core XS advanced services package.

Service brokers are used to manage service instances. The following table lists additional XS advanced services that are available for download from SAP Service Marketplace:

Additional XS Advanced Service Brokers

Service Broker	Service	Description	Package Name
messaging-service	messaging-service	Messaging service broker on the XS advanced platform	`XS MESSAGING SERVICE`
jobscheduler	jobscheduler	Job Scheduler service broker on the XS advanced platform	`XS SERVICES`
portal-services	portal-services	Service broker for creating and accessing portal DB	`XSAC_PORTAL_SERVICES`
sapui5	sapui5_sb	Service broker providing resources for UI5 applications	`SAPUI5 SERVICE BROKER XSA`

→ Tip

The additional XS advanced services listed here are available for download on *SAP Software Downloads* at the following location for those people with the required S-User ID:

Service Marketplace > *SAPSoftware Downloads [Downloads]* > *SUPPORT PACKAGES & PATCHES* > *By Alphabetical Index (A-Z)* > *H* > *SAP HANA PLATFORM EDITION*

- *SAP HANA PLATFORM EDIT. 1.0*
- *SAP HANA PLATFORM EDITION 2.0*

Related Information

Messaging Services in XS Advanced [page 847]
Core Services in XS Advanced [page 827]
SAP Software Download Center (Logon required)

8.2.1 Messaging Services in XS Advanced

The messaging service enables communication between software components or applications.

In SAP HANA XS advanced model, the messaging service is an application service that enables consumer applications to receive messages sent by producer applications depending on the topics or queues that consumer applications are subscribed to. The messaging service represents a message broker for remote communication between systems using the Apache ActiveMQ library with JMS (Java Message Service) specification. It uses ActiveMQ as message-oriented middleware, which enables the service to provide scalability, high availability, reliability, and security for enterprise messaging.

The service fulfills the need for both notification as well as interoperation between applications and enables applications to communicate in an asynchronous, loosely coupled manner. Indeed, the application sending a message does not need to know anything about the application that receives the message, nor does the application receiving the message need to know which application sent it. To communicate with each other, the sending and receiving applications simply need to connect to the same messaging service instance and only need to know which destination to use.

Service Configuration

After installing the messaging service on an XS advanced system, you can see the messaging service displayed in the list of services registered in the Service Marketplace, as illustrated in the following example:

Sample Code

Messaging Service in the Marketplace

```
$ xs marketplace
Getting services from marketplace in org acme / space devel as admin...
OK
service            plans                       description
-------------------------------------------------------------------------
fs-storage         free                        Environment variable denotes the
                                               root of client app's file system
hana               hdi-shared, sbss, schema    SAP HANA database
                   securestore
xsuaa              default, devuser, space     XS UAA Service Broker for user
                                               authentication & authorization...
auditlog           free                        Audit log broker on XS advanced
portal-services    site-host, site-content,    Service broker for creating and
                   admin-cockpit               accessing portal DB
jobscheduler       default                     Job Scheduler on XS advanced
sapui5_sb          sapui5-1.44, sapui5-1.38    SAPUI5 1.0.1-SNAPSHOT
messaging-service  free                        Messaging Service Broker on XSA
```

The messaging service adds the following service instances for the services that it uses:

- `messaging-service-sbss`
  Instance of SAP HANA database service
- `messaging-service-storage`
  Instance of file-storage service
- `messaging-service-audit`
  Instance of audit-log service

You can use the `xs services` command to display a list of all services running available in a particular space in XS advanced, as illustrated in the following example:

Sample Code

Available Messaging Services

```
$ xs services
Getting services in org "acme" / space "devel" as admin...
Found services:
name                        service       plan      bound apps
------------------------------------------------------------------------------
messaging-service-sbss      hana          sbss      hub, node, broker
messaging-service-storage   fs-storage    free      hub, node
messaging-service-audit     auditlog      free      hub, node
```

The messaging service also provides three applications that form the network of brokers for ActiveMQ. You can display a list of the applications in a command shell with the command `xs apps`, as illustrated in the following example:

Sample Code

Available Messaging Applications

```
$ xs apps
Getting apps in org "myOrg" / space "DEV" as admin...
Found apps:
name                    requested  instances  memory  disk   urls
                        state
------------------------------------------------------------------------------
messaging-service-hub     STARTED    1/1        512 MB  <...>  tcps://host:51
messaging-service-node    STARTED    1/1        512 MB  <...>  tcps://host:51
messaging-service-broker  STARTED    1/1        512 MB  <...>  tcps://host:51
```

To deploy a client application, you can use the XS CLI command `xs push`, as illustrated in the following example, which assumes the client is called `message-producer`:

```
xs push message-producer
```

To create an instance of the messaging service, use the XS CLI command `xs create-service`, as illustrated in the following example:

```
xs create-service messaging-service free messaging-service-instance
```

→ Tip

The service plan "`free`" enables unlimited access to the functionality provided by the messaging service.

To bind the messaging-service instance to the client application, use the XS CLI command `xs create-service`, as illustrated in the following example:

```
xs bind-service message-producer messaging-service-instance
```

Check that the deployed application (`message-producer`) has been successfully bound to the message-service instance by displaying details of the application's `<VCAP_SERVICES>` environment variable, as illustrated in the following example

```
xs env message-producer
```

The message-service broker provisions the credentials in the `<VCAP_SERVICES>` for the client application. The environment variable configuration should looks like the following example:

Output Code

```
"VCAP_SERVICES" : {
  "messaging-service" : [ {
    "name" : "messaging-service-instance",
    "label" : "messaging-service",
    "tags" : [ "messagingservice", "messaging service", "messaging-
service" ],
    "plan" : "free",
    "credentials" : {
      "password" : "***********",
      "user" : "SBSS_**********",
      "url" : "amqps://host:51070"
    }
   } ]
}
```

Maintaining the Messaging Service Instance

To unbind your application from the messaging service instance, use the `xs unbind-service` command in an XS CLI command shell, as illustrated in the following example, which assumes that the application name is `message-producer` and the name of the service instance is `messaging-service-instance`:

```
xs unbind-service message-producer messaging-service-instance
```

To delete the service instance, use the `xs delete service` command, as follows:

```
xs delete-service messaging-service-instance
```

Logging the Messaging Service

By default, the logging level of the messaging service is ERROR at the root level, and the conversion pattern is:

Sample Code

```
<pattern>#%d{yyyy MM
dd HH:mm:ss.SSS}#%timeZone#%level#%replace(%lo){'([\\#])','\\
$1'}#%n#%guid#%class#%replace(%t){'([\\#])','\\$1'}#%n%m%n%n</pattern>
```

Messaging-Service Logging-Pattern Components

Pattern Component	Description
`d{yyyy MM dd HH:mm:ss.SSS}`	The date and time in format: YYYY MM DD HH:MM:SS:mmm.
`timeZone`	The current timezone of the system
`level`	The severity level of the log record, for example: `DEBUG, PATH, INFO, WARNING, ERROR, FATAL`
`lo`	The name of the logger at the origin of the logging event
`guid`	The globally unique ID of the log record
`class`	The fully qualified class name of the caller issuing the logging request
`t`	The thread name
`m`	The message text from the logging event

Related Information

Additional Services in XS Advanced [page 846]
Core Services in XS Advanced [page 827]
Additional Services in XS Advanced [page 846]
Service Plans and Resources in XS Advanced [page 856]

8.2.2 Job Scheduler Services in XS Advanced

The Job Scheduler service enables you to create and schedule long-running operations or jobs.

In SAP HANA XS advanced model, the Job Scheduler is an application service. The Job Scheduler service enables you to create and schedule long-running operations or jobs. This service is deployed during the installation of the SAP HANA XS advanced model.

Note

Some services are not part of the core XS advanced package but can be deployed during the default installation. For example, if the installation package of a non-default service such as the job scheduler is located in the component directory of the SAP HANA installation tool `hdblcm` during installation of XS

advanced, then the non-default service will also be installed along with the default services included in the core XS advanced services package.

Service Configuration

You can use the XS command-line interface (CLI) to check that the `jobscheduler` service is displayed in the list of services registered in the Service Marketplace:

```
$ xs marketplace
Getting services from marketplace in org acme / space devel as admin...
OK
service          plans                       description
------------------------------------------------------------------------------
fs-storage       free                        Environment variable denotes the
                                             root of client app's file system
hana             hdi-shared, sbss, schema    SAP HANA database
                 securestore
xsuaa            default, devuser, space     XS UAA Service Broker for user
                                             authentication & authorization...
auditlog         free                        Audit log broker on the XSA Platform
portal-services  site-host, site-content,    Service broker for creating and
                 admin-cockpit               accessing portal DB
jobscheduler     default                     Job Scheduler on the XSA platform
```

To deploy a client application, you can use the XS CLI `xs push` command, as illustrated in the following example, which assumes the client is called `job-scheduler-client`

```
xs push job-scheduler-client
```

To create an instance of the `jobscheduler` service, use the XS CLI `xs create-service` command, as illustrated in the following example:

```
xs create-service jobscheduler default scheduler-instance
```

→ Tip

The service plan `default` ensures standard access to the functionality provided by the `jobscheduler` service. For more information about access configuration, see Job Scheduler Access [page 853] below.

To bind the `jobscheduler` service instance to the client application that you deployed, use the `xs bind-service` command:

```
xs bind-service job-scheduler-client scheduler-instance
```

Check that the deployed application (`job-scheduler-client`) has been successfully bound to the instance by checking its `<VCAP_SERVICES>` environment variable, for example, with the `xs env` command:

```
xs env job-scheduler-client
```

The `jobscheduler` service broker provisions these credentials for the client application. The environment variable configuration should looks like the following example:

Sample Code

VCAP_SERVICES Environment Variable

```
"VCAP_SERVICES" : {
  "jobscheduler" : [ {
    "name" : "scheduler-instance",
    "label" : "jobscheduler",
    "tags" : [ "jobscheduler" ],
    "plan" : "default",
    "credentials" : {
      "password" : "Aa_93080244...221922102809335986",
      "user" : "SBSS_6592728326...602494781306527846",
      "url" : "https://<REST_URL>:<REST_PORT>"
    }
  } ]
}
```

Job Schedule Execution Mode

Job Scheduler supports the following **modes** for applications to execute a job:

- Synchronous Mode
  Suitable for jobs that run for a short span of time, for example, an OData service end point
- Asynchronous Mode
  Suitable for jobs that run for a long span of time, for example, end points which trigger batch processing

Job Schedule Execution Type

Job Scheduler provides the following **types** of schedules for a job:

- Recurring Schedule
  Runs periodically at a specified time, dates, or interval. Recurring schedules can be created in the following ways:
  - The `repeatInterval` parameter:
    Defines the interval in human-readable text (for example, “2 minutes”), which can be used to set up a recurring schedule. The repeat interval defines the gap between each run of the schedule.
  - The `cron` parameter:
    Defines a `cron` expression (for example, `"cron": "* * * * * *"`) used to represent a set of times, when the job is executed.
  - The `repeatAt` parameter:
    Defines the exact time, every day, when the job is executed.
- One-Time Schedule
  Runs only once at the specified time. One-time schedules can be created in the following ways:
  - Human-readable text string:
    A human-readable text string that defines the specific time for schedule execution (for example: “10 hours from now”, “3.30pm”, or “Friday at 2am”)

- Using a `Date` object, with a pre-defined format, for example,
  `"startTime": {"date": "2015-10-20 4:30 +0000", "format": "YYYY-MM-DD HH:mm Z"}`
  The string is checked against both IETF-compliant RFC 2822 timestamps and ISO-8601.

Job Scheduler Access

The Job Scheduler can be accessed and used in the following ways during application development:

- APIs:
  The Job Scheduler service offers RESTful and client-specific APIs for Java and Node.js. The administrator **scope** is required to use the Job Scheduler API to maintain run time configurations for jobs and job schedules.
- User Interface:
  The *Job Scheduler Dashboard* is the tool used to manage run time configurations for jobs and job schedules. Administrator authorization is required to maintain jobs and job schedules in the *Job Scheduler Dashboard*.

> → Tip
>
> You can program actions in any programming language or platform. The runtime also supports jobs created in the SAP HANA XS classic version.

Related Information

Core Services in XS Advanced [page 827]
Service Plans and Resources in XS Advanced [page 856]
Scheduling Jobs in XS Advanced [page 881]

8.2.3 SAP UI5 Services in XS Advanced

The UI5 service provides resources to UI5 applications needed to run their graphical user interfaces

In a typical XS advanced installation, one service is installed at any point in time and the service corresponds to a UI5 release. UI5 applications no longer need to refer to the particular UI5 service that they depend on; they can refer to the UI5 Service Broker, which serves the bootstrap URL of the service they require.

Unlike standard service brokers, the UI5 Service Broker does not need to manage bindings to UI5 applications. If an application refers to a version of SAP UI5 during deployment, the SAP UI5 service broker is called and serves the UI5 bootstrap URL to the application, to complete its installation.

If an application needs UI5 updates or fixes, which are included with a new version of the UI5 release, the UI5 service is updated silently for the consuming application. It is not necessary to redeploy the application; the changes and fixes included in the new UI5 version are loaded and made available automatically.

→ Tip

The SAPUI5 service on XS advanced returns the CORS header `Access-Control-Allow-Origin=*` for all served resources. If you want to customize the CORS headers for the served resources of the SAPUI5 service so that you can restrict the `Access-Control-Allow-Origin` to the host requesting the resources or only allow a list of predefined hosts to access the SAPUI5 resources, see SAP Note 3259984 (*Apply CORS headers to SAPUI5 XS Advanced applications*).

Although there is typically only one UI5 release installed on your system, the UI5 Service Broker is designed to be able to discover multiple UI5 releases, dynamically. Whenever a new SAPUI5 release is installed, it appears as a service plan in the service Market Place. The following example shows two versions of SAPUI5 available as service plans:

Sample Code

```
$ xs marketplace
Getting services from marketplace in org acme / space devel as admin...
OK
service          plans                       description
-------------------------------------------------------------------------
fs-storage       free                        Environment variable denotes the
                                             root of client app's file system
hana             hdi-shared, sbss, schema    SAP HANA database
                 securestore
xsuaa            default, devuser, space     XS UAA Service Broker for user
                                             authentication & authorization...
auditlog         free                        Audit log broker on XS advanced
portal-services  site-host, site-content,    Service broker for creating and
                 admin-cockpit               accessing portal DB
jobscheduler     default                     Job Scheduler on XS advanced
sapui5_sb        sapui5-1.44, sapui5-1.38    SAPUI5 1.0.1
```

Service Configuration

An UI5 app, consuming the UI5 broker, should have a reference to it in its configuration. In the application deployment descriptor (`mtad.yaml`), configuration entries are required in both the `modules` and `resources` sections, as illustrated below:

- `modules`
  Declare the name of the required UI5 resource:

  ```
  modules:
    - name: myapp-ui5-consumer
      requires:
        - name: myapp-ui5-provider
  ```

  ! Restriction

  The service name must be unique in the XS advanced environment.

- `resources`
  Declare the name of the UI5 service along with the resource type:

  ```
  resources:
  ```

```
  - name: myapp-ui5-provider
    type: com.sap.xs.ui5
```

> **Note**
>
> The default version of the UI5 service is currently 1.52, and only one UI5 version of each major release can be installed per run-time space. This means that in one space you could have versions 1.44.10 and 1.52.3, but not 1.52.3 and 1.52.4.

To specify a specific version of the UI5 service (for example, 1.44), use the `parameters` property along with the resource type as shown in the following example:

```
to the hostresources:
 - name: myapp-ui5-provider
   type: com.sap.xs.ui5
   parameters:
     service-plan: sapui5-1.44
```

You also need to configure the application router to use the UI5 broker `sapui5_sb`, as illustrated in the following example:

> **Sample Code**
>
> Application Router Configuration for the UI5 Service Broker

```
{
  "source": "/(.*)",
  "localDir": "resources",
  "cacheControl": "public, max-age=1000,must-revalidate",
  "authenticationType": "xsuaa",
  "scope": {
          "GET": "$XSAPPNAME.Display",
          "default": "$XSAPPNAME.Display"
          },
  "replace": {
          "pathSuffixes": [
               "index.html"
           ],
          "services": {
               "sapui5_sb": {
                   "tag": "ui5"
                }
          }
   }
}]
```

The `index.html` file provides the start-up page of the UI5 application, and must include a script tag with UI5 bootstrap information as illustrated in the following example:

> **Sample Code**
>
> Application Start-up Page with UI5 Service Broker Configuration

```
<!DOCTYPE HTML>
<html>
<head>
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<meta http-equiv='Content-Type' content='text/html;charset=UTF-8' />
<title>web</title>
    <script id="sap-ui-bootstrap" src="{{{sapui5_sb.url}}}/resources/sap-ui-
core.js"
```

```
            data-sap-ui-libs="sap.m"
            data-sap-ui-theme="sap_belize_plus">
    </script>
[...]
</head>
<body>
[...]
</body>
</html>
```

At application-deployment time, UI5 service broker determines the UI5 bootstrap URL and substitutes `sapui5_sb.url` with the URL of the appropriate UI5 release.

Related Information

SAP Note 3259984
Core Services in XS Advanced [page 827]
Service Plans and Resources in XS Advanced [page 856]
XS Advanced Security Tools and Components [page 984]

8.3 Service Plans and Resources in XS Advanced

A service plan is a particular type of service (for example, a database configuration) that is available for use.

A service broker advertises **service plans** using the catalog. You can use the advertised plans to configure the service you want to create. For example, you can use the "`hdi-shared`" service plan to create a plain HDI container on a shared HANA database system.

→ Tip

The service plan is mapped to a corresponding application "resource type" that is typically specified in the `resources` section of the application's MTA deployment descriptor (`mtad.yaml`).

For example, the resource type "`com.sap.xs.hana-sbss`" is mapped to the "`hana`" service plan "`sbss`"; the resource type "`com.sap.xs.uaa-devuser`" is mapped to the "`xsuaa`" service plan "`devuser`".

The following table shows how **resources** defined in the MTA deployment descriptor are mapped to a corresponding service and service plan. Bear in mind that the availability of features and functionality in the Cloud environment can change at short notice. For information about the latest changes and additions to the support for Cloud Foundry applications in the SAP Cloud Platform, see *MTA Resource Types in the SAP CP Cloud Foundry Environment* in *Related Information* below.

! Restriction

In the following table, "platform" is either XS (XS advanced) or CF (Cloud Foundry).

Services, Service Plans, and MTA Resources

Service	Service Plan	Platform	Resource	Created Service
auditlog	free	XS	com.sap.xs.auditlog	Audit-log service
	standard	CF	auditlog	
fs-storage	free	XS	com.sap.xs.fs	File-system storage service
hana	hdi-shared	XS/CF	com.sap.xs.hdi-container	HDI container service
	sbss	XS/CF	com.sap.xs.hana-sbss	Service-broker security service
	schema	XS/CF	com.sap.xs.hana-schema	Plain schema service
	securestore	XS/CF	com.sap.xs.hana-securestore	SAP HANA secure-store service
managed-hana	hdi-shared	XS	com.sap.xs.managed-hdi-container	Managed HDI container service
	schema	XS	com.sap.xs.managed-hana-schema	Managed SAP HANA plain-schema service
	securestore	XS	com.sap.xs.managed-hana-securestore	Managed SAP HANA secure-store service
jobscheduler	default	XS	com.sap.xs.job-scheduler	Global job-scheduler service
	lite	CF		
portal-services *	site-host	XS/CF	com.sap.portal.site-host	Portal DB site-host service
	site-content	XS/CF	com.sap.portal.site-content	Portal DB content service
	admin-cockpit	XS/CF	-	Portal DB admin cockpit service
sds	default	XS/CF	com.sap.xs.sds	Global streaming analytics service
sapui5_sb	sapui5-<Version>	XS	org.cloudfoundry.managed-service	SAPUI5 service
xsuaa	default	XS	com.sap.xs.uaa	Global UAA service

Service	Service Plan	Platform	Resource	Created Service
	devuser	XS	com.sap.xs.uaa-devuser	Development-user UAA service
	space	XS	com.sap.xs.uaa-space	UAA service for a space
	application	CF	com.sap.xs.uaa-application	PaaS-enabled UAA application service plan for CF scenarios
application-logs	lite	CF	application-logs	Streams logs of bound applications to a central application logging stack
autoscaler	lite	CF	autoscaler	Automatically increase or decrease the number of application instances based on a policy you define.
connectivity	lite	CF	connectivity	Establishes secure, reliable connectivity between Cloud applications and on-premise systems
feature-flags	lite	CF	feature-flags	Feature Flags service for the control of feature rollout
ml-foundation-services	lite	CF	ml-foundation-services	Provides access to foundation services by provisioning and deprovisioning“ ml foundation” services
mongodb	v3.0-container	CF	mongodb	MongoDB document-oriented database system
objectstore	s3-standard	CF	objectstore	Provides a highly available, distributed, consistent object store
postgresql	v9.4-container	CF	postgresql	Provides a postgreSQL object-relational database system

Service	Service Plan	Platform	Resource	Created Service
rabbitmq	v3.6-container	CF	rabbitmq	Provides RabbitMQ messaging
redis	v3.0-container	CF	redis	Redis in-memory data structure store

> ! Restriction
>
> * The MTA resource type `com.sap.portal.site-host` is only for use with the SAP Node.js module `@sap/site-entry`. The MTA resource type `com.sap.portal.site-content` is only for use with the SAP Node.js module `@sap/site-content-deployer`.

Related Information

MTA Deployment Descriptor Syntax [page 173]
Core Services in XS Advanced [page 827]
Service Types in XS Advanced [page 859]
MTA Resource Types in the SAP CP Cloud Foundry Environment

8.4 Service Types in XS Advanced

Services provided in XS advanced are available in different types.

XS advanced applications can make use of services that are either created by a user or available for general use in the service market place. Services available in the Service Marketplace are managed by a service broker. The following table lists and explains the types of service to which you can bind your XS advanced applications:

XS Advanced Service Types

Service Type	Description	Example

Managed	Services that are available in pre-configured scopes in the Service Marketplace	`fs-storage`
		`xsuaa`
		`hana`
		`auditlog`
		`portal-services`
		`jobscheduler`
		`sapui5_sb`

Service Type	Description	Example
User-provided	Services that are provided on demand and by hand for use by applications in a specific, dedicated space. User-provided services are **not** available in (or managed by) the Service Marketplace and are not associated with a service plan. Unlike managed services, user-provided services do not include any automatic provisioning of resources or credentials; these must by provided manually.	`lcm-service-credentials`

> **Note**
>
> The following examples are for illustration purposes only.

To display a list the services running in the current organizational space, log in to an XS advanced space and run the `services` command, as illustrated in the following example:

```
$ [xs | cf] services
Getting services in org "myorg" / space "SAP" as XSA_ADMIN...
Found services:
name                      service        plan         bound apps
-----------------------------------------------------------------------
myHdiService              hana           hdi-shared   xmla
myUaaService              xsuaa          space        xmla
xsa-cockpit-db            hana           schema       xsa-cockpit
xsa-cockpit-uaa           xsuaa          space        xsa-cockpit
lm-service-credentials    user-provided               product-installer
```

For details of the services available in the service marketplace, and the corresponding service plans, use the `marketplace` command, as illustrated in the following example:

```
$ [xs | cf] marketplace
Getting services from marketplace...
service           plans                          description
-------------------------------------------------------------------------------
fs-storage        free                           xs file service...
xsuaa             default, devuser, space        XS UAA Service Broker...
hana              hdi-shared, sbss, schema,      SAP HANA database
                  securestore
managed-hana      hdi-shared, schema,            Run-time service instances
                  securestore                    for SAP HANA DB
auditlog          free                           Audit logs on XS advanced
portal-services   site-host, site-content,       Creating and accessing portal
DB
                  admin-cockpit
jobscheduler      default                        Job Scheduler for repeated tasks
sapui5_sb         sapui5-1.44, sapui5-1.38       SAPUI5 1.1.0-SNAPSHOT
```

> **Note**
>
> The file system service is not available on SAP Cloud Platform. The `sapui5_sb` service is only available on XS advanced on-premise.

Related Information

Maintaining Application Services in XS Advanced [page 826]
Core Services in XS Advanced [page 827]
Service Plans and Resources in XS Advanced [page 856]
XS CLI: Services Management

8.5 Create a Service Instance

Make a service instance available to applications.

Context

Developers must create an instance of a service before it can be used and then bind the service to the application that needs to use it. The service instance also requires a **service broker**.

To create an instance of a service controlled by an SAP HANA service broker, perform the following steps:

Procedure

1. Create an instance of a **service broker**.

   The service-broker instance requires a name, for example, “myServiceBroker”:

   ```
   xs create-service-broker <myServiceBroker> <USER> <PASSWORD> http://<service-broker-url>
   ```

2. Check that the new service instance is up and running.

   Check details of the new service by displaying a list of the service instances available in the service **market place**:

   ```
   xs marketplace
   ```

3. Create an instance of the **service**.

   The new service instance requires a name, for example, “myService”and is controlled by the service-broker instance you already created:

   ```
   xs create-service myService default <myServiceInstance>
   ```

   > → Tip
   >
   > In this example of how to create a service, “`default`” refers to the service plan to use. The HDI and UAA services provide additional service plans.

4. Bind the service instance to an application.

   ```
   xs bind-service <myApp> <myServiceInstance>
   ```

5. Verify that the application has been bound to the service instance.

   To confirm the application-service binding, check the environment variable `<VCAP_SERVICES>`.

   ```
   xs env <myApp>
   ```

6. Unbind the service instance from an application.

   ```
   xs unbind-service <myApp> <myServiceInstance>
   ```

7. Delete the service instance.

   ```
   xs delete-service <myServiceInstance>
   ```

8. Remove the service broker you created.

   ```
   xs delete-service-broker <myServiceBroker>
   ```

9. Verify that the broker has been deleted.

   To confirm that the service broker has been removed, check the service marketplace:

   ```
   xs marketplace
   ```

Related Information

The Service-Broker CLI [page 863]
Maintaining Application Services in XS Advanced [page 826]

8.5.1 The Service Brokers

Service brokers manage instances of services used by applications running in the SAP HANA XS advanced model run time.

In SAP HANA, the service broker is responsible for maintaining and managing instances of service brokers and all available services. Services are bound to the application that requires the services provided by a service instance. Developers can use the service broker to bind service instances to the application that wants to make use of the advertised services.

> **i** Note
>
> The SAP HANA service broker is only used for SAP HANA **database-level** services, for example: schemas, HDI containers, secure stores, and the Service Broker Security Service (SBSS). The Job Scheduler and the User Account and Authentication (UAA) service each has its own service broker that is used to manage instances of the corresponding service.

A dedicated application-programing interface (API) is provided to enable access to the service broker and any managed services; the API enables you to create and delete service instances (as well as instances of any service brokers that control them) and bind the services to an application. You can also use the commands `xs`

`marketplace` and `xs services` to display a list of the currently available services, either in the marketplace or in a target organizational space.

Related Information

Create a Service Instance [page 861]
The Service-Broker CLI [page 863]
Maintaining Application Services in XS Advanced [page 826]

8.5.2 The Service-Broker CLI

Maintain and manage SAP HANA XS, advanced model, services and service brokers, for example: list, create, delete, bind, and update.

```
xs <Service_Broker_COMMAND> <Service_Broker_NAME>
```

```
xs <Service_COMMAND> <Service_NAME>
```

XS CLI: Services Command Overview

Command	Alias	Description
`marketplace`	`m`	List available services in the marketplace
`services`	`s`	List all services in the target space
`managed-services`		List all instances of a managed service in the target space
`managed-service`		Display details of a selected instance of a managed service in the target space
`create-service`	`cs`	Create a service instance
`update-service`		Update a service instance
`delete-service`	`ds`	Delete a service instance
`rename-service`		Rename a service instance
`bind-service`	`bs`	Bind a service instance to an application
`unbind-service`	`us`	Remove the binding between a service instance an an application

Command	Alias	Description
`service-keys`	`sks`	List all service keys for a specified service instance.
`service-key`	`sk`	Displays a service key for a specified service instance.
`create-service-key`	`csk`	Create a service key for a specified service instance.
`delete-service-key`	`dsk`	Delete a service key for a specified service instance.
`service-brokers`		List all available service brokers
`create-service-broker`		Create a service broker
`delete-service-broker`		Delete a service broker
`update-service-broker`		Update a service broker
`rename-service-broker`		Rename a service broker
`create-user-provided-service`	`cups`	Create a user-provided service instance and make it available for use by applications. *
`update-user-provided-service`	`uups`	Update the name-values pairs used to define a user-provided service instance
`register-service-url`		Register a Uniform Resource Locator (URL) for a named service.
`unregister-service-url`		Unregister a service URL

> **i** Note
>
> To access databases not managed by the SAP HANA Service Broker, you can create “user provided services”. To create a user-provided service, you must provide the service credentials for access to the database and bind the service to the application.

Related Information

Create a Service Instance [page 861]
The Service Brokers [page 862]
XS CLI: Services Management

8.6 Update a Service Instance

Update configuration details of an existing service instance.

Context

You can modify details of an existing service and make the modified service available to the applications that use it. As part of the update operation, you can perform the following tasks:

- Change the service plan associated with a service
- Modify the service configuration
  Configuration details can be provided either in a JSON formatted file or as a JSON-compliant string as an argument on the command line.

To update the configuration of an existing service plan in SAP HANA XS advanced run time, perform the following steps:

Procedure

1. Log on to the SAP HANA XS advanced run time where the service that you want to update is running.

   ```
   xs login -a https://hanahost.acme.com:30030 -u <Username> -p <Password> -o
   <myorg> -s <myspace>
   ```

   Output Code

   ```
   USERNAME: XSMASTER
   PASSWORD>
   ORG: myorg
   SPACE: myspace
   API endpoint:   https://hanahost.acme.com:30030 (API version: 1)
   User:           Username
   Org:            myorg
   Space:          myspace
   ```

2. Check that the service instance you want to update is running.

   To display a list of the service instances running in the current organization and space, run the following command in a command shell:

   ```
   xs services
   ```

   SAP HANA XS advanced displays a list of the currently available services:

   Output Code

   ```
   xs services
   Getting services in org "myorg" / space "SAP" as XSMASTER...
   Found services:
   name                          service   plan          bound apps
   ```

```
-------------------------------------------------------------------------
hdi-container                hana      hdi-shared
hdi-schema                   hana      schema
uaa                          xsuaa     default
deploy-service-ss            hana      securestore   deploy-service
deploy-service-database      hana      schema        deploy-service
deploy-service-uaa           xsuaa     default       deploy-service
product-installer-database   hana      schema        product-installer
product-installer-uaa        xsuaa     default       product-installer
```

3. Update details of an existing service plan.

   You can change the service plan associated with an existing service.

   > **!** Restriction
   >
   > It is not possible to update the service plan assigned to an instance of the XS User Account and Authentication (XSUAA) service.

   To change the service plan associated with a running service, use the `xs update-service` command as follows:

   ```
   xs update-service <SERVICE_INSTANCE> -p <NEW_SERVICE_PLAN>
   ```

4. Update details of an existing service's configuration.

   Some services are configured with a service descriptor. For example, the XSUAA service for user authentication is configured using the security descriptor `xs-security.json`. You can change selected details of the security configuration and update the service with the modified configuration.

   To modify details of an existing XSUAA service's security configuration, use the `xs update-service` command as follows:

   ```
   xs update-service <SERVICE_INSTANCE> -p <SERVICE_PLAN> -c xs-security.json
   ```

   > **!** Restriction
   >
   > When updating the configuration of the XSUAA service, you can add or delete any defined scopes, any security-related attributes, and any role templates. However, you can only **modify** the following elements of an existing service's security configuration:
   >
   > - A security scope's description and “granted applications”
   > - A security attribute's description
   >   When updating the XSUAA service, it is not permitted to change the value type of an attribute defined in the `xs-security.json` application security descriptor.
   > - A role template's description, scope reference, or attribute reference

Related Information

Create a Service Instance [page 861]
Set up Application Security in XS Advanced [page 978]

8.7 Set up and Use an Application-Managed Instance

Set up the Instance Manager and consume an application-managed instance and the corresponding application programming interface (API) in XS advanced.

Prerequisites

- An Instance Manager is running and registered as service broker (check with the command `[xs|cf] service-brokers`).
- The Instance Manager provides a service (check with the command `[xs|cf] marketplace`).

> → Tip
>
> You can identify an Instance Manager service in the command output by the pattern '`managed-<service>`', for example, '`managed-hana`'.

Context

Multi-tenant-aware applications need to be able to create additional instances of an Instance Manager service dynamically, on-demand at run time. Whenever a new tenant is added, the application creates a new application-managed instance, for example, where each tenant stores its data in a separate HDI container. These applications also need to be able to connect to any one of these application-managed instances when processing a request for a specific database tenant.

Procedure

1. Create a service instance of the Instance Manager service in the desired space.

   ```
   [xs | cf] create-service <managed-service> <managed-plan> <service-instance>
   ```

   > ℹ Note
   >
   > The created service instance does not create any artifacts in the corresponding managed broker (for example, the HDI-Broker); it simply stores the parameters used for creating the application-managed instance per tenant on demand.

2. Consume the Instance Manager application programming interface (API).

   To consume the Instance Manager API used for creating the actual service instance, bind your application to the created service instance of the Instance Manager.

   ```
   [xs | cf] bind-service <myApp> <service-instance>
   ```

The binding process, if successful, adds a section called "managed-<service>" to the <VCAP_SERVICES> environment variable, as illustrated below:

```
[xs | cf] env <myApp>
```

```
VCAP_SERVICES {
...
"managed-hana" : [ {
   "name" : "tenant-hdi-container",
   "label" : "managed-hana",
   "tags" : [ "managed-hana", "managed-database", "managed-relational", "xsa-
instance-manager",],
   "plan" : "hdi-shared",
   "credentials" : {
      "user" : "UqWHx9u1PryJKgcc3_okomDk5s1UU2ZJ",
      "password" : "DiRM06NZUqz_8eIudAtw8FyprcxqE8wX",
      "post_managed_instance_url" : "https://<host>:<port>/<path>/
{tenant_id}",
      "get_managed_instance_url" : "https://<host>:<port>/<path>/{tenant_id}",
      "delete_managed_instance_url" : "https://<host>:<port>/<path>/
{tenant_id}"
}}]}
```

3. Applications can send HTTP/REST requests to the Instance Manager using the URLs provided in the `credentials` stored in <VCAP_SERVICES>.

   You must add the following information:

   - Replace the path variable `{tenant_id}` in the URL (for example, `post_managed_instance_url`) with the corresponding ID of the tenant.
   - Set a basic authentication with `user:password` in the authentication header (`'Authentication'`).

   The response body is always in JSON format. The response returned for unsuccessful requests has the following form:

   ```
   {"description" : "<Description text>"}
   ```

Related Information

Managed Service Instance API URLs and Properties [page 868]
Core Services in XS Advanced [page 827]

8.7.1 Managed Service Instance API URLs and Properties

Multi-tenant-aware applications can connect to the Instance Manager by using the URLs defined in the VCAP_SERVICES environment variable.

If successful, the application-service binding process adds a section called 'managed-<service>' to the VCAP_SERVICES environment variable; the new section contains information that is used by the application to connect to the Instance Manager and looks like the following example:

```
managed-hana" : [ {
   "name" : "tenant-hdi-container",
```

```
   "label" : "managed-hana",
   "tags" : [ "managed-hana", "managed-database", "managed-relational", “xsa-
instancemanager” ],
   "plan" : "hdi-shared",
   "credentials" : {
      "user" : "UqWHx9u1PryJKgcc3_okomDk5s1UU2ZJ",
      "password" : "DiRM06NZUqz_8eIudAtw8FyprcxqE8wX",
      "post_managed_instance_url" : "https://<host>:<port>/<path>/{tenant_id}",
      "get_managed_instance_url" : "https://<host>:<port>/<path>/{tenant_id}",
      "delete_managed_instance_url" : "https://<host>:<port>/<path>/{tenant_id}"
}}]
```

The following table describes the URLs provided in the `credentials` property in the `<VCAP_SERVICES>` environment variable:

> **i Note**
>
> Creating and deleting the application-managed instance is done asynchronously to the corresponding `POST` or `DELETE` request on the Instance Manager service instance.

URL Properties

URL property	HTTP Method	Description	Result
`post_managed_instance_url`	POST	Asynchronously creates a new application-managed service instance in the managed broker for the specified tenant. Response body: • Success: `{}` • Failure: `{<error message>}`	Returns HTTP status code: • 201: Creation success • 400: Path variable '`tenantId`' is missing • 409: Application-managed instance already exists for this tenant
`get_managed_instance_url`	GET	Returns the application-managed instances for the specified tenant. Response body: • Success: the application-managed instance in JSON format [page 871] • Failure: `{<error message>}` **→ Tip** Cache the credentials of an application-managed instance as soon as the `GET` operation indicates a final state. This will minimize the interaction between the application and the Instance Manager and increase resilience.	Returns HTTP status code: • 200: GET success • 404: No application-managed instance exists for this tenant

URL property	HTTP Method	Description	Result
`delete_managed_instance_url`	DELETE	Asynchronously deletes an existing application-managed instance for the specified tenant. The corresponding application-managed instance shows the status of the deletion process. Response body: • Success: `{}` • Failure: `{<error message>}`	Returns HTTP status code: • 204: Deletion success • 400: Path variable '`tenantId`' is missing • 404: No application-managed instance exists for this tenant

During the tenant on-boarding process, a client first sends a `POST` request to the Instance Manager with the corresponding tenant ID. To check that the application-managed instance is available, the client polls the Instance Manager service instance by sending `GET` requests until the status has reached a final state (for example, `CREATION_SUCCEEDED`) and the result is returned and can be consumed. The information returned contains either the required credentials (if the operation was successful) or an error message if there was a problem. The same is true for retrieving a service instance, if applications can request service instances during tenant on-boarding (in parallel).

→ Tip

It is highly recommended to ensure that the applications using the Instance Manager's API can cache the credentials of an application-managed instance, for example, as soon as the `GET` operation indicates a final state. Caching the credentials minimizes the interaction required between the application and the Instance Manager.

The current status of the creation or deletion process can be retrieved by getting the application-managed instance with `GET`, which returns something like the following example:

Sample Code

Getting the Status of an Application-Managed Instance Operation

```
{
  "tenant_id": "tenantX",
  "id": "0db6b276-a22f-43ee-9709-364e0fd152ae",
  "binding_id": "0b550fb1-9511-4e34-beca-c35c2dde3fc8",
  "instance_id": "70a4bf6a-7597-4da3-8549-af7d07f055e4",
  "managed_service_id": "4a20979a-1c8b-4c4a-a602-345e1dce685b",
  "managed_plan_id": "5f5dcb19-1798-4a35-b4d5-a461575175ee",
  "managed_instance_id": "dfda313c-6261-46ea-9c16-10500f342417",
  "managed_binding_id": "9b15f637-33e3-49bc-914b-865d66b80026",
  "status": "CREATION_IN_PROGRESS"
}
```

The information describing the status of an application-managed instance is contained in the properties described in the following table:

Elements of the Response Body on GET Requests

Property	Description	Notes
`tenant_id`	The id of the tenant.	-

`id`
`binding_id`
`managed_binding_id`
`instance_id`
`managed_instance_id`
`service_id`
`managed_service_id`
`plan_id`
`managed_plan_id`
	This information is needed in a support case.	-

`status`	The current status of the application-managed instance:	CREATION_IN_PROGRESS CREATION_SUCCEEDED CREATION_FAILED DELETION_IN_PROGRESS DELETION_FAILED
`credentials`	JSON bearing the credentials from managed broker binding if the requested operation was successful	Status: CREATION_SUCCEEDED
`failed_message`	The error message from the managed broker if the requested operation failed.	Status: CREATION_FAILED DELETION_FAILED

Related Information

Set up and Use an Application-Managed Instance [page 867]
Core Services in XS Advanced [page 827]

8.8 Set up and Use the XS Advanced Message Service

Deploy, set up, and use the inter-application messaging service for XS advanced.

Prerequisites

You have access to the messaging service package `XS MESSAGING SERVICE 1`.

> → Tip
>
> The messaging service for XS advanced is available for download on SAP Service Marketplace at the following location for those people with the required S-User ID:
>
> *Service Marketplace* > *Products* > *Software download* > *SUPPORT PACKAGES & PATCHES* > *By Alphabetical Index (A-Z)* > *H* > *SAP HANA PLATFORM EDITION* > *SAP HANA PLATFORM EDITION 2.0* > *XS MESSAGING SERVICE 1*

Context

To deploy, set up, and use the messaging service for XS advanced applications, perform the following steps:

Procedure

1. Download and unpack the messaging service `XS MESSAGING SERVICE 1` archive to folder in the local file system.

   > → Tip
   >
   > `XS MESSAGING SERVICE 1` contains the file `messaging-service-<VERSION>.zip`, which you use to install the messaging service in XS advanced.

2. Install the messaging service for XS advanced.

   In an XS CLI shell, log onto the XS advanced run-time platform, organization, and space, where you want to deploy the XS advanced messaging service.

   > i Note
   >
   > The version of the messaging-service file in the following example (`-1.3.1`) is for illustration purposes only; it will change with time.

   ```
   xs install messaging-service-1.3.1.zip
   ```

3. Update the messaging service for XS advanced.

   Sample Code

   ```
   xs install
   ```

4. Check that the messaging-service applications were successfully deployed to the XS advanced run-time environment.

   Sample Code

   The messaging service provides three applications that form the network of brokers for ActiveMQ. You can display a list of the applications in a command shell with the command `xs apps`, as illustrated in the following example:

   Sample Code

   Available Messaging Applications

   ```
   $ xs apps
   Getting apps in org "myOrg" / space "DEV" as admin...
   Found apps:
   name                      requested  instances  memory  disk   urls
                             state
   ------------------------------------------------------------------------
   --
   messaging-service-hub     STARTED    1/1        512 MB  <...>  tcps://
   host:51
   messaging-service-node    STARTED    1/1        512 MB  <...>  tcps://
   host:51
   messaging-service-broker  STARTED    1/1        512 MB  <...>  tcps://
   host:51
   ```

5. Check that the messaging-service is available in the XS advanced marketplace.

   After installing the messaging service on an XS advanced system, you can see the messaging service displayed in the list of services registered in the Service Marketplace, as illustrated in the following example:

   Sample Code

   Messaging Service in the Marketplace

   ```
   $ xs marketplace
   Getting services from marketplace in org myOrg / space DEV as admin...
   OK
   service            plans                       description
   ------------------------------------------------------------------------
   --
   fs-storage         free                        Environment variable
   denotes...
   hana               hdi-shared, sbss, schema    SAP HANA database...
                      securestore
   xsuaa              default, devuser, space     XS UAA Service Broker for
   user..
   ...
   messaging-service  free                        Messaging Service Broker on
   XSA
   ```

6. Create an instance of the message service.

```
xs create-service messaging-service free my-message-service
```

7. Deploy the application that you want to subscribe to the messaging service.

```
xs push subscribing-app
```

8. Subscribe an application to the message service.

   Bind to the message-service instance the application you want to subscribe to this message service.

```
xs bind-service subscribing-app my-message-service
```

9. Subscribe the application to the ActiveMQ message-service.

   To subscribe to the ActiveMQ message service queue, the application must use the URL and connection credentials provided by the message-service broker. For more information, see *Consuming Messages with XS Advanced Message Service* in *Related Information*.

10. Unsubscribe an application from the message service.

    Unbind from the message-service instance the application whose subscription to a message service you want to remove.

```
xs unbind-service UNsubscribing-app my-message-service
```

Related Information

Messaging Services in XS Advanced [page 847]
Consuming Messages with the XS Advanced Message Service [page 874]

8.8.1 Consuming Messages with the XS Advanced Message Service

Configuring a client application to generate and consume messages with the XS advanced messaging service.

The XS advanced messaging service utilizes the Apache ActiveMQ Library with the specification defined in the Java Message Service (JMS). ActiveMQ provides client application programming interfaces (API) for many languages, for example: Java, C/C++, .NET, Perl, PHP, Python, Ruby, and so on.

The XS advanced messaging service comprises the following three applications:

- `messaging-service-broker`
- `messaging-service-hub`
- `messaging-service-node`

The `messaging-service-node` application instance starts an ActiveMQ broker with one transport connector. This connector provides client-to-broker communication that is configured to accept incoming connections using the Advanced Message Queuing Protocol (AMQP) 1.0. Messaging tools that speak AMQP and support

many languages and platforms can be used to manage the messages that are generated and consumed. Using `qpid-jms-client` and `activemq-all` libraries, the required dependencies in the `pom.xml` would look like those displayed in the following example:

Sample Code

Library Dependencies in the `pom.xml` File

```
<dependency>
    <groupId>org.apache.activemq</groupId>
    <artifactId>activemq-all</artifactId>
    <version>5.15.2</version>
</dependency>
<dependency>
    <groupId>org.apache.qpid</groupId>
    <artifactId>qpid-jms-client</artifactId>
    <version>0.27.0</version>
</dependency>
<dependency>
    <groupId>com.sap.xs.java</groupId>
    <artifactId>xs-env</artifactId>
    <version>${xs.env.version}</version>
</dependency>
```

Binding Client Applications to the XS Advanced Messaging Service

Client applications that want to use the XS advanced messaging service should be bind to an instance of the messaging service, for example, using the `xs bind-service` command in a command shell, as illustrated in the following example:

```
xs bind-service subscribing-client-app my-message-service
```

The credentials for connecting to the messaging cluster are provided by the messaging broker (`messaging-service-broker`) in the service binding and can be retrieved with the `xsenv` library classes `com.sap.xs.env.VcapServices` and `com.sap.xs.env.Credentials`, as illustrated in the following Java application example:

Sample Code

Connection Credentials for the XS Advanced Messaging Service

```
import com.sap.xs.env.Credentials;
import com.sap.xs.env.Service;
import com.sap.xs.env.VcapServices;
,,,
public void getCredentials() throws Exception {

VcapServices services = VcapServices.fromEnvironment();
Service activeMQStorage = services.findService("messagingservice", "", "");
Credentials credentials = activeMQStorage.getCredentials();
String user = credentials.getUser();
String password = credentials.getPassword();
String brokerUrl = "failover:(" + credentials.getUrl() + ")";
}
```

→ Tip

A JMS client that sends messages is known as a producer; a JMS client that receives messages is known as a consumer.

Producing Messages with a Java Application

The client application that produces messages should provide the connection credentials to `jms` session and `connection` classes together with the name of the destination that the client application wants to send messages to. In the following example, `my.test.queue` is the name of the destination for the message producer, and the JMS class `MessageProducer` is used to send messages to the desired destination. The producer is created using the `Session.createProducer()` method, and messages are actually sent using the `MessageProducer.send()` method.

Sample Code

```
import javax.jms.Connection;
import javax.jms.DeliveryMode;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.MessageProducer;
import javax.jms.Session;
import javax.jms.TextMessage;
,,,
public void sendMessage() throws Exception {

//setup jmx connection
JmsConnectionFactory connectionFactory = new JmsConnectionFactory(brokerUrl);
Connection connection = connectionFactory.createConnection(user, password);
connection.start();

//create jmx session
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

//create destination "my.test.queue" and setup producer
Destination destination = session.createQueue("my.test.queue");
MessageProducer producer = session.createProducer(destination);
producer.setDeliveryMode(DeliveryMode.PERSISTENT);

//send custom message
TextMessage textMessage = session.createTextMessage("This message will be
send to my test queue.");
producer.send(textMessage);

//close producer, session and connection
producer.close();
session.close();
connection.close();
}
```

Consuming Messages with a Java Application

An example application that consumes messages generated by another application requires a similar setup to the one required for generating messages; the major difference is that the consuming application must take

care of receiving messages rather than sending them. For example, it is necessary to create and manage a `jmx` "sender". The client consuming application must make the connection credentials received from the message broker available to the `jms` session and `connection` classes along with the name of the destination that the client application wants to receive messages from.

> → Tip
>
> JMS clients use the JMS `MessageConsumer` class to consume messages from a defined destination, and the destination is set when the (JMS client) consumer is created using the `Session.createConsumer()` method. The `MessageConsumer` class can consume messages synchronously by using one of the `receive()` methods.

> ‹≡› Sample Code

```
//setup jmx connection
JmsConnectionFactory connectionFactory = new JmsConnectionFactory(brokerUrl);
Connection connection = connectionFactory.createConnection(user, password);
connection.start();

// consider setting an exception listener to the connection,
// so you could be notified in cases of connectivity problems
// connection.setExceptionListener(this);

//create jmx session
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

//create destination "my.test.queue" that the consumer will read messages from
Destination destination = session.createQueue("my.test.queue");
MessageConsumer consumer = session.createConsumer(destination);

//consumer starts to receive messages for the given time period - 1000ms
Message message = consumer.receive(1000L);
if (message != null) {
      out.println("received message! -> " + (message instanceof TextMessage ?
((TextMessage) message).getText() : String.valueOf(message)));
                    }

//close consumer, session and connection
consumer.close();
session.close();
connection.close();
```

Monitoring Message-Service Communication with a Java Application

JMS makes use of the concepts of queues and topics. The following types of topics can exist in the Messaging Service:

- Topics created by the client application
- Topics created and used by the ActiveMQ JMS brokers for internal communication. These topics are called "advisory topics" and enable you to monitor the system using regular JMS messages.

An example monitoring Java application needs to support the previously described client application requirements, for example:

- Bind the client application to a message-service instance

- Get the connection credentials from the service broker
- Provide the connection credentials to the connection and session objects.

In addition, in order to monitor the communication, you can use `org.apache.activemq.advisory.AdvisorySupport`. For example, the following methods are available:

- `AdvisorySupport.getProducerAdvisoryTopic()`
  Displays information about "`ActiveMQ.Advisory.Producer.Queue.>`"
- `AdvisorySupport.getConsumerAdvisoryTopic()`
  Displays information about "`ActiveMQ.Advisory.Consumer.Queue.>`" .

Sample Code

```
import javax.jms.Connection;
import javax.jms.Destination;
import javax.jms.MessageConsumer;
import javax.jms.Session;

import org.apache.activemq.advisory.AdvisorySupport;
import org.apache.activemq.command.ActiveMQQueue;
import org.apache.qpid.jms.JmsConnectionFactory;
,,,

//setup jmx connection
JmsConnectionFactory connectionFactory = new JmsConnectionFactory(brokerUrl);
Connection connection = connectionFactory.createConnection(user, password);
connection.start();

//create jmx session
Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

//create jmx destination
Destination advisoryProducerDestination =
AdvisorySupport.getProducerAdvisoryTopic(new ActiveMQQueue(destinationName));

MessageConsumer advisoryProducerQueue =
session.createConsumer(advisoryProducerDestination);

// class AdvisoryListener implements MessageListener
advisoryProducerQueue.setMessageListener(new AdvisoryListener());
```

Python Application with the Messaging Service

As part of Apache's Qpid project that aims to facilitate communication for messaging purposes using the Advanced Message Queuing Protocol (AMQP), Qpid Proton provides a convenient way to integrate with the AMQP 1.0 ecosystem from different platforms and languages. For the purposes of this Python example, we use the library `python-qpid-proton`.

Binding A Python Client Application to a Messaging Service Instance

Any client application that wants to use the XS advanced messaging service must be bound to an instance of the messaging service. The credentials required to connect to the messaging cluster are provided by the messaging broker. To obtain these credentials, you can use the `cfenv` application, which must then

be named (along with the required version) in the list of dependencies stored in the Python application's `requirements.txt` file, as illustrated in the following example:

Sample Code

Specifying Required Libraries in Python Applications

```
--- requirements.txt ---
cfenv==0.5.3
python-qpid-proton==0.20.0
```

The following example shows how to enable a Python application to use `cfenv` to obtain the credentials required to connect to the messaging cluster:

Sample Code

```
from cfenv import AppEnv

def get_address_to_connect(cred):
    if cred is None:
        return "Bind to messaging service instance!"
    else:
        user = cred.get('user')
        password = cred.get('password')
        url = cred.get('url')
        cert = url[:8]
        host = url[8:]
        destination = 'test.my.queue'
        address = "%s%s:%s@%s/%s" % (cert, user, password, host, destination)
        return address

env = AppEnv()
service = env.get_service(label='messaging-service').credentials
```

Producing and Consuming Messages with a Python Application

The sender class (`sender.py`) and receiver class (`receiver.py`) can be used to enable a Python application to produce and consume messages using the XS advanced message services. The following example shows how to enable a Python application to **generate** messages in the XS advanced message service:

Sample Code

Enabling Python Apps to Generate Messages in the XS Message Service

```
from proton.reactor import Container
import Sender

def index():
    # service – credentials obtained from the previous section
    address = get_address_to_connect(service)

    try:
       sender = Sender(address, content.messages)
       Container(sender).run()
       messages = sender.total_messages
       def generate():
           received_messaged_count = 0
           for message in messages:
               received_messaged_count += 1
                time.sleep(1)
                yield 'sending message: %s<br/>\n' % (message)
```

```
    except KeyboardInterrupt:
       pass
    return Response(generate(), mimetype='text/html')
```

The following example shows how to enable a Python application to consume messages managed by the XS advanced message service:

Sample Code

Enabling Python Apps to Consume Messages in the XS Message Service

```
from proton.reactor import Container
import Receiver

def index():
    # service – credentials obtained from the previous section
    address = get_address_to_connect(service)

    try:
        receive = Receiver(address, 30)
        Container(receive).run()
        messages = receive.total_messages
        def generate():
            received_messaged_count = 0
            yield 'Starting to receive messages: Current count = %s<br/>\n' %
(received_messaged_count)
            for message in messages:
                received_messaged_count += 1
                time.sleep(1)
                yield 'received message! -> %s<br/>\n' % (message)
    except KeyboardInterrupt:
       pass
    return Response(generate(), mimetype='text/html')
```

Message Expiration Time

Messages can be configured to have an optional expiration time. If this "time-to-live" configuration is not set, for example, by the message producer, the default value of five (5) seconds is applied. The XS advanced messaging service does not deliver a message to a consumer if the message's expiration time has been exceeded.

The XS advanced messaging service uses the parameter `TimeStampPlugin` to configure the default message expiration time. For more information, see *TimeStampPlugin* in *Related Information* below.

Note

If the message producer sets a custom message-expiration time that exceeds the maximum permitted value of 30 seconds, the custom setting is ignored and the maximum permitted value of 30 seconds is applied instead.

From version 1.4.6 of the XS advanced messaging service, you can extend the limit of the "time-to-live" configuration during an application's installation.

The following example shows how to create an extension descriptor in the yaml file containing the `MESSAGES_TTL_CEILING` property for `messaging-service-hub` and `messaging-service-node`. The value of `MESSAGES_TTL_CEILING` will be the new limit of the "time-to-live" configuration for every message.

Sample Code

Extending the limit of the "time-to-live" configuration

```
_schema-version: "2.0.0"
extends: com.sap.xs.messaging-service.xsac.mess.srv
ID: com.sap.xs.messaging-service.xsac.mess.srv.extends
modules:
- name: messaging-service-hub
    properties:
    MESSAGES_TTL_CEILING: 41000
- name: messaging-service-node
    properties:
    MESSAGES_TTL_CEILING: 41000
```

You can install the XS advanced messaging service with an extension descriptor, for example, using the `xs install` command in a command shell, as illustrated in the following example:

```
xs install messaging-service-1.4.6.zip -e path/to/extension/descriptor.yaml
```

Setting the Message Expiration Time with the Producer

To set a message expiration value through the producer, see the recommendations provided in the documentation from Apache ActiveMQ. For more information, see *How do I set the message expiration* in *Related Information* below.

Related Information

Messaging Services in XS Advanced [page 847]

Set up and Use the XS Advanced Message Service [page 872]

TimeStampPlugin

How do I set the message expiration

8.9 Scheduling Jobs in XS Advanced

The Job Scheduler service enables you to create and schedule long-running operations or jobs.

In the SAP HANA XS advanced model, the Job Scheduler is an application service. The Job Scheduler service enables you to create and schedule long-running operations or jobs. This service is deployed during the installation of the SAP HANA XS advanced model.

The following table lists the sequence of tasks required to use an instance of the Job Scheduler service:

Note

To configure and setup Job Scheduler you require specific roles and permissions.

Step	Task	Role
1	Configure the Service Broker for Job Scheduler	Space Developer
2	Create a Job Scheduler Service Instance	Space Developer
3	Bind an Application to the Job Scheduler Service	Space Developer
4	Maintain jobs and job schedules	Administrator

Related Information

Job Scheduler Services in XS Advanced [page 850]
Roles for Running the Job Scheduler [page 883]
Configure the Service Broker for Job Scheduler [page 882]
Bind an Application to the Job Scheduler Service [page 885]
Maintain Jobs and Job Schedules in XS Advanced [page 886]

8.9.1 Configure the Service Broker for Job Scheduler

You need to configure the service broker to consume the Job Scheduler service.

Prerequisites

You have the Space Developer role. For more information about Organizations and spaces, see the topic *Maintaining Organizations and Spaces in SAP HANA XS Advanced Model* in the *SAP HANA Administration Guide*.

Context

A successful installation of the SAP HANA XS advanced model runtime ensures that the Job Scheduler service is deployed.

Procedure

1. Set the credentials for the Job Scheduler service broker.

```
xs set-env jobscheduler-broker brokeruser <user name>
```

```
xs set-env jobscheduler-broker brokerpassword <password>
```

2. Restart the broker.

```
xs restart jobscheduler-broker
```

3. Register the service broker to consume the service.

```
xs create-service-broker <serviceName> <userName> <password> <brokerURL>
```

> → Tip
>
> `<userName>` and `<password>` are the ones that were set when registering credentials in the previous step. To check the broker URL, log on to the space where the application is deployed and use the command `xs apps`. In the list of deployed applications, check the URL corresponding to the jobscheduler-broker application.

8.9.1.1 Roles for Running the Job Scheduler

You need to have specific roles to work with the Job Scheduler service.

A role defines the permission (scope) assigned to your user. For example, with the Space Developer role you can create a service instance, bind an instance with an application, or unbind a service instance and so on.

The roles and permissions that you require depend on the type of tasks that a user is required to perform. As a technical user, you will receive all the credentials required to work with the Job Scheduler service from `<VCAP_SERVICES>` at runtime. At application runtime, the service provides the credentials required for the application in the `<VCAP_SERVICES>` environment variable to communicate with the service instance. The application code parses the `<VCAP_SERVICES>` variable and extracts the credentials. These credentials are unique for a binding. The binding process generates the technical user. The Space Developer role will enable you to perform all the technical user tasks.

To perform tasks on the Job Scheduler dashboard, you need to have specific roles. You need the SAP HANA administrator role to assign the roles specific to Job Scheduler. To assign roles as a SAP HANA administrator, access the Job Scheduler dashboard. To assign specific roles to the end users of the Job dashboard, choose *User Management* in the home navigation pane. Using *User Management* create a new role collection and then add Job Scheduler roles. For more information about role collection, see *Manage Users* in the *SAP HANA Administration Guide*. The Job Scheduler roles are:

- Administration:
  This role enables the end-user to edit jobs and schedules.
- Viewer
  This role enables the end-user to access the user interface in read-only mode.
- Configuration
  This role enables the end-user to maintain the global configurations of the service through the configuration user interface.

For more information about roles and templates, see *Maintaining Security in XS Advanced* in the *SAP HANA Administration Guide*.

Related Information

The Job Scheduler Dashboard [page 906]

8.9.2 Create a Job Scheduler Service Instance

To use a Job Scheduler service, an application requires a Job Scheduler service instance.

Prerequisites

- The Job Scheduler service is installed as part of the SAP HANA XS advanced model installation.
- You have registered the service broker.

Context

You create a Job Scheduler service instance to use the service.

Procedure

1. Check for Job Scheduler service details in a space.

   ```
   xs marketplace
   ```

2. Create an instance of the Job Scheduler service.

   ```
   xs create-service <service name> <service plan name><service instance name>
   ```

   ```
   xs create-service jobscheduler default myinstance
   ```

Related Information

Configure the Service Broker for Job Scheduler [page 882]

8.9.3 Bind an Application to the Job Scheduler Service

To use a Job Scheduler service, you have to bind an application with the Job Scheduler service.

Prerequisites

- The service broker and the service instance for the Job Scheduler service are available.
- The application required for binding is deployed in the space.

Context

You bind an application to the Job Scheduler service after creating a Job Scheduler service instance.

Procedure

1. Check if an instance of the Job Scheduler service is available for binding.

   Use the command `XS services` to check for Job Scheduler instances.
2. Bind the Job Scheduler service to the application.

   ```
   xs bind-service <application name><service instance name>
   ```
3. Restart the binding.

   ```
   xs restart <application name>
   ```
4. Check if the application is bound.

   Check the details of the VCAP_SERVICES in the environment variable.

   ```
   xs env <application name>
   ```

Results

> **i** Note
>
> To unbind a service instance from an application, use the command:
>
> ```
> xs unbind-service <application name> <service instance>
> ```

To delete a service instance, use the command:

```
xs delete-service instance
```

Related Information

Create a Job Scheduler Service Instance [page 884]

8.9.4 Maintain Jobs and Job Schedules in XS Advanced

Maintain run time configurations for jobs and job schedules in SAP HANA XS advanced.

Prerequisites

- The service broker and the service instance for the Job Scheduler service are available.
- The application using the Job Schedule is deployed in the space and bound to the Job Scheduler service instance.
- You have the authorization scope for `POST`, `PUT`, and `DELETE` requests (for example, *jobscheduler.Admin*).
- To access the *Job Scheduler Dashboard*, you must have the authorization scopes defined in the roles grouped together in one of the following role collections:
  - XS_CONTROLLER_ADMIN
    Full access: no access restrictions
  - XS_CONTROLLER_USER
    Modify and read-only access
  - XS_CONTROLLER_AUDITOR
    Read-only access

> → Tip
>
> Role collections can be assigned to an SAP HANA user in SAP HANA studio by means of user parameters, for example, `XS_RC_XS_CONTROLLER_ADMIN` or `XS_RC_XS_CONTROLLER_USER`, or `XS_RC_XS_CONTROLLER_AUDITOR`.

Context

To maintain jobs and job schedules, you use the Job Scheduler REST APIs (for example, *Job Creation*, *Job Configuration*, or *Job Deletion*) as illustrated in the following examples.

> **Note**
>
> The code examples are not always complete; they are intended for illustration purposes only.

Procedure

1. Create a new job.

   Use the *Job Creation* API (`POST /scheduler/Jobs`), as illustrated in the example request:

   ```
   POST /scheduler/jobs HTTP/1.1
   Host: localhost:4242
   Authorization: Basic YWJjOmRlZg==
   Content-Type: application/json
   Cache-Control: no-cache
   {"name":"validateSalesOrder", "description": "cron job that validates
   sales order requests", "action":"http://salesOrderApp.hana.acme.com:40023/
   salesOrders/validate","active": true, "httpMethod":"PUT", "schedules":
   [{"cron":"* * * * * * */10", "description": "this schedule runs every
   10 seconds", "data":{"salesOrderId":"1234"}, "active": true, "startTime":
   {"date": "2015-10-20 04:30 +0000", "format": "YYYY-MM-DD HH:mm Z"}}]}
   ```

   The response to the job-creation request should look like the following example:

   ```
   {"name":
   "validateSalesOrder", "action":"http://salesOrderApp.hana.acme.com:40023/
   salesOrders/validate","active":true,"httpMethod":"PUT","description":"cron
   job that validates sales order
   requests","startTime":null,"endTime":null,"signatureVersion":0,"schedules":
   [{"active":true,"startTime":"2015-10-20
   04:30:00","endTime":null,"description":"every 10 seconds, every
   2 minutes","data":"{\"salesOrderId\":\"1234\"}","cron":"* *
   * * * */10","type":"recurring","scheduleId":"cb5c9def-
   e2a0-4294-8a51-61e4db373f99"}],"_id":3}
   Headers:
   Connection → keep-alive
   Content-Length → 468
   Content-Type → application/json; charset=utf-8
   Date → Mon, 09 Nov 2016 09:08:53 GMT
   ETag → W/"1d4-P7BnAm3yordzbrYyJtpalg"
   Location → /scheduler/jobs/3
   X-Powered-By → Express
   ```

2. Modify (configure) a new job.

   Use the *Job Configuration* API (`PUT /scheduler/Jobs`), as illustrated in the example request:

   ```
   PUT /scheduler/jobs/3 HTTP/1.1
   Host: localhost:4242
   Authorization: Basic YWJjOmRlZg==
   Content-Type: application/json
   Cache-Control: no-cache
   {"active": true, "user":"abc", "password":"def", "httpMethod": "GET"}
   ```

   The response to the job-configuration request should look like the following example:

   ```
   {"success":true}
   Headers:
   Connection → keep-alive
   ```

```
Content-Length → 16
Content-Type → application/json; charset=utf-8
Date → Mon, 09 Nov 2016 09:30:36 GMT
ETag → W/"10-c2PoX+nt7m8FOksxlYjAhg"
X-Powered-By → Express
```

3. Delete an existing job.

   Use the *Job Deletion* API (`DELETE /scheduler/Jobs`), as illustrated in the example request:

```
DELETE /scheduler/jobs/4 HTTP/1.1
Host: localhost:4242
Authorization: Basic YWJjOmRlZg==
Content-Type: application/json
Cache-Control: no-cache
```

   The response to the job-deletion request should look like the following example:

```
{"success":true}
Headers:
Connection → keep-alive
Content-Length → 16
Content-Type → application/json; charset=utf-8
Date → Mon, 09 Nov 2016 09:30:36 GMT
ETag → W/"10-c2PoX+nt7m8FOksxlYjAhg"
X-Powered-By → Express
```

4. Create a new job schedule.

   Use the *Job Schedule Creation* API (`POST /scheduler/jobs/3/schedules`), as illustrated in the example request:

```
POST /scheduler/jobs/3/schedules HTTP/1.1
Host: localhost:4242
Authorization: Basic YWJjOmRlZg==
Content-Type: application/json
Cache-Control: no-cache
{"repeatEvery":"2 hours","data":{"order_id":"abcd"}, "active":true,
"description":"New Schedule", "startTime":{"date": "2016-04-21", "format":
"YYYY-MM-DD"}}
```

   The response to the job-schedule creation request should look like the following example:

```
"repeatInterval":"2
hours","repeatAt":null,"time":null,"cron":null,"data":"{\"order_id\":\"abcd\"}
","description":"New
Schedule","type":"recurring","active":true,"startTime":"2016-04-21
18:30:00","endTime":null,"jobId":3,"scheduleId":"0e29c67c-563e-4931-
af08-43acb10813e8"}
Headers:
Connection → keep-alive
Content-Length → 274
Content-Type → application/json; charset=utf-8
Date → Mon, 09 Nov 2016 09:42:13 GMT
ETag → W/"112-rdQSXHBVY0u6JNI/Wf0I7w"
Location → /scheduler/jobs/3/schedules/0e29c67c-563e-4931-af08-43acb10813e8
X-Powered-By → Express
```

5. Delete an existing job schedule.

   Use the *Job Schedule Deletion* API (`DELETE /scheduler/jobs/3/schedules`), as illustrated in the example request:

```
DELETE /scheduler/jobs/4 HTTP/1.1
```

```
Host: localhost:4242
Authorization: Basic YWJjOmRlZg==
Content-Type: application/json
Cache-Control: no-cache
```

The response to the job-schedule deletion request should look like the following example:

```
{"success":true}
Headers:
Connection → keep-alive
Content-Length → 16
Content-Type → application/json; charset=utf-8
Date → Mon, 09 Nov 2016 09:51:39 GMT
ETag → W/"10-c2PoX+nt7m8FOksxlYjAhg"
X-Powered-By → Express
```

Related Information

Job Scheduler REST API for XS Advanced [page 889]
The Job Scheduler Dashboard [page 906]
Scheduling Jobs in XS Advanced

8.9.4.1 Job Scheduler REST API for XS Advanced

The Job Scheduler APIs enable applications to use the functionality provided in Job Scheduler.

The Job Scheduler-as-a-Service is a microservice component, which enables you to create, schedule, and run application tasks. The component exposes REST endpoints for interaction, with JSON as the format for data communication. The Job Scheduler API for SAP HANA XS advanced includes the commands listed in the following table. For more information about the configuration parameters required for the request, see the API documentation provided with the *Job Scheduler Dashboard* tool.

> **i Note**
>
> Access to the APIs is controlled by authorization scopes, for example, `admin` for `POST` and `PUT` requests, or `view` for `GET` requests. Scopes are built into roles, which can be assigned to users in role collections. The Job Scheduler REST APIs are protected with basic authentication.

An application, which has been bound to the Job Scheduler service and wants to interact with the Job Scheduler service, must extract the authentication credentials from the `<VCAP_SERVICES>` environment variable and use these credentials to call the REST APIs. To invoke the API, the user-authentication credentials must be encoded and passed in the "Authorization" header. If the credentials are not passed or they are passed wrongly, the APIs return a response with the status code "`401- Unauthorized`".

In this section, you can find information about the following topics:

- Command Overview
- Human-Readable Dates
- Time Formats

Command Overview

XS Advanced Job Scheduler REST API

API	Description	Required Scope
Job Creation	Used to create a job. Job creation can accept a collection of job schedules to be created.	`admin`
Job Configuration	Configure a job with updated run time information. The API can also be used to create a job if a Job with the Job Name in the URI segment, is not found.	`admin`
Job Deletion	Delete a job and purge all its run time information such as job schedules and logs.	`admin`
Job Schedule Creation	Create a job schedule for a specified job. All job configuration values (Action URL, HTTP Method, User, Password & Job Activation Status) are valid for the newly created schedule. A job schedule will only run if both the job and the schedule are active.	`admin`
Job Schedule Modification	Configure the run time information of a job schedule for a specified job. All job configuration values (for example: `Action URL`, `HTTP Method`, `User`, `Password`, and `Job Activation Status`) remain valid for the modified schedule.	`admin`
Job Schedule Deletion	Delete and purge run time information of the job schedule of the specified job. All related information like job schedule configurations and logs are purged. The processing of the schedule is also immediately stopped.	`admin`
Bulk Job Schedule Activation Bulk Job Schedule Deactivation	This is a utility API used to activate or deactivate all existing schedules of a job. This API triggers the immediate processing (or a halt in processing) of all job schedules for the specified job.	`admin`
Job Details	Retrieve the saved details and configurations of a specified job. If the `displaySchedules` parameter is not provided, the schedules for the job are not returned and only the job details are returned.	`view`
Job Schedule Details	Retrieve the saved details and configurations of a specified job schedule & optionally the generated logs for the schedule.	`view`
Bulk Job Schedule Deletion	Delete and purge run time information of all the currently configured job schedules of the specified job. All related information like job schedule configurations and logs are purged. The processing of the schedules is also immediately stopped.	`admin`
Job Run Log Update	Used by applications, to inform the Job Scheduler about the status of an asynchronous, long-running job run.	`admin`

Job Creation

To create a job schedule, at least one of the fields `repeatAt`, `repeatEvery`, `cron` and `time` must be used. The response from the job creation API is a JSON body with the job details, including the ID of the job.

- **Route**
  `POST /scheduler/jobs`
- **Response**
  A JSON body containing the job details, including the ID of the job with status code "`201-CREATED`" , if the call was successful. A location header with the relative path to the job-details is included in the response.

Sample Code

```
{
  "name": "validateSalesOrder",
  "description": "cron job that validates sales order requests",
  "action": "http://salesOrderApp.hana.ondemand.com:40023/salesOrders/
validate",
  "active": true,
  "httpMethod": "PUT",
  "schedules": [
    {
      "cron": "* * * * * */10 0",
      "description": "this schedule runs every 10 minutes",
      "data": {
        "salesOrderId": "1234"
      },
      "active": true,
      "startTime": {
        "date": "2015-10-20 04:30 +0000",
        "format": "YYYY-MM-DD HH:mm Z"
      }
    }
  ]
}
```

```
Response:
{
  "name": "validateSalesOrder",
  "action": "http://<application-url>/action",
  "active": true,
  "httpMethod": "PUT",
  "description": "cron job that validates sales order requests",
  "startTime": null,
  "endTime": null,
  "signatureVersion": 0,
  "schedules": [
    {
      "active": true,
      "startTime": "2015-10-20 04:30:00",
      "endTime": null,
      "description": "every 10 seconds, every 2 minutes",
      "data": "{\"salesOrderId\":\"1234\"}",
      "cron": "* * * * * * */10",
      "type": "recurring",
      "scheduleId": "schedule ID details"
    }
  ],
  "_id": 3
}
```

The job schedule creation request is defined with the parameters listed in the following table:

> **Note**
>
> Parameters marked with an asterisk (*) are mandatory.

Job Creation: Request Body Fields

Request Field	Type	Description
`name` *	String	The unique name of the job to be created **Note** If a job with the same name for the technical user credentials already exists, the job creation request fails.
`description`	String	Describes the user-defined job
`action` *	String	The fully qualified URL endpoint to be called when the job runs, for example: `http://host.acme.com/app/call`
`active`	Boolean	Defines if the job should be activated on creation. Allowed values are: • `false` (default) The job is in inactive mode on creation • `true` The job is activated on creation
`httpMethod`	String	The HTTP method to be used to call the end-point URL for the job action . Allowed values are: GET, POST (default), PUT, and DELETE
`startTime`	Object	The start time for the job. If the start time is specified for the job, the scheduler checks if a start time is provided for the schedule as well. If a start time is provided for the schedule, it is used for determining the start of the schedule run. If no job-schedule start time is defined, the start time for the job is used. The date and time-formats must be specified as strings.
`endTime`	Object	The end time for the job. If the end time is specified for a job, the scheduler checks if an end time is provided for the schedule as well. If an end time is provided for the schedule, it is used for determining the end of the schedule run If not, the end time for the job is used. The date and time-formats must be specified as strings.
`schedules` *	Array	The array of job schedule objects, to be created on job creation.

The `schedules` parameter can be used to provide details of the job schedule (as properties of each job schedule object); the following table lists the permitted properties:

Schedule Parameter Fields

Schedule Field	Type	Description
`data`	object	Optional data to be passed to the job action endpoint when invoked. Typically, the custom data is sent based on the HTTP method configured for invoking the end point URL,for example: `{"dataParam": "somevalue"}`
`time`	string or object	For one-time schedules, the parameter denoting the time at which the task executes. A human-readable text can be used to specify the time, for example, "`3.30pm`" or "`tomorrow at 2am`". If an object is used, the date and time-formats must be specified as strings.
`repeatInterval`	string	For recurring schedules, the parameter denoting the intervals when the schedule should run. The parameter supports the use of human readable formats.
`repeatAt`	string	For recurring schedules, the parameter denoting the exact time when the job schedule must run. A human-readable text can be used to denote a specified time, for example, "`3.30pm`" or "`tomorrow at 2am`", if the schedule runs repeatedly.
`cron`	string	For recurring schedules, the parameter denoting the `cron` pattern. It must be a valid `crontab` format, for example: `"* * * * * * */10"`
`startTime`	object	The time when the job scheduling should start. The date and time-formats must be specified as strings.
`endTime`	object	The time when the job scheduling should end. The date and time-formats must be specified as strings
`description`	string	The user-provided description of the job schedule

Job Configuration

Configure a job with updated run time information. The API can also be used to create a job if a Job with the Job Name in the URI segment, is not found. If the API is being used to create a job, the parameters must conform to the same constraints as provided in the Job Creation API

- **Route**
  `PUT /scheduler/jobs/:jobId`
  `PUT /scheduler/jobs/:jobName`
  "`:jobId`" is the ID of the job previously created using the Job Creation API. If the job name is used in the URI, it is first checked if the job with the name, exists. If no such named job exists, the API tries to create the job. If it does exist, the API configures the job with the details provided in the request body.

> **Note**
>
> If the API is used to create a job, care must be taken to ensure that the job name in the request URI matches the name of the job in the request body. If the names do not match, an error is returned.

- **Response**
  If the API finds an existing job, the response has a status code of "`200-OK`", if the call was successful. The response has a status code of "`201-CREATED`", if the API is used to create a new job; for new jobs, a location header containing the relative path to the job-details is returned in the response.

Sample Code

```
PUT /schedule/jobs/5 HTTP/1.1
content-type:application/json;charset=utf-8
host:https://scheduler.service.acme.com
content-length: 500
{"active": true, "user":"abc", "password":"def", "httpMethod": "GET"}
```

```
Response:
status: 200 OK
content-type: application/json; charset=utf-8
{"success": true}
```

Sample Code

```
PUT /schedule/jobs/jobwhichdoesnotexist HTTP/1.1
content-type:application/json;charset=utf-8
host:https://scheduler.service.acme.com
content-length: 500
{"name":"jobwhichdoesnotexist", "jobDescription": "greets
the world periodically", "action":"http://httpbin.org/basic-auth/abc/
def","active":true, "httpMethod":"GET", "schedules": [{"repeatEvery":"2
minutes", "scheduleDescription": "every 2 minutes, run this schedule", "data":
{"time":"abc"}, "active": true}, {"cron":"* * * * *", "scheduleDescription":
"every 4 minutes, run this schedule", "data":{"time":"abc"}, "active":
false}]}
```

```
Response:
status: 201 CREATED
content-type: application/json; charset=utf-8
{"_id":120,"name":"jobwhichdoesnotexist","description":"","action":"http://
httpbin.org/basic-auth/abc/
def","active":true,"user":null,"httpMethod":"GET","schedules":
[{"scheduleId":"b373469c-c6d4-4d5f-a002-c56f18455dc5","description":"Default
Schedule","data":
{"time":"abc"},"type":"recurring","active":true,"startTime":null,"endTime":nul
l,"repeatInterval":"2 minutes"},{"scheduleId":"2f98471c-26de-4293-ae53-
e4a16e1513f5","description":"Default Schedule","data":
{"time":"abc"},"type":"recurring","active":false,"startTime":null,"endTime":nu
ll,"cron":"* * * * *"}]}
```

The job schedule configuration request is defined with the parameters listed in the following table:

Job Creation: Request Body Fields

Request Field	Type	Description
`active`	Boolean	Defines if the job should be activated on configuration. Allowed values are: • `false` (default) The job is in inactive mode when configured • `true` The job is active when configured
`user`	String	The name of the user account to run the configured job
`password`	String	The password for the user account to run the configured job
`httpMethod`	String	The HTTP method to be used to call the end-point URL for the job action . Allowed values are: `GET`, `POST` (default), `PUT`, and `DELETE`
`startTime`	Object	The start time for the job. If the start time is specified for the job, the scheduler checks if a start time is provided for the schedule as well. If a start time is provided for the schedule, it is used for determining the start of the schedule run. If no job-schedule start time is defined, the start time for the job is used. The date and time-formats must be specified as strings.
`endTime`	Object	The end time for the job. If the end time is specified for a job, the scheduler checks if an end time is provided for the schedule as well. If an end time is provided for the schedule, it is used for determining the end of the schedule run If not, the end time for the job is used. The date and time-formats must be specified as strings.

Job Deletion

Delete a job and purge all its run time information such as job schedules and logs.

- **Route**
  `DELETE /scheduler/jobs/:jobId`
- **Response**
  If the call is successful, the response has a status code "`200-OK`" and includes a JSON response `{"success": true}`.

Sample Code

```
DELETE /schedule/jobs/:jobId HTTP/1.1
content-type:application/json;charset=utf-8
host:https://scheduler.service.acme.com
```

```
Response: Status: 200 OK
Content-Type: application/json;charset=utf-8
{"success":true}
```

Job Schedule Creation

Create a job schedule for a specified job. All job configuration values (Action URL, HTTP Method, User, Password & Job Activation Status) are valid for the newly created schedule. A job schedule will only run if both the job and the schedule are active.

- **Route**
  `POST /scheduler/jobs/:jobId/schedules`
- **Response**
  If the call is successful, the response has a status code of "`201-CREATED`". A location header with the relative path to the schedule-details, is returned in the response.

Sample Code

```
PUT /scheduler/jobs/3/schedules
{
    "repeatInterval":"2 hours",
    "active":true,
    "description":"New Schedule",
    "startTime": {
        "date": "2017-08-21",
        "format": "YYYY-MM-DD"
    }
}
```

```
Response:
{
  "repeatInterval": "2 hours",
  "repeatAt": null,
  "time": null,
  "cron": null,
  "data": "{\"order_id\":\"abcd\"}",
  "description": "New Schedule",
  "type": "recurring",
  "active": true,
  "startTime": "2015-04-20 18:30:00",
  "endTime": null,
  "jobId": 3,
  "scheduleId": "<schedule ID details>"
}
```

Job Schedule Creation Parameters

Request Field	Type	Description
`time`	string or object	For one-time schedules, the parameter denoting the time at which the task executes. A human-readable text can be used to specify the time, for example, "`3.30pm`" or "`tomorrow at 2am`". If an object is used, the date and time-formats must be specified as strings.
`repeatInterval`	string	For recurring schedules, the parameter denoting the interval when the schedule should run. The parameter supports the use of human readable formats.

Request Field	Type	Description
`repeatAt`	string	For recurring schedules, the parameter denoting the exact time when the job schedule must run. A human-readable text can be used to denote a specified time, for example, "`3.30pm`" or "`tomorrow at 2am`", if the schedule runs repeatedly.
`cron`	string	For recurring schedules, the parameter denoting the `cron` pattern. It must be a valid `crontab` format, for example: `"* * * * * * */10"`
`data`	object	The parameter denoting optional data to be passed to the job action endpoint when invoked. Typically, the custom data is sent based on the HTTP method configured for invoking the end point URL,for example: `{"dataParam": "somevalue"}`
`startTime`	object	The time when the job scheduling should start. The date and time-formats must be specified as strings.
`endTime`	object	The time when the job scheduling should end. The date and time-formats must be specified as strings
`active`	Boolean	Defines if the job should be activated on configuration. Allowed values are: • `false` (default) The job is in inactive mode when configured • `true` The job is active when configured
`description`	string	The user-provided description of the job schedule

Job Schedule Modification

Configure the run time information of a job schedule for a specified job. All job configuration values (for example: `Action URL`, `HTTP Method`, `User`, `Password`, and `Job Activation Status`) remain valid for the modified schedule.

- **Route**
  `PUT /scheduler/jobs/:jobId/schedules/:scheduleId`
- **Response**
  If the call is successful, the response has a status code of `200- OK`.

Calling this API stops further scheduling of the previously configured job schedule and, if activated, the processing for the newly configured schedule is started. This API cannot be used to change the scheduling mode for the job schedule. For example, if the schedule was created as a recurring "cron"-type schedule, it cannot be changed to a "repeatEvery"-type schedule. However, existing schedule values can be changed.

Sample Code

```
PUT /schedule/jobs/:jobId/schedules/:scheduleId HTTP/1.1
```

```
content-type:application/json;charset=utf-8
host:https://scheduler.service.acme.com
content-length: 500
{"description": "Edited Schedule", "startTime": {"date": "2013-02-08
09:30:26.123"}, "endTime": {"date": "2015-06-08 09:30:26.123"}, "active":
true, "cron": "* * * * *" }
```

```
Response:
Status: 200 OK
Content-Type: application/json; charset=utf-8
{"scheduleId":"80e23846-734e-4b4b-a130-159a492ec482","name":"greet the
world3","data":
{"time":"abc"},"type":"recurring","priority":0,"action":"http://httpbin.org/
basic-auth/abc/
def","nextRunAt":"2015-04-23T03:58:21.358Z","startTime":"2013-02-08T04:00:26.1
23Z","endTime":"2015-06-08T04:00:26.123Z","active":true,"description":"Edited
Schedule","jobId":"136","cron":"* * * * *"}
```

Job Schedule Modification Parameters

Request Field	Type	Description
`time`	string or object	For one-time schedules, the parameter denoting the time at which the task executes. A human-readable text can be used to specify the time, for example, "`3.30pm`" or "`tomorrow at 2am`". If an object is used, the date and time-formats must be specified as strings.
`repeatInterval`	string	For recurring schedules, the parameter denoting the interval when the schedule should run. The parameter supports the use of human readable formats.
`repeatAt`	string	For recurring schedules, the parameter denoting the exact time when the job schedule must run. A human-readable text can be used to denote a specified time, for example, "`3.30pm`" or "`tomorrow at 2am`", if the schedule runs repeatedly.
`cron`	string	For recurring schedules, the parameter denoting the `cron` pattern. It must be a valid `crontab` format, for example: `"* * * * * * */10"`
`data`	object	The parameter denoting optional data to be passed to the job action endpoint when invoked. Typically, the custom data is sent based on the HTTP method configured for invoking the end point URL,for example: `{"dataParam": "somevalue"}`
`startTime`	object	The time when the job scheduling should start. The date and time-formats must be specified as strings.
`endTime`	object	The time when the job scheduling should end. The date and time-formats must be specified as strings

Request Field	Type	Description
`active`	Boolean	Defines if the job should be activated on configuration. Allowed values are: • `false` (default) The job is in inactive mode when configured • `true` The job is active when configured
`description`	string	The user-provided description of the job schedule

Job Schedule Deletion

Delete and purge run time information of the job schedule of the specified job. All related information like job schedule configurations and logs are purged. The processing of the schedule is also immediately stopped.

> ⚠ Caution
>
> This API removes all the run time configuration information of the job schedule, irrespective of whether the schedule is active or not.

- **Route**
  `DELETE /scheduler/jobs/:jobId/schedules/:scheduleId`
- **Response**
  If the call is successful, the response has a status code, "`200-OK`" and includes a JSON response `{"success": true}`.

> Sample Code

```
DELETE /scheduler/jobs/:jobId/schedules/:scheduleId HTTP/1.1
content-type:application/json;charset=utf-8
host:https://scheduler.service.acme.com
```

```
Response: {"success": true}
Status Code: 200 OK
```

Bulk Job Schedule Activation/Deactivation

This is a utility API used to activate or deactivate all existing schedules of a job.

- **Route**
  `POST /scheduler/jobs/:jobId/schedules/activationStatus`
- **Response**
  If the call is successful, the response has a status code, "`200-OK`" and includes a JSON response `{"success": true}`.

Sample Code

```
POST /scheduler/jobs/:jobId/schedules/activationStatus HTTP/1.1
content-type:application/json;charset=utf-8
host:https://scheduler.service.acme.com
{"activationStatus": true}
```

```
Response: {"success": true}
Status Code: 200 OK
```

Bulk Job Schedule Activation Parameters

Request Field	Type	Description
`activationStatus`	Boolean	The desired activation status of the job schedules for the job. Allowed values for the activation status are: • `false` (default) All job schedules for the specified job should be **de**activated • `true` All job schedules for the specified job should be activated

Job Details

Retrieve the saved details and configurations of a specified job.

- **Route**
  `GET /scheduler/jobs/:jobId?displaySchedules=true`
  `GET /scheduler/jobs?jobId=:jobId&displaySchedules=true Route`
  `GET /scheduler/jobs?name=:jobName&displaySchedules=true`
- **Response**
  If the call is successful, the response has a status code, "`200-OK`"
  and includes a JSON response with the schedule details, for example:
  `{"schedules":[{"data":{"time":"abc"},"type":"recurring","repeatInterval":"2 minutes","active":false,"startTime":null,"endTime":null,"repeatAt":null, [...]}`.

Sample Code

```
GET /scheduler/jobs/:jobId?displaySchedules=true HTTP/1.1
content-type:application/json;charset=utf-8
host:https://scheduler.service.acme.com
```

```
Response:
Status: 200 OK
Content-type: application/json;charset=utf-8
{"schedules":[{"data":{"time":"abc"},"type":"recurring","repeatInterval":"2
minutes","active":false,"startTime":null,"endTime":null,"repeatAt":null,"sched
uleId":"0d3b4cc1-0f7b-4ee6-ab12-63d474b900f2","description":"Default
Schedule"},{"data":{"time":"abc"},"type":"recurring","cron":"* * * *
*","active":false,"startTime":null,"endTime":null,"repeatAt":null,"scheduleId"
:"1b1bb70f-cada-46c9-9974-a7a1b87ba24f","description":"Default
Schedule"}],"name":"greet the world2","description":"","action":"http://
```

```
httpbin.org/basic-auth/abc/
def","user":null,"httpMethod":"GET","active":false,"_id":111}
```

Job Details Parameters

Request Field	Type	Description
`displaySchedules`	Boolean	Display details of the job schedules for the job. Allowed values for the job details are: • `false` Do **not** display details of job schedules for the specified job • `true` Display details of job schedules for the specified job
`jobId`	String	The job ID needed to query for the job details. This can be passed as a URI segment parameter or as a query parameter.
`name`	String	The job name needed to query the job details. This can be passed as a query parameter

Job Schedule Details

Retrieve the saved details and configurations of a specified job schedule & optionally the generated logs for the schedule. Either `:jobId` or `:name` is required to invoke this API. If `displayLogs` is not provided, the logs for the schedule are not returned and only the schedule details are returned.

- **Route**
  `GET /scheduler/jobs/:jobId/schedules/:scheduleId?displayLogs=true`
- **Response**
  If the call is successful, the response has a status code, "`200-OK`" and includes a JSON response with the schedule details, for example: `{"data":{"time":"abc"},"type":"recurring","repeatInterval":"2 minutes","plannedTime":"2015-04-19T15:12:44.000Z","active":true,"startTime":null,"endTime":null,"nextRunAt": "2017-08-11 10:00:00","repeatAt":null, [...]}`.

Sample Code

```
GET /scheduler/jobs/112/schedules/550d1b96-8002-4d0d-850e-368aaa591671?
displayLogs=true
HTTP/1.1 content-type:application/json;charset=utf-8
host:https://scheduler.service.acme.com
```

```
Response:
Status: 200 OK
Content-Type: application/json; charset=utf-8
{"data":{"time":"abc"},"type":"recurring","repeatInterval":"2
minutes","plannedTime":"2015-04-19T15:12:44.000Z","active":true,"startTime":nu
ll,"endTime":null,"nextRunAt": "2017-08-11 10:00:00","repeatAt":null,"logs":
[{"text":null,"httpStatus":null,"executionTime":null,"status":"SCHEDULED","sch
eduleTime":"2015-04-19T15:10:53.000Z","completionTime":null}],"scheduleId":"55
0d1b96-8002-4d0d-850e-368aaa591671","description":"Default Schedule"}
```

Job Schedule Details Parameters

Request Parameter	Type	Description
`displayLogs`	Boolean	Controls whether the API should return (`true`) all the generated logs for the job schedule or not (`false`)

Bulk Job Schedule Deletion

Delete and purge run time information of all the currently configured job schedules of the specified job. All related information like job schedule configurations and logs are purged. The processing of the schedules is also immediately stopped.

> ⚠ Caution
>
> This API removes all the run time configuration information of the job schedule, irrespective of whether the schedule is active or not.

- **Route**
  `DELETE /scheduler/jobs/:jobId/schedules`
- **Response**
  If the call is successful, the response has a status code, "`200-OK`" and includes a JSON response `{"success": true}`.

> Sample Code

```
DELETE /scheduler/jobs/:jobId/schedules HTTP/1.1
content-type:application/json;charset=utf-8
host:https://scheduler.service.acme.com
```

```
Response: {"success": true}
Status Code: 200 OK
```

Job Run Log Update

Inform the Job Scheduler about the status of an asynchronous, long-running job run. This API must be invoked by the application after the asynchronous execution of the job has completed, with the status of the job run and optionally some text about the job execution.

> ⚠ Caution
>
> This API must be invoked by the application after the **asynchronous** execution of the job has completed, with the status of the job run and optionally some text about the job execution.

- **Route**
  `PUT /scheduler/jobs/:jobId/schedules/:scheduleId/runs/:runId`

> **Note**
>
> Parameters marked with an asterisk (*) are mandatory.

Job Run Log Update Parameters

Request Parameter	Type	Description
`success` *	Boolean	Indicates that the job run was successful (`true`) or failed (`false`)
`message`	String	Additional log/text about the job run

Human Readable Dates

The job scheduler for XS advanced supports human readable dates and ranges for the parameters `time`, `repeatAt` and `repeatEvery`, which are used for configuring job schedules. The job scheduler uses an embedded English language date parser for this facility. Valid human readable strings for the parameters are shown below:

> **Note**
>
> The date parser expects a valid readable string; invalid strings will either throw parser errors or cause the job scheduling to happen inconsistently.

Date and Time Parameters

Parameter	Comments	Examples
`time`	Designates a particular timestamp for running a job schedule. If an invalid string is provided, the scheduler falls back to the current timestamp and runs the schedule immediately. The following example strings are valid for the `time` parameter:	“10 hours from now” “20 minutes from now” “in 2 hours” “tomorrow at 4pm” “next week monday at 5am” “9pm tonight” “3.30pm”

Parameter	Comments	Examples
`repeatAt`	Represents a convenient way to create daily timestamp-based schedules. The string should designate a particular timestamp for repeatedly running a job schedule. This follows the same pattern as the recommendations for the "time" parameter, barring a few discrepancies. While the text for the "time" parameter must denote something concrete and in the future, the 'repeatAt' must designate a timestamp, which is valid and constant daily. If an invalid string is used, the scheduler falls back to the current timestamp and runs the schedule immediately. **i Note** Second-based precision can sometimes be inaccurately timed; timezones must be specified using the offset (in hours), for example, “+07:00”	“4.40pm” “18.40” “6.20am” “17.20:30 ”
`repeatInterval`	The string should designate a interval to repeat the job execution. Word strings for denoting the numeric value are not supported yet. For example, for “twenty minutes”, use “20 minutes” to denote the interval. Supported time-units for this parameter are "years", "months", "weeks", "days", "hours", "minutes", "seconds".	“10 hours ” “2 days ” “3 seconds”

Date and Time Formats in Job Schedule Parameters

The date-time parameters for job schedules (for example, `startTime`, `endTime`, and `time`) can be passed as objects , with the mandatory `date` field denoting the date as a string and an optional `format` field denoting a date-time format for correctly parsing the user-provided date value. If the parameters are passed as strings, they must be valid date representations, in either the ISO-8601 or IETF-compliant RFC 2822 formats. For object representations, the following rules apply:

- **Date field as input**
  If only the date field is provided as input, the string is checked against both IETF-compliant RFC 2822 time stamps and ISO-8601. If the date string is of an unknown format, the parser displays an error. For ISO-8601 compliant dates, calendar dates (for example, “2013-02-08”), week dates (“2013-W06-5”), ordinal dates (“2013-039”) and time-based dates (“2013-02-08 09+07:00”) are all supported.
- **Date string format**
  If the format of the date string is customized, an optional format string can be passed. The allowed parsing tokens are as described in the following table:

Date and Time Parameters

Input Token	Example	Description
YYYY	2014	4 digit year

Input Token	Example	Description
YY	14	2 digit year
Q	1-4	Quarter of year. Sets month to first month in quarter
M MM	1-12	Month number
MMM MMMM	January- Dec	Month name in locale
D DD 1- 31		Day of month
Do	1st- 31st	Day of month with ordinal
DDD DDDD	1-365	Day of year
X	1410715640.579	Unix Timestamp
x	1410715640579	Unix Timestamp (ms)
gggg	2015	Locale 4 digit week year
gg	15	Locale 2 digit week year
w ww	1- 53	Locale week of year
e	1-7	Locale day of week
GGGG	2015	ISO 4-digit week year
GG	15	ISO 2-digit week year
W WW	1- 53	ISO week of year
E	1-7	ISO day of week
H HH	0 -23	24 Hour Time
h hh	1 -12	12 hour time used with 'a A'
a A	am pm	Post or ante meridiem
m mm	0 -59	Minutes
s ss	0 -59	Seconds
S	0 -9	Tenths of a second
SS	0 -99	Hundredths of a second
SSS	0 -999	Thousandths of a second
Z ZZ	+12:00	Offset from UTC as +-HH:mm, +-HHmm, or Z

Date-Time Format Examples

- `startTime`
  `"startTime": {"date": "2015-10-20 4:30 +0000", "format": "YYYY-MM-DD HH:mm Z"}`
  4.30 UTC on 20th Oct 2015
- `endTime`
  `"endTime": {"date": "2015-W06-5"}`
  Friday, February 06, 2015
- `time`
  `"time": {"date": "2010-10-20 4:30", "format": "YYYY-MM-DD HH:mm"}`
  4.30 Local Time (the timezone for the scheduler service is considered here)

Related Information

Scheduling Jobs in XS Advanced [page 881]

8.9.4.2 The Job Scheduler Dashboard

The Job Scheduler dashboard enables you to manage job schedules for a service instance.

The dashboard lists the available jobs. Select a job to create a schedule or to view existing schedules.

How to access the Job Scheduler dashboard

1. **Get the Dashboard URL**
   To access the Job Scheduler dashboard, you can connect remotely using secure shell (SSH) to the SAP XS Advanced server and perform the following steps:
   1. List the applications running on the server using the command `xs apps`.
   2. In the *URL* column, identify the relevant URL for the application (“jobscheduler-dashboard” for Job Scheduler).
   3. Copy the URL to any Web browser to launch the application.

      > **i** Note
      >
      > If a valid certificate is not available, the Web browser indicates an issue with the certificate. To resolve the issue, add the required certificate.

   4. Enter the logon credentials to access the application.
2. **Permission to Access the Dashboard**
   The administrator role with the **XS_CONTROLLER_ADMIN** role template contains the **controller.admin** permission. This role template gives you full access to perform all the administration-related tasks. If you do not have the administrator role, you need the roles and role collection templates listed in the table below to view or modify settings.

Task	Roles Required	Role Collection Template Required
Open the dashboard and only view the job listing page **i Note** You cannot create jobs.	Org Manager Space Developer	jobscheduler_viewer_template
Configure settings	Org Manager Space Developer	jobscheduler_config_template
Create or edit jobs or schedules	Org Manager Space Developer	jobscheduler_admin_template

i Note

If you already have the permission to create or edit jobs or schedules, you still need **jobscheduler_config_template** to configure settings.

The SAP HANA administrator creates a role collection and adds the Job Scheduler roles to the role collection. For more information, see *Maintaining the SAP HANA XS Advanced Model Run Time* in the *SAP HANA Administration Guide*.

Dashboard Screens

The following table contains the various screens available on the dashboard and their descriptions:

Screen	Description
Configuration	Enables you to maintain the global configuration required for a specific Job Scheduler service instance. • *Max. Invocation Attempts:* The number of attempts made by the Job Scheduler to reach the job action endpoint before it deactivates the job. If the Job Scheduler fails repeatedly to reach the endpoint, it sets the job to inactive. The default value of this parameter is 3. • *Asynchronous Execution timeout (ms):* The duration (in milliseconds) that the Job Scheduler waits for a response for the asynchronous job from the application endpoint. If the application does not provide a response in the specified duration, the run status is set to COMPLETED/UNKNOWN.

Screen	Description
Jobs	Lists all the jobs created for a specific service instance. You can delete a job or navigate to the job details by choosing the job name.
Overview	Displays the details of the selected job. You can edit a job.
Schedules	Enables you to create and configure schedules for a job. To access schedules, select a job listed on the dashboard. The *Schedules* screen is displayed. Select a schedule to see the history and logs corresponding to the schedule. To display the run logs of a schedule, choose *Logs*.
Action History	Maintains the history of a job or schedule for a specific job.

Related Information

Roles for Running the Job Scheduler [page 883]
Scheduling Jobs in XS Advanced

9 Creating the Client User Interface

Create the user interface (GUI) to display the database content (HTML pages).

The application module for the client user interface (`web/` in the example below) also contains the application-route description (`xs-app.json`), which is used by the application router module. You set up the application routes description in a subsequent step.

Sample Code

```
JavaScript-AppName
|- db/
|  |- package.json
|  |- src/
|     |- .hdiconfig
|     \- mytable.hdbdd
|- web/                          # Application descriptors
|  |- xs-app.json                # Application routes configuration
|  |- package.json               # Application router details/dependencies
|  \- resources/                 # Static resources
|- js/
|  |- start.js
|  |- package.json
|  \- src/
|     \- odata/
|        |- resources/
|        \- services/
|           | - srv1.xsodata
|           \ - srvN.xsodata
|- security/
|  \- xs-security.json
\- mtad.yaml
```

9.1 Building User Interfaces with SAPUI5 for SAP HANA

UI development toolkit for HTML5 (SAPUI5) is a user-interface technology that is used to build and adapt client applications based on SAP HANA. You can install SAPUI5 and use it to build user interfaces delivered by SAP HANA's Web server.

The SAPUI5 run time is a client-side HTML5 rendering library with a rich set of standard and extension controls. The SAPUI5 run time provides a lightweight programming model for desktop as well as mobile applications. Based on JavaScript, it supports Rich Internet Applications (RIA) such as client-side features. SAPUI5 complies with OpenAjax and can be used with standard JavaScript libraries.

SAPUI5 Demo Kit

The SAPUI5 Demo Kit contains the following components:

- *Documentation*
  Information about the programming languages used, open source technology, development tools, and API usage. You can also find tutorials to help you get started and details about the new features delivered with each version of SAPUI5.
- *API reference*
  The complete JavaScript documentation for the Framework and Control API, including featured namespaces such as `sap.m` (main controls), `sap.ui.layout` (layout controls), `sap.ui.table` (table controls), `sap.f` (SAP Fiori), and `sap.ui.core` (UI5 core run time).
- *Samples*
  A detailed view of almost every control available in the kit, including detailed information about featured controls such as: user interaction elements, lists, tables, pop-up dialogs tiles, messages, maps and charts, smart controls, step-based interaction, and so on. You can also find detailed information about object pages, dynamic pages, and how to use flexible columns.
- *Demo Apps*
  A selection of apps for download that are designed to showcase UI5 concepts and controls in real-life scenarios
- *Tools*
  SAPUI5 comes with a built-in set of icons for use in your applications. The *Icon Explorer* helps you to find icons by allowing you to browse through categories or simply start a search. You can see a live preview of how the icon looks when being applied to a SAPUI5 control. You can also add icons to your favorites list to make them easier to find at a later stage.
  In addition, you can download development tools such as SAP Web IDE (and the SDK) as well as the *UI5 Inspector* which you can use for debugging and supporting both OpenUI5 and SAPUI5 applications.

Related Information

SAPUI5 Demo Kit

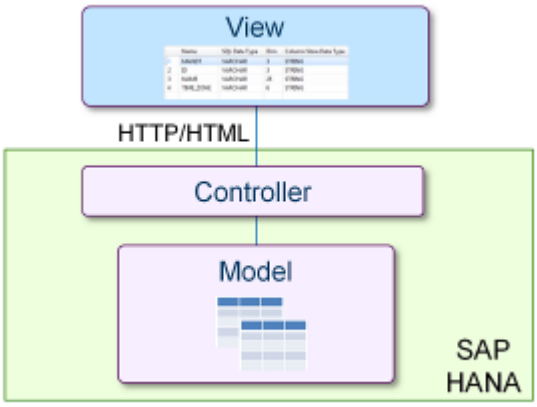
9.2 Consuming Data and Services with SAPUI5 for SAP HANA

SAP HANA Extended Application Services (SAP HANA XS) can be used to expose the database data model, with its tables, views and database procedures, to UI clients.

You can expose an SAP HANA model using OData services or by writing native server-side JavaScript code that runs in the SAP HANA context. You can also use SAP HANA XS to build dynamic HTML5 client applications, for example, using SAPUI5 for SAP HANA.

The server-centric approach to native application development envisaged for SAP HANA assumes the following high-level scenario:

- View
  UI rendering occurs completely in the client (SAPUI5, browser, mobile applications)
- Controller
  Procedural (control-flow) logic is defined in (XS) JavaScript, SQLScript or an OData service
- Model
  All application artifacts are stored in SAP HANA



SAP HANA Application Development with SAP HANA XS

Each of the levels illustrated in the graphic (view, control, model) is manifested in a particular technology and dedicated languages. After you have defined the data model with design-time artifacts and the equivalent run-time objects, you develop the control-flow logic to expose the data, for example, using server-side JavaScript or an OData service. With the data model and control-flow logic in place, you can build the presentation logic to view the exposed data in a UI client application using SAPUI5 for SAP HANA. For example, you can use an SAPUI5 client to request and display data exposed by an OData service; the UI could include buttons that trigger operations performed by SAP HANA XS JavaScript service; and the data displayed is retrieved from data end points defined in your data model, for example, SQLScript or CDS.

Related Information

SAPUI5 Demo Kit (version 1.32.7)

10 Maintaining XS Advanced Application Routes and Destinations

Define application routes and destinations.

In XS Advanced Programming Model, the application router (“approuter”) is the entry point for a business application and is used for routing incoming requests to the appropriate microservices. The application routes are defined in the application descriptor `xs-app.json`, a JSON-formatted file that is mandatory for the approuter. In the following example, the approuter is located in the `web/` directory, so the `xs-app.json` application descriptor must be placed there, too.

Sample Code

Application Routes Configuration

```
JavaScript-AppName
|- db/
|   |- package.json
|   |- src/
|      |- .hdiconfig
|      \- mytable.hdbdd
|- web/                          # Application descriptors
|   |- xs-app.json               # Application routes configuration
|   |- package.json              # Application router details/dependencies
|   \- resources/
|- js/
|   |- start.js
|   |- package.json
|   \- src/
|      \- odata/
|         |- resources/
|         \- services/
|            | - srv1.xsodata
|            \ - srvN.xsodata
|- security/
|   \- xs-security.json
|- manifest.yml
\- mtad.yaml
```

Related Information

Configure the XS Advanced Application Router [page 913]
XS Advanced Application-Router Resource Files [page 916]

10.1 Configure the XS Advanced Application Router

The application routes to the available microservices are described in the `xs-app.json` file.

Prerequisites

- File name
  - `xs-app.json`
    Configures the application router and describes the routes.

    > **i** Note
    >
    > Without the application router configuration, no access to the dependent application is possible.

  - `package.json`
    Contains the start command for the application router and a list of package dependencies.
- Location
  Application's Web resources folder, for example, `/path/appname/web`
- Format
  JavaScript Object Notation (JSON)

Context

The application router is used to serve static content, authenticate users, rewrite URLs, and forward or proxy requests to other micro services while propagating user information. The application-router configuration file `xs-app.json` configures the application router and describes the routes. To configure the application router, perform the following steps:

Procedure

1. Create the application resource-file structure.

   For example, `/path/appname/`
2. Create a sub-folder for the static Web-resources module.

   The sub-folder for the Web-resources module must be located in the application root folder, for example, `/path/<myAppName>/web`

   > → Tip
   >
   > The Web-resource module uses `@sap/approuter` as a dependency; the Web-resources module also contains the configuration and static resources for the application.

3. Create sub-folder for the application's static resources.

   Static resources can include the following file and components: `index.html`, style sheets (`.css` files), images and icons, etc.). Typically, static resouces for a Web application are placed in a subfolder of the Web module, for example, `/path/<myAppName>/web/resources`.

4. Create the application-router configuration file.

   The application router configuration file `xs-app.json` must be located in the application's Web-resources folder, for example, `/path/<appname>/web`.

   Sample Code

   ```
   <myAppName>
   |- web/                          # Application descriptors
   |  |- xs-app.json                # Application routes configuration
   |  |- package.json               # Application router details/dependencies
   |  \- resources/
   ```

5. Define details of the application's routes, destinations, and security scopes.

   The contents of the `xs-app.json` file must use the required JSON syntax. For more information, see *XS Advanced Application Router Configuration Syntax* in *Related Information* below.

   a. Create the required “destinations configuration”.

      Sample Code

      `<myAppName>/web/xs-app.json`

      ```
      {
        "welcomeFile": "index.html",
        "routes": [
          {
            "source": "/sap/ui5/1(.*)",
            "target": "$1",
            "localDir": "sapui5"
          },
          {
            "source": "/rest/addressbook/testdataDestructor",
            "destination": "backend",
            "scope": "node-hello-world.Delete"
          },
          {
            "source": "/rest/.*",
            "destination": "backend"
          },
          {
            "source": "^/(.*)",
            "localDir": "resources"
          }
        ]
      }
      ```

   b. Add the routes (destinations) for the specific application (for example, `node-hello-world`) to the `env` section of the application’s deployment manifest (`manifest.yml`).

      Every route configuration that forwards requests to a micro service has property destination. The destination is a name that refers to the same name in the destinations configuration. The destinations configuration is specified in an environment variable passed to the approuter application.

Sample Code

<myAppName>/manifest.yml

```
 - name: node-hello-world
   host: myHost-node-hello-world
   domain: xsapps.acme.ondemand.com
   memory: 100M
   path: web
   env:
     destinations: >
      [
        {
          "name":"backend",
          "url":"https://myHost-node-hello-world-
backend.ondemand.acme.com",
          "forwardAuthToken": true
        }
      ]
```

6. Add a package descriptor for the XS advanced application router.

   The package descriptor (`package.json`) describes the prerequisites and dependencies for the application and starts the application router, too. Add a `package.json` file to the root folder of your application's Web resources module (`web/`).

Sample Code

```
<myAppName>
|- web/                          # Application descriptors
|  |- xs-app.json                # Application routes configuration
|  |- package.json               # Application router details/dependencies
|  \- resources/
```

The basic `package.json` file for your Web-resources module (`web/`) should look similar to the following example:

Sample Code

```
{
  "name": "node-hello-world-approuter",
  "dependencies": {
     "@sap/approuter": "^2.7.1"
  },
  "scripts": {
     "start": "node node_modules/@sap/approuter/approuter.js"
  }
}
```

→ Tip

The start script (for example, `approuter.js`) is mandatory; the start script is executed after application deployment.

Related Information

XS Advanced Application Routes and Destinations [page 936]
XS Advanced Application-Router Resource Files [page 916]
The XS Advanced Application Descriptor [page 917]
XS Advanced Application Router Configuration Syntax [page 922]

10.1.1 XS Advanced Application-Router Resource Files

The routing configuration for an application is defined in one or more so-called "destinations" that are defined in a destinations configuration.

The application router is used to serve static content, authenticate users, rewrite URLs, and forward (or proxy) requests to other micro services while propagating user information. The following table lists the configuration files used to define routes for XS advanced, multi-target applications:

XS Advanced Application-Router Resource Files Overview

File	Description	Mandatory
`package.json`	The package descriptor is used by the node.js package manager (`npm`) to start the application router; in the `"dependencies": {}` section	Yes
`xs-app.json`	The application descriptor contains the configuration used by the application router (for example, destinations for request forwarding)	Yes
`resources/`	A folder that contains all static resources which should be served by the application router. If no `resources/` folder is present, the application router will not serve any static content. However, it still forwards requests to the configured destinations. → Tip Static resources can be stored in multiple folders, and the folder name "`resources/`" is optional.	No
`default-services.json`	Defines the configuration for one or more special User Account and Authentication (UAA) services for local development; this is typically used for testing in local environments.	-

Related Information

Configure the XS Advanced Application Router [page 913]

10.1.2 The XS Advanced Application Descriptor

Understand the contents of the file used to configure the XS advanced application router.

When a business application consists of several different apps (microservices), the application router is used to provide a single entry point to that business application. The application router is responsible for the following tasks:

- Dispatch requests to back-end microservices (reverse proxy)
- Authenticate users
- Serve static content
- In multi-tenancy scenarios, derive the tenant information from the URL and forward the information to the XS UAA service so that the authentication request is redirected to the appropriate tenant-specific Identity Provider (IdP), for example,using SAML "Bearer Assertions".

The application descriptor is a file that contains the configuration information used by the application router. The file is named `xs-app.json` and its content is formatted according to JavaScript Object Notation (JSON) rules.

> → Tip
>
> The different applications (microservices) are the destinations to which the incoming requests are forwarded. The rules that determine which request should be forwarded to which destination are called routes. For every destination there can be more than one route.

User Authentication Services

User authentication is performed by the User Account and Authentication (UAA) server. In the XS advanced run-time environment (on-premise and in Cloud Foundry), a service is created for the UAA configuration; by using the standard service-binding mechanism, the content of this configuration is available in the `<VCAP_SERVICES>` environment variable, which the application router can access to read the configuration details.

> ℹ Note
>
> The UAA service should have `xsuaa` in its tags or the environment variable `<UAA_SERVICE_NAME>` should be defined, for example, to specify the **exact** name of the UAA service to use.

A calling component accesses a target service by means of the application router only if there is no JSON Web Token (JWT) token available, for example, if a user invokes the application from a Web browser. If a JWT token is already available, for example, because the user has already been authenticated, or the calling component uses a JWT token for its own OAuth client, the calling component calls the target service directly; it does not need to use the application router.

> ℹ Note
>
> The application router does not "hide" the back-end microservices in any way; they remain directly accessible when bypassing the application router. So the back-end microservices must protect all their end points by validating the JWT token and implementing proper authorization scope checks.

The application router supports the use of the `$XSAPPNAME` placeholder, which you can use in your route configuration, for example, in the `scope` property for the specified route. The value of `$XSAPPNAME` is taken from the UAA configuration (for example, the `xsappname` property). For more information, see *XS Advanced Application Router Configuration Syntax* in *Related Information*.

Headers

In many cases back-end nodes need to respond to client requests with URLs. Since micro services are accessed through the application router component, the clients always request the application router's URL. The back end nodes need to know the actual request URL in order can build URLs that are accessible by the client. If back-end nodes respond to client requests with URLs, these URLs need to be accessible to the client. For this reason, the application router passes the following `x-forwarding-*` headers to the client:

- `x-forwarded-host`
  Contains the **host** header value that was sent by the client to the application router
- `x-forwarded-proto`
  Contains the **protocol** that was used by the client to connect to the application router
- `x-forwarded-for`
  Contains the **address** of the client which connects to the application router
- `x-forwarded-path`
  Contains the original **path** which was requested by the client from the approuter

> ⚠ Caution
>
> When the application router forwards a request to a destination, it blocks the header `host`.

"Hop-by-hop" headers are meaningful only for a single transport-level connection; these headers are not forwarded by the application router. The following headers are classified as "Hop-By-Hop" headers:

- Connection
- Keep-Alive
- Public
- Proxy-Authenticate
- Transfer-Encoding
- Upgrade

You can configure the application router to send additional HTTP headers, for example, by using the `httpHeaders` environment variable.

Sessions

The XS advanced model's application router establishes a session with the client (browser) using a session cookie. The application router intercepts all session cookies sent by back-end services and stores them in its own session. To prevent collisions between the various session cookies, back-end session cookies are not sent to the client. On request, the application router sends the cookies back to the respective back-end services so the services can establish their own sessions.

> **i Note**
>
> Non-session cookies from back-end services **are** forwarded to the client, which might cause collisions between cookies. Applications should be able to handle cookie collisions.

Session Contents

A session established by the XS advanced application router typically contains the following elements:

- Redirect location
  The location to redirect to after login; if the request is redirected to a UAA login form, the original request URL is stored in the session so that, after successful authentication, the user is redirected back to it.
- CSRF token
  The CSRF token value if it was requested by the clients. For more information about protection against Cross Site Request Forgery see CSRF Protection [page 920] below.
- OAuth token
  The JSON Web Token (JWT) fetched from the User Account and Authentication service (UAA) and forwarded to back-end services in the `Authorization` header. The client never receives this token. The application router refreshes the JWT automatically before it expires (if the session is still valid). By default, this routine is triggered 5 minutes before the expiration of the JWT, but it can also be configured with the `<JWT_REFRESH>` environment variable (the value is set in minutes). If `<JWT_REFRESH>` is set to 0, the refresh action is disabled.
- OAuth scopes
  The scopes owned by the current user, which is used to check if the user has the authorizations required for each request
- Back-end session cookies
  All session cookies sent by back-end services

Scaling

The application router keeps all established sessions in local memory, and if multiple instances of the application router are running, there is no synchronization between the sessions. To scale the application router for multiple instances, session stickiness must be enabled so that each HTTP session is handled by the same application router instance. In SAP HANA XS advanced model's on-premise run time, session stickiness is enabled, if the SAP Web Dispatcher is used as a router. In the more recent versions of the SAP HANA XS Advanced run time, the SAP Web Dispatcher is used as a router by default.

Sizing and Memory Configuration

The application-router process should run with at least 256MB memory. The amount of memory actually required depends on the application the router is serving. The following aspects have an influence on the application's memory usage:

- Concurrent connections
- Active sessions

- Size of the Java Web Token
- Back-end session cookies

You can use the start-up parameter `max-old-space-size` to restrict the amount of memory used by the JavaScript heap. The default value for `max-old-space-size` is less than 2GB. To enable the application to use all available resources, the value of `max-old-space-size` should be set to a number equal to the memory limit for the whole application. For example, if the application memory is limited to 2GB, set the heap limit as follows, in the application's `package.json` file:

Sample Code

Set Application Router Memory Usage in `package.json` File

```
"scripts": {
  "start": "node --max-old-space-size=2048 node_modules/@sap/approuter/
approuter.js"
}
```

If the application router is running in an environment with limited memory, set the heap limit to about 75% of available memory. For example, if the application router memory is limited to 256MB, add the following command to your `package.json`:

Sample Code

```
"scripts": {
  "start": "node --max-old-space-size=192 node_modules/@sap/approuter/
approuter.js"
}
```

→ Tip

For detailed information about memory consumption in different scenarios, see the Sizing Guide for the Application Router located in `approuter/approuter.js/doc/sizingGuide.md`

CSRF Protection

The application router enables CSRF protection for any HTTP method that is not `GET` or `HEAD` and the route is not public. A path is considered public, if it does not require authentication. This is the case for routes with `authenticationType: none` or if authentication is disabled completely via the top level property `authenticationMethod: none`.

To obtain a CSRF token one must send a `GET` or `HEAD` request with a `x-csrf-token: fetch` header to the application router. The application router will return the created token in a `x-csrf-token: <token>` header, where `<token>` will be the value of the CSRF token.

If a CSRF protected route is requested with any of the above mentioned methods, `x-csrf-token: <token>` header should be present in the request with the previously obtained token. This request must use the same session as the fetch token request. If the `x-csrf-token` header is not present or is invalid, the application router will return status code “403 - Forbidden”.

Cloud Connectivity

The application router supports integration with SAP Cloud Platform connectivity service. The connectivity service enables you to manage proxy access to SAP Cloud Platform Cloud Connector, which you can use to create tunnels for connections to systems located in private networks, for example, on-premise. To use the connectivity feature, you must create an instance of the connectivity service and bind it to the `Approuter` application.

In addition, the relevant destination configurations in the `<destinations >` environment variable must have the proxy type `"onPremise"`, for example, `"proxyType": "onPremise"`. You must also ensure that you obtain a valid XSUAA login token for the XS advanced User Account and Authentication service.

Troubleshooting

The XS advanced application router uses the `@sap/logging` package, which means that all of the typical logging features are available to control application logging. For example, to set all logging and tracing to the most detailed level, set the `<XS_APP_LOG_LEVEL>` environment variable to "`debug`". For the on-premise XS advanced model run time, you can use the following command to set the logging level to "`debug`" without restarting the target application:

> **i** Note
>
> This could lead to a very large amount of data being written to the application logs and trace files. The asterisk wild card (*) enables options that trace sensitive data that is then written to the logs and traces.

> Sample Code
>
```
xs set-logging-level <application-name> '*' debug
```

> → Tip
>
> Logging levels are application-specific and case-sensitive; they can be defined with lower-case characters (for example, "debug") or upper-case characters (for example, "DEBUG"). An error occurs if you set a logging level incorrectly, for example, using lower-case characters "debug" where the application defines the logging level as "DEBUG".

You can enable additional traces of the incoming and outgoing requests by setting the environment variable `<REQUEST_TRACE>` to true. When enabled basic information will be logged for every incoming and outgoing request of the application router.

The `@sap/logging` package sets the header `x-request-id` in the application router's responses. This is useful if you want to search the application router's logs and traces for entries that belong to a particular request execution. Note that the application router does not change the headers received from the back end and forwarded to the client. If the back end is a Node.js application which uses the `@sap/logging` package (and also sets the `x-request-id` header), then the value of the header that the client receives is the one coming from the back end and not the one from the application router itself.

Related Information

XS Advanced Application Router Configuration Syntax [page 922]
XS Advanced Application Router Environment Variables [page 938]
Configure the XS Advanced Application Router [page 913]
XS Advanced Application-Router Resource Files [page 916]

10.1.3 XS Advanced Application Router Configuration Syntax

The application description defined in the `xs-app.json` file contains the configuration information used by the application router.

The following example of an `xs-app.json` application descriptor shows the JSON-compliant syntax required and the properties that either must be set or can be specified as an additional option.

Code Syntax

```
{
  "welcomeFile": "index.html",
  "authenticationMethod": "route",
  "sessionTimeout": 10,
  "pluginMetadataEndpoint": "/metadata",
  "routes": [
    {
      "source": "^/sap/ui5/1(.*)$",
      "target": "$1",
      "destination": "ui5",
      "csrfProtection": false
    },
    {
      "source": "/employeeData/(.*)",
      "target": "/services/employeeService/$1",
      "destination": "employeeServices",
      "authenticationType": "xsuaa",
      "scope": ["$XSAPPNAME.viewer", "$XSAPPNAME.writer"],
      "csrfProtection": true
    },
    {
      "source": "^/(.*)$",
      "target": "/web/$1",
      "localDir": "static-content",
      "replace": {
        "pathSuffixes": ["/abc/index.html"],
        "vars": ["NAME"]
      }
    }
  ],
  "login": {
     "callbackEndpoint": "/custom/login/callback"
  },
  "logout": {
     "logoutEndpoint": "/my/logout",
     "logoutPage": "/logout-page.html"
  },
  "destinations": {
     "employeeServices": {
       "logoutPath": "/services/employeeService/logout",
       "logoutMethod": "GET"
```

```
      }
   },
   "compression": {
      "minSize": 2048
   },
   "whitelistService": {
      "endpoint": "/allowlist/service"
   },
   "websockets": {
     "enabled": true
   },
   "errorPage": [
     {"status": [400,401,402], "file": "/custom-err-4xx.html"},
     {"status": 501, "file": "/custom-err-501.html"}
   ]
}
```

welcomeFile

The Web page served by default if the HTTP request does not include a specific path, for example, `index.html`.

XS Advanced Application-Router Parameters

Property	Type	Mandatory	Values
`welcomeFile`	String	No	HTML page, for example, `index.html`

Code Syntax

```
"welcomeFile": "index.html",
```

authenticationMethod

The method used to authenticate user requests, for example: “route” or “none” (no authentication).

Code Syntax

```
"authenticationMethod" : "route",
```

Property	Type	Mandatory	Values
`authenticationMethod`	String	No	The following authentication methods are allowed: • `route` Authentication type is defined in the route configuration • `none` Disables authentication for all routes

> ⚠ Caution
>
> If `authenticationMethod` is set to "`none`", logon with User Account and Authentication (UAA) is disabled.

routes

Defines all route objects, for example: `source`, `target`, and, `destination`.

Application Router: Routes Properties

Property	Type	Mandatory	Description
`source`	RegEx	Yes	Regular expression that matches an incoming request path. To ensure the RegEx matches the complete path, wrap it with ^ and $, for example, `"^/sap/ui5/1(.*)$"`. A request matches a particular route if its `path` contains the specified regular expression. → Tip • Use `"matchCase"` to specify if the regular expression defined in `path` is matched in a case-sensitive (`true`) way or not (`false`). Default is `true`.
`httpMethods`	Array of uppercase HTTP methods	No	HTTP methods that will be served by this route; the supported methods are: `DELETE`, `GET`, `HEAD`, `OPTIONS`, `POST`, `PUT`, `TRACE`, and `PATCH`. → Tip If this option is not specified, the route will serve any HTTP method.
`target`	String	No	The incoming request path is rewritten to this target

Property	Type	Mandatory	Description
`destination`	String	No	The destination to which the incoming request should be forwarded. This must match one of the `name` properties defined in the destinations environment variable..
`scope`	String	No	The authorization scope required to access the target path. The scope itself is defined in the application's security descriptor (`xs-security.json`), for example, `"$XSAPPNAME.Display"` or `"$XSAPPNAME.Create"`.
`localDir`	String	No	The directory from which application router serves static content (for example, from the application's `web/` module)
`replace`	Object	No	An object that contains the configuration for replacing placeholders with values from the environment. It is only relevant for static resources.
`csrfProtection`	Boolean	No	Toggle whether this route needs CSRF token protection. The default value is "true". The application router enforces CSRF protection for any HTTP request that changes state on the server side, for example: PUT, POST, or DELETE.
`authenticationType`	String	No	The value can be "`xsuaa`", "`basic`" or "`none`". The default value is "`xsuaa`". When "`xsuaa`" is used the specified UAA server will handle the authentication (the user is redirected to the UAA's login form). The basic mechanism works with SAP HANA users. If "`none`" is used then no authentication is needed for this route.
`cacheControl`	String	No	A string representing the value of the Cache-Control header, which is set on the response when serving static resources. By default the Cache-Control header is not set. **Note** This value is relevant only for static resources.

Code Syntax

```
"routes": [
  {
    "source": "^/sap/ui5/1(.*)$",
    "target": "$1",
    "destination": "ui5",
```

```
    "scope": "$XSAPPNAME.viewer",
    "authenticationType": "xsuaa"
    "csrfProtection": true
  }
]
```

Note

The properties `target`, `destination`, and `localDir` are optional. However, at least one of them **must** be defined.

The properties `destination` and `localDir` cannot be used together in the same route.

If there is no route defined for serving static content via `localDir`, a default route is created for "`resources`" directory as follows:

Sample Code

```
{
  "routes": [
    {
      "source": " ^/(.*)$",
      "localDir": "resources"
    }
  ]
}
```

Note

If there is at least one route using `localDir`, the default route is not added.

The `httpMethods` option allows you to split the same path across different targets depending on the HTTP method. For example:

Sample Code

```
"routes": [
  {
    "source": "^/app1/(.*)$",
    "target": "/before/$1/after",
    "httpMethods": ["GET", "POST"]
  }
]
```

This route will be able to serve only GET and POST requests. Any other method (including extension ones) will get a `405 Method Not Allowed` response. The same endpoint can be split across multiple destinations depending on the HTTP method of the requests:

Sample Code

```
"routes": [
{
  "source": "^/app1/(.*)$",
  "destination" : "dest-1",
  "httpMethods": ["GET"]
},
{
```

```
  "source": "^/app1/(.*)$",
  "destination" : "dest-2",
  "httpMethods": ["DELETE", "POST", "PUT"]
}
]
```

The sample code above will route GET requests to the target `dest-1`, DELETE, POST and PUT to `dest-2`, and any other method receives a `405 Method Not Allowed` response. It is also possible to specify `catchAll` routes, namely those that do not specify `httpMethods` restrictions:

Sample Code

```
"routes": [
{
  "source": "^/app1/(.*)$",
  "destination" : "dest-1",
  "httpMethods": ["GET"]
},
{
  "source": "^/app1/(.*)$",
  "destination" : "dest-2"
}}
]
```

In the sample code above, GET requests will be routed to `dest-1`, and all the rest to `dest-2`.

replace

The `replace` object configures the placeholder replacement in static text resources.

Sample Code

```
{
 "replace": {
   "pathSuffixes": ["index.html"],
   "vars": ["escaped_text", "NOT_ESCAPED"]
 }
}
```

The `replace` keyword requires the following properties:

Replacement Properties for Static Resource URLs

Property	Type	Description
`pathSuffixes`	Array	An array defining the path suffixes that are relative to `localDir`. Only files with a path ending with any of these suffixes will be processed.
`vars`	Array	A list of the environment variables that will be replaced in the files matching the suffix specified in `pathSuffixes`.

The supported tags for replacing environment variables are: `{{ENV_VAR}}` and `{{{ENV_VAR}}}` . If there such an environment variable is defined, it will be replaced, otherwise it will be just an empty string.

Note

Any variable that is replaced using two-brackets syntax `{{ENV_VAR}}` will be HTML-escaped; the triple brackets syntax `{{{ENV_VAR}}}` is used when the replaced values do not need to be escaped and all

values will remain unchanged. For example, if the value of the environment variable is `ab"cd` the result will be `ab"cd`.

If your application descriptor `xs-app.json` contains a route like the one illustrated in the following example,

```
{
 "source": "^/get/home(.*)",
 "target": "$1",
 "localDir": "resources",
 "replace": {
   "pathSuffixes": ["index.html"],
   "vars": ["escaped_text", "NOT_ESCAPED"]
 }
}
```

And your application uses the following `index.html` start file:

Sample Code

```
<html>
 <head>
  <title>{{escaped_text}}</title>
  <script src="{{{NOT_ESCAPED}}}/index.js"/>
 </head>
</html>
```

Then, in `index.html`, `{{escaped_text}}` and `{{{NOT_ESCAPED}}}` will be replaced with the value defined in the environment variables `<escaped_text>` and `<NOT_ESCAPED>`.

Note

All `index.html` files are processed; if you want to apply the replacement only to specific files, you must set the path relative to `localDir`. In addition, all files should comply with the UTF-8 encoding rules.

The content type returned by a request is based on the file extension specified in the route. The application router support the following file types:

- .json (application/json)
- .txt (text/plain)
- .html (text/html) **default**
- .js (application/javascript)
- .css (test/css)

The following table illustrates some examples of the `pathSuffixes` properties:

Examples of the pathSuffixes Property

Example	Result
`{ "pathSuffixes": [".html"] }`	All files with the extension `.html` under `localDir` and its subfolders will be processed.

Example	Result
`{ "pathSuffixes": ["/abc/main.html", "some.html"] }`	For the suffix `/abc/main.html`, all files named main.html which are inside a folder named `abc` will be processed. For the suffix `some.html`, all files with a name that ends with "`some.html`" will be processed, for example: `some.html`, `awesome.html`.
`{ "pathSuffixes": [".html"] }`	All files with the name "`some.html`" will be processed. For example: `some.html` , `/abc/some.html`.

sessionTimeout

Define the amount of time (in minutes) for which a session can remain inactive before it closes automatically (times out); the default time out is 15 minutes.

> **i** Note
>
> The `sessionTimeout` property is no longer available; to set the session time out value, use the environment variable `<SESSION_TIMEOUT>`.

> Sample Code

```
{
  "sessionTimeout": 40,
}
```

With the configuration in the example above, a session timeout will be triggered after 40 minutes and involves central log out.

login

A redirect to the application router at a specific endpoint takes place during OAuth2 authentication with the User Account and Authentication service (UAA). This endpoint can be configured in order to avoid possible collisions, as illustrated in the following example:

> Sample Code
>
> Application Router "login" Property

```
"login": {
  "callbackEndpoint": "/custom/login/callback"
}
```

> → Tip
>
> The default endpoint is "`/login/callback`".

logout

Define any options that apply if you want your application to have central log out end point. In this object you can define an application's central log out end point by using the `logoutEndpoint` property, as illustrated in the following example:

Sample Code

```
"logout" {
  "logoutEndpoint": "/my/logout"
}
```

Making a `GET` request to "`/my/logout`" triggers a client-initiated central log out. Central log out can be initiated by a client or triggered due to a session timeout, with the following consequences:

- Client-initiated central log out
  - Deletes the user session
  - Requests the log out paths for all your back-end services (if you provided these paths in the `destinations` property).
  - Redirects to the User Account and Authentication service (UAA), if such a service is provided, and logs out from there.

    → Tip

    It is not possible to redirect back to your application after logging out from UAA.
- Session time out
  - Deletes the user session
  - Requests the log out paths for all your back-end services (if you provided these paths in the `destinations` property).

You can use the `logoutPage` property to specify the Web page in one of the following ways:

- URL path
  The UAA service redirects the user back to the application router, and the path is interpreted according to the configured routes.

  i Note

  The resource that matches the URL path specified in the property `logoutPage` should not require authentication; for this route, the property `authenticationType` must be set to "`none`". If your XS advanced application enables user logon with single sign-on authentication, a custom logout page is needed to prevent a login-logout loop. For more information see SAP Note 2699214.

  In the following example, `my-static-resources` is a folder in the working directory of the application router; the folder contains the file `logout-page.html` along with other static resources.

  Sample Code

  ```
  {
    "authenticationMethod": "route",
    "logout": {
      "logoutEndpoint": "/my/logout",
      "logoutPage": "/logout-page.html"
  ```

```
  },
  "routes": [
    {
      "source": "^/logout-page.html$",
      "localDir": "my-static-resources",
      "authenticationType": "none"
    }
  ]
}
```

- Absolute HTTP(S) path
  The UAA will redirect the user to a page (or application) different from the application router.

  Sample Code

```
"logout": {
  "logoutEndpoint": "/my/logout",
  "logoutPage": "http://acme.com/employees.portal"
}
```

destinations

Specify any additional options for your destinations. The destinations section in `xs-app.json` extends the destination configuration in the deployment manifest (`manifest.yml`), for example, with some static properties such as a logout path.

Sample Code

```
{
  "destinations": {
    "node-backend": {
      "logoutPath": "/ui5logout",
      "logoutMethod": "GET"
    }
  }
}
```

The following syntax rules apply:

Application Router: Destination Properties

Property	Type	Mandatory	Description
`logoutPath`	String	No	The log out end point for your destination. The `logoutPath` will be called when central log out is triggered or a session is deleted due to a time out. The request to `logoutPath` contains additional headers, including the JWT token.

Property	Type	Mandatory	Description
`logoutMethod`	String	No	The `logoutMethod` property specifies the HTTP method with which the `logoutPath` will be requested, for example, POST, PUT, GET; the default value is POST

compression

The `compression` keyword enables you to define if the application router compresses text resources before sending them. By default, resources larger than 1KB are compressed. If you need to change the compression size threshold, for example, to “2048 bytes”, you can add the optional property `“minSize”`: `<size_in_KB>`, as illustrated in the following example.

Sample Code

```
{
  "compression": {
      "minSize": 2048
  }
}
```

You can disable compression in the following ways:

- Global
  Within the compression section add "enabled": false
- Front end
  The client sends a header “`Accept-Encoding`” which does not include “`gzip`”.
- Back end
  The application sends a header “`Cache-Control`” with the “`no-transform`” directive.

Application Router: Compression Properties

Property	Type	Mandatory	Description
`minSize`	Number	No	Text resources larger than this size will be compressed.
`enabled`	Boolean	No	Globally disables or enables compression. The default value is true.

> **Note**
>
> If the `<COMPRESSION>` environment variable is set it will overwrite any existing values.

pluginMetadataEndpoint

Adds an endpoint that serves a JSON string representing all configured plugins.

Sample Code

```
{
  "pluginMetadataEndpoint": "/metadata"
}
```

Note

If you request the relative path `/metadata` of your application, a JSON string is returned with the configured plug-ins.

whitelistService

This service uses a list of allowed hosts and domains to help prevent click-jacking attacks. Enabling the service opens an endpoint accepting GET requests at the relative path configured in the `endpoint` property, as illustrated in the following example:

Sample Code

```
{
  "whitelistService": {
    "endpoint": "/allowlist/service"
  }
}
```

If the service is enabled in the application router, each time an HTML page needs to be rendered in a frame, the service is used to check if the parent frame is allowed to render the content in a frame.

Allowed Host Names and Domain Names

This service reads a list of allowed host names and domains which are defined in the environment variable `<CJ_PROTECT_WHITELIST>`; the content is a JSON list of objects with the properties specified in the following table:

List of Allowed Host and Domain Names

Property	Type	Mandatory	Description
`protocol`	String	No	URI scheme, for example “HTTP”.
`host`	String	Yes	A valid host name, for example, `acme.com.hostname`, or a domain name defined with an asterisk (*) `*.acme.com`.

Property	Type	Mandatory	Description
`port`	String/Number	No	Port string or number containing a valid port.

The following snippet shows an example of the resulting JSON array:

Sample Code

```
[
 {
  "protocol": "http",
  "host": "*.acme.com",
  "port": 12345
 },
 {
  "host": "hostname.acme.com"
 }
]
```

Note

Matching is done against the properties provided. For example, if only host name is provided, the allowlist service returns "`framing: true`" for all, and matching will be for all schemata and protocols.

Return Value

The allowlist service accepts only `GET` requests, and returns a JSON object as the response; the service call uses the parent origin as URI parameter (URL encoded) as follows:

Sample Code

```
GET url/to/allowlist/service?parentOrigin=https://parent.domain.acme.com
```

The response is a JSON object with the following properties; property "`active`" has the value `false` only if `<CJ_PROTECT_WHITELIST>` is not provided:

Sample Code

```
{
  "version" : "1.0",
  "active" : true | false,
  "origin" : "<same as passed to service>",
  "framing" : true | false
}
```

The "`active`" property enables framing control; the "`framing`" property specifies if framing should be allowed. By default, the Application Router (`approuter.js`) sends the `X-Frame-Options` header with value the `SAMEORIGIN`.

Tip

If the allowlist service is enabled, the header value probably needs to be changed, see the *X-Frame-Options* header section for details about how to change it.

websockets

The application router can forward web-socket communication. Web-socket communication must be enabled in the application router configuration, as illustrated in the following example. If the back-end service requires authentication, the upgrade request should contain a valid session cookie. The application router supports the destination schemata "`ws`", "`wss`", "`http`", and "`https`".

Sample Code

```
{
  "websockets": {
    "enabled": true
  }
}
```

! Restriction

A web-socket ping is not forwarded to the back-end service.

errorPage

Errors originating in the application router show the HTTP status code of the error. It is possible to display a custom error page using the `errorPage` property.

The property is an array of objects, each object can have the following properties:

Application Router: errorPage Properties

Property	Type	Mandatory	Description
`status`	Number/Array	Yes	HTTP status code.
`file`	String	Yes	File path relative to the working directory of the application router.

In the following code example, errors with status code “400”, “401” and “402” will show the content of `./custom-err-4xx.html`; errors with the status code “501” will display the content of `./http_resources/custom-err-501.html`.

Sample Code

```
{ "errorPage" : [
    {"status": [400,401,402], "file": "./custom-err-40x.html"},
    {"status": 501, "file": "./http_resources/custom-err-501.html"}
  ]
}
```

> **Note**
>
> The contents of the `errorPage` configuration section have no effect on errors that are not generated by the application router.

Related Information

XS Advanced Application Router Environment Variables [page 938]
Application Security Descriptor Configuration Syntax [page 999]
The XS Advanced Application Descriptor [page 917]
Configure the XS Advanced Application Router [page 913]

10.1.4 XS Advanced Application Routes and Destinations

The application router is the single point of entry for an application.

The application router is used to serve static content, propagate user information, and act as a proxy to forward requests to other micro services. The routing configuration for an application is defined in one or more destinations. The application router configuration is responsible for the following tasks:

- Act as the main user-entry point to an application service
- Serve static content
- Serve routes and destinations
- Rewrite URLs
- Forward requests to other services
- Propagate user information
- Handle authentication
- Manage HTML5 application/browser sessions

> **Note**
>
> The application router does not manage server caching. The server cache must be set (with e-tags) in the server itself. The cache should contain not only static content but container resources, too, for example, relating to OData metadata.

In Cloud-based scenarios, a request is routed by means of domain names and a central router component. This requires a DNS wild card entry so that all requests are redirected to the "Cloud Router" which does dispatching and load balancing. For SAP on-Premise, it is also possible (as a fall back) to provide a routing service based on ports. In this scenario, the router opens a dedicated in-bound port for each application name, and each incoming request to an in-bound port is mapped to a range of outgoing addresses. The central router uses the mapping for load-balancing and forwards the request to the appropriate target.

Destinations

A destination defines the back-end connectivity. In its simplest form, a destination is a URL to which requests are forwarded.

Destinations are defined in an environment variable for the `approuter` application. The destinations are typically specified in the application manifest file (`manifest.yml`). The router information is added to the `env: destinations` section of the `manifest.yml` file, as illustrated in the following example:

Sample Code

Destinations Defined in the Application Manifest (`manifest.yml`)

```
---
applications:
- name: node-hello-world
  port: <approuter-port>  #the port used for the approuter
  memory: 100M
  path: web
  env:
    destinations: >
      [
        {
          "name":"backend",
          "url":"http://<hostname>:<node-port>",
          "forwardAuthToken": true
        }
      ]
  services:
    - node-uaa
```

Note

The “`name`” and “`url`” properties are mandatory.

The value of the destination “`name`” property (“`name`”:“`backend`” in the example below) must match the value of the `destinations` property configured for a route in the corresponding application-router configuration file (`xs-app.json`). It is also possible to define a logout path and method for the `destinations` property in the `xs-app.json` file. For more information, see the section describing the application-router configuration syntax in *Related Information*.

Related Information

Configure the XS Advanced Application Router [page 913]
XS Advanced Application-Router Resource Files [page 916]
XS Advanced Application Router Configuration Syntax [page 922]
Configure the XS Advanced Application Router [page 913]

10.1.5 XS Advanced Application Router Environment Variables

A list of environment variables that can be used to configure the application router.

The following table lists the environment variables that you can use to configure the application router. The table also provides a short description of each variable and, where appropriate, an example of the configuration data.

Application Router Configuration Variables

Variable	Description
`CJ_PROTECT_WHITELIST`	A list of allowed server or domain origins to use when checking for click-jacking attacks
`COMPRESSION`	Configure the compression of resources before a response to the client.
`CORS`	Provide support for cross-origin requests, for example, by allowing the modification of the request header.
`destinations`	Provide information about the available application (microservice) destinations.
`httpHeaders`	Configure the application router to return additional HTTP headers in its responses to client requests
`INCOMING_CONNECTION_TIMEOUT`	Specify the maximum time (in milliseconds) for a client connection. If the specified time is exceeded, the connection is closed.
`JWT_REFRESH`	Configures the automatic refresh of the JSON Web Token (JWT) provided by the User Account and Authentication (UAA) service to prevent expiry (default is 5 minutes).
`REQUEST_TRACE`	Enable additional traces of the incoming and outgoing requests
`SECURE_SESSION_COOKIE`	Configure the enforcement of the `Secure` flag of the application router's session cookie.
`SEND_XFRAMEOPTIONS`	Set or change the `X-Frame-Options` header
`SESSION_TIMEOUT`	Set the time to trigger an automatic central log out from the User Account and Authentication (UAA) server.
`TENANT_HOST_PATTERN`	Define a regular expression to use when resolving tenant host names in the request's host name.
`UAA_SERVICE_NAME`	Specify the **exact** name of the UAA service to bind to an application.
`WS_ALLOWED_ORIGINS`	A list of the allowed server (or domain) origins that the application router uses to verify requests

httpHeaders

If configured, the application router sends additional HTTP headers in its responses to a client request. You can set the additional HTTP headers in the `<httpHeaders>` environment variable. The following example configuration shows how to configure the application router to send two additional headers in the responses to the client requests from the application `<myApp>`:

```
xs set-env <myApp> httpHeaders "[ { \"X-Frame-Options\": \"ALLOW-FROM http://
acme.com\" }, { \"Test-Additional-Header\": \"1\" } ]"
```

or

```
xs set-env <myApp> httpHeaders '[{ "X-Frame-Options": "ALLOW-FROM http://
acme.com" }, { "Test-Additional-Header": "1" }]'
```

> → Tip
>
> To improve security in your application, set the `Content-Security-Policy` response header, which can be used to provide those Web browsers capable of interpreting it with information about the trusted sources from which an application can load resources.

The content-security-policy mechanism allows the client to detect and block any attempts to inject malicious scripts into the application. A value can be set for the security policy by using the `<httpHeaders>` environment variable in the additional headers configuration. The value set represents a security policy which contains directive-value pairs. The value set for directive takes the form of a list of trusted sources. You can use the `Content-Security-Policy` to define a default source (`default-src`) or define individual settings, for example, to restrict connections (`connect-src`), limit the type of plug-ins that can be run (`plugin-types`), or ensure that scripts are only loaded from a permitted origin (`script-src`).

> i Note
>
> The `Content-Security-Policy` header is seen as a second line of defense; applications should always provide and ensure suitable input validation and output encoding.

destinations

The destinations configuration is an array of objects that is defined in the `destinations` environment variable. A destination is required for every application (microservice) that is part of the business application. The following table lists the properties that can be used to describe the destination:

Destination Environment Variable Properties

Property	Type	Mandatory	Description
`name`	String	Yes	A unique indentifier for the destination
`url`	String	Yes	The Unique Resource Locator for the application (microservice)

Property	Type	Mandatory	Description
`proxyHost`	String	No	The host of the proxy server used in case the request should go through a proxy to reach the destination. → Tip Mandatory if `proxyPort` is defined.
`proxyPort`	String	No	The port of the proxy server used in case the request should go through a proxy to reach the destination. → Tip Mandatory if `proxyHost` is defined.
`forwardAuthToken`	Boolean	No	If true, the OAuth token will be sent to the destination. The default value is “false”. This token contains the user identity, scopes, and some other attributes. The token is signed by the User Account and Authentication (UAA) service so that the token can be used for user-authentication and authorization purposed by back-end services.
`strictSSL`	Boolean	No	Configures whether the application router should reject untrusted certificates. The default value is “true”. ⚠ Caution For testing purposes only. Do **not** use this property in production environments!
`timeout`	Number	No	A positive integer representing the maximum amount of time to wait for a response (in milliseconds) from the specified destination. The default is 30000ms. → Tip The timeout specified here also applies to the logout path, `logoutPath`, if the logout path is defined, for example, in the application's descriptor file `xs-app.json`.

Property	Type	Mandatory	Description
`proxyType`	String	No	Indicates if the destination is used to access applications in private networks (for example, on-premise) or publicly available on the Internet. Possible value: `onPremise`. If `proxyType` is not specified, the default (Internet access) is assumed. **→ Tip** In Cloud environments, if you set the application's destination `proxyType` to `onPremise`, a binding to the SAP Cloud Platform connectivity service is required, and the `forwardAuthToken` property must not be set.

The following example shows a simple configuration for the `<destinations>` environment variable:

Sample Code

```
[
 {
   "name" : "ui5",
   "url" : "https://sapui5.acme.com",
   "proxyHost" : "proxy",
   "proxyPort" : "8080",
   "forwardAuthToken" : false,
   "timeout" : 1200
 }
]
```

It is also possible to include the destinations in the `manifest.yml` file, as illustrated in the following example:

Sample Code

```
- name: node-hello-world
  memory: 100M
  path: web
  env: destinations: >
       [
        {"name":"ui5", "url":"https://sapui5.acme.com"}
       ]
```

The Destination Service

It is also possible to read a "destination" configuration from the `destination service` in Cloud Foundry. For this configuration scenario, however, some restrictions apply regarding where the service can be used and what properties and options are recognized by the application router.

! Restriction

The destination service is only available in Cloud Foundry; it is not available for use in on-premise environments.

If a destination with the same name is defined in both an environment destination **and** a destination service, the environment destination configuration takes precedence and will be loaded and used.

AppRouter Limitations for the Destination Service in Cloud Foundry

Property	Optional	Destination Service Restrictions
`type`	No	Supported protocol type: `HTTP` only
`Authentication`	No	Supported authentication types: • `none` • `basic authentication` `User` and `Password` are mandatory • `principal propagation` `ProxyType` must be "`on-premise`"
`ProxyType`	No	Supported proxy types: • `on-premise` A binding to the SAP Cloud Platform connectivity service is required. • `internet`
`ForwardAuthToken`	Yes	If set to `true`, the OAuth token is sent to the specified destination. The default value is `false`. The OAuth token contains the user identity, authorization scopes, and other attributes. Since the token is signed by the UAA, it can be used for user authentication and authorization with back-end services. **i Note** If `ProxyType` is set to `on-premise`, the `ForwardAuthToken` property cannot be set.
`Timeout`	Yes	A positive integer representing the maximum amount of time (in milliseconds) to wait for a response from the destination. The default value is 30000ms. **i Note** The timeout specified here also applies to the destination's `logout path` (if one is set).

SESSION_TIMEOUT

You can configure the triggering of an automatic central log-out from the User Account and Authentication (UAA) service if an application session is inactive for a specified time. A session is considered to be inactive if no requests are sent to the application router. The following command shows how to set the environment variable `<SESSION_TIMEOUT>` to 40 (forty) minutes for the application `<myApp1>`:

```
xs set-env <myApp1> SESSION_TIMEOUT 40
```

> **Note**
>
> You can also set the session timeout value in the application's `manifest.yml` file, as illustrated in the following example:

> **Sample Code**

```
- name: myApp1
  memory: 100M
  path: web
  env:
    SESSION_TIMEOUT: 40
```

> **Tip**
>
> If the authentication type for a route is set to `"xsuaa"` (for example, `"authenticationType": "xsuaa"`), the application router depends on the UAA server for user authentication, and the UAA server might have its own session timeout defined. To avoid problems caused by unexpected timeouts, it is recommended that the session timeout values configured for the application router and the UAA are identical."

SEND_XFRAMEOPTIONS

By default, the application router sends the `X-Frame-Options` header with the value "`SAMEORIGIN`". You can change this behavior either by disabling the sending of the default header value (for example, by setting SEND_XFRAMEOPTIONS environment variable to false) or by overriding the value, for example, by configuring additional headers (with the `<httpHeaders>` environment variable.

The following example shows how to disable the sending of the `X-Frame-Options` for a specific application, `myApp1`:

```
xs set-env <myApp1> SEND_XFRAMEOPTIONS false
```

CJ_PROTECT_WHITELIST

The `<CJ_PROTECT_WHITELIST>` specifies a list of origins (for example, host or domain names) that do not need to be protected against click-jacking attacks. This list of allowed host names and domains are used by the application router's allow-list service to protect XS advanced applications against click-jacking attacks. When an HTML page needs to be rendered in a frame, a check is performed to validate if the parent frame is allowed to render the requested content. The check itself is performed by the allow-list service, which can be enabled using the `whitelistService` parameter in the application-router configuration file `xs-app.json`.

The following example shows how to add a host name and corresponding port number to the click-jacking-protection allow list for the application, `myApp1`:

```
xs set-env <myApp1> CJ_PROTECT_WHITELIST {<protocol>, <hostname>, <portNr>}
```

The content is a JSON list of objects with the properties listed in the following table:

Details of the List of Allowed Host and Domain Names

Property	Type	Mandatory	Description
`protocol`	String	No	URI scheme, for example “HTTP”.
`host`	String	Yes	A valid host name, for example, `acme.com.hostname`, or a domain name defined with an asterisk (*) `*.acme.com`.
`port`	String/Number	No	Port string or number containing a valid port.

> **i Note**
>
> Matching is done against the properties provided. For example, if only the host name is provided, the allow-list service matches all schemata and protocols.

```
xs set-env <myApp1> CJ_PROTECT_WHITELIST {<*.acme.com>}
```

WS_ALLOWED_ORIGINS

When the application router receives an upgrade request, it verifies that the `origin` header includes the URL of the application router. If this is not the case, then an HTTP response with status 403 is returned to the client. This origin verification can be further configured with the environment variable `<WS_ALLOWED_ORIGINS>`, which defines a list of the allowed origins the application router uses in the verification process.

> **i Note**
>
> The structure of the `<WS_ALLOWED_ORIGINS>` variable is the same as the variable `<CJ_PROTECT_WHITELIST>`.

```
xs set-env <myApp1> WS_ALLOWED_ORIGINS{<*.acme.com>}
```

JWT_REFRESH

The `JWT_REFRESH` environment variable is used to configure the application router to refresh a JSON Web Token (JWT) for an application, by default, 5 minutes before the JWT expires, if the session is active.

```
xs set-env <myApp1> JWT_REFRESH 1
```

If the JWT is close to expiration and the session is still active a JWT refresh will be triggered `<JWT_REFRESH>` minutes before expiration. The default value is 5 minutes. To disable the automatic refresh, set the value of `<JWT_REFRESH>` to 0 (zero).

UAA_SERVICE_NAME

The `UAA_SERVICE_NAME` environment variable enables you to configure an instance of the User Account and Authentication service for a specific application, as illustrated in the following example:

```
xs set-env <myApp1> UAA_SERVICE_NAME <myUAAServiceName>
```

> **Note**
>
> The details of the service configuration is defined in the `<VCAP_SERVICES>` environment variable, which is not configured by the user.

INCOMING_CONNECTION_TIMEOUT

The `INCOMING_CONNECTION_TIMEOUT` environment variable enables you to set the maximum time (in milliseconds) allowed for a client connection, as illustrated in the following example:

```
xs set-env <myApp1> INCOMING_CONNECTION_TIMEOUT 60000
```

If the specified time is exceeded, the connection is closed. If `INCOMING_CONNECTION_TIMEOUT` is set to zero (0), the connection-timeout feature is disabled. The default value for `INCOMING_CONNECTION_TIMEOUT` is 120000 ms (2 min).

TENANT_HOST_PATTERN

The `TENANT_HOST_PATTERN` environment variable enables you to specify a string containing a regular expression with a capturing group. The requested host is matched against this regular expression. The value of the first capturing group is used as the tenant Id., as illustrated in the following example:

```
xs set-env <myApp1> TENANT_HOST_PATTERN
```

COMPRESSION

The `COMPRESSION` environment variable enables you to configure the compression of resources before a response to the client, as illustrated in the following example:

```
xs set-env <myApp1> COMPRESSION
```

SECURE_SESSION_COOKIE

The `SECURE_SESSION_COOKIE` can be set to *true* or *false*. By default, the `Secure` flag of the session cookie is set depending on the environment the application router runs in. For example, when the application router

is running behind a router that is configured to serve HTTPS traffic, then this flag will be present. During local development the flag is not set.

> **i** Note
>
> If the `Secure` flag is enforced, the application router will reject requests sent over unencrypted connections.

The following example illustrates how to set the `SECURE_SESSION_COOKIE` environment variable:

```
xs set-env <myApp1> SECURE_SESSION_COOKIE true
```

REQUEST_TRACE

You can enable additional traces of the incoming and outgoing requests by setting the environment variable `<REQUEST_TRACE>` to "`true`". If enabled, basic information is logged for every incoming and outgoing request to the application router.

The following example illustrates how to set the `REQUEST_TRACE` environment variable:

```
xs set-env <myApp1> REQUEST_TRACE true
```

> **→** Tip
>
> This is in addition to the information generated by the Node.js package `@sap/logging` that is used by the XS advanced application router.

CORS

The `CORS` environment variable enables you to provide support for cross-origin requests, for example, by allowing the modification of the request header. Cross-origin resource sharing (CORS) permits Web pages from other domains to make HTTP requests to your application domain, where normally such requests would automatically be refused by the Web browser's security policy.

Cross-origin resource sharing (CORS) is a mechanism that allows restricted resources on a Web page to be requested from another domain (protocol and port) outside the domain (protocol and port) from which the first resource was served. The CORS configuration enables you to define details to control access to your application resource from other Web browsers. For example, you can specify where requests can originate from or what is allowed in the request and response headers. The following example illustrates a basic CORS configuration:

```
[
  {
      "uriPattern": "^\route1$",
      "allowedMethods": [
        "GET"
      ],
      "allowedOrigin": [
        {
          "host": "host.acme.com",
          "protocol": "https",
```

```
          "port": 345
        }
      ],
      "maxAge": 3600,
      "allowedHeaders": [
        "Authorization",
        "Content-Type"
      ],
      "exposeHeaders": [
        "customHeader1",
        "customHeader2"
      ],
      "allowedCredentials": true
    }
]
```

The CORS configuration includes an array of objects with the following properties, some of which are mandatory:

Available Settings for CORS Options

CORS Property	Type	Mandatory	Description
`uriPattern`	String	Yes	A regular expression (RegExp) representing the source routes to which the CORS configuration applies. To ensure that the RegExp matches the complete path, surround it with ^ and $,f or example, `"uriPattern": "^\route1$"`. Defaults: none
`allowedOrigin`	Array	Yes	A comma-separated list of objects each of which contains a host name, port and protocol that are allowed by the server, for example: `[{“host”: "www.acme.com"}]` or `[{“host”: “.acme.com”}]`. **i Note** Matching is case-sensitive. In addition, if no port or protocol is specified, the default is `“*”`. The defaults configuration is: `[{“host”: "*"}]`, which means that the server allows **any** origin to access the resource.
`allowedMethods`	Array	No	A comma-separated list of HTTP methods that are allowed by the server, for example, `“GET”, “POST”`. If `allowMethods` is defined but no method is specified, the default `“GET”, “POST”, “HEAD”, “OPTIONS”` (all) applies. **→ Tip** The specified methods must be upper-case, for example,`GET`. Matching of the method type is case-sensitive.

CORS Property	Type	Mandatory	Description
`allowedHeaders`	Array	No	A comma-separated list of request headers that are allowed by the server. the default values are as follows: `["Origin", "Accept", "X-Requested-With", "Content-Type", "Access-Control-Request-Method", "Access-Control-Request-Headers"]`.
`maxAge`	String	No	A single value specifying the length of time (in seconds) a preflight request should be cached for. A negative value the prevents CORS filter from adding this response header to the pre-flight response. If `maxAge` is defined but no value is specified, the default time of "1800" seconds applies.
`exposeHeaders`	Array	No	A comma-separated list of **response** headers that are allowed to be exposed. If `exposeHeaders` is defined but no response header is specified for exposure, no default value is supplied.
`allowedCredentials`	Boolean	No	A flag that indicates if the specified resource supports user credentials. The default setting is "true".

It is also possible to include the CORS configuration in either the `manifest.yml` or the `manifest-op.yml` file. The code in the following example enables the CORS configuration for any route with a source URI pattern that matches the RegExp "`^\route1$`":

Sample Code

CORS Configuration in the `manifest.yml` File

```
- name: node-hello-world
  memory: 100M
  path: web
  env:
    CORS: >
      [
        {
          "allowedOrigin":[
                          {
                              "host":"my_host",
                              "protocol":"https"
                          }
                         ],
          "uriPattern":"^/route1$"
        }
```

Related Information

Configure the XS Advanced Application Router [page 913]

XS Advanced Application Router Configuration Syntax [page 922]

The XS Advanced Application Descriptor [page 917]

10.2 Extend the XS Advanced Application Router

Use middleware and custom extension scripts to extend the functionalilty of the XS advanced application router.

Context

There are various ways you can extend the functionality provided with the XS advanced application router. You can configure XS advanced JavaScript applications to use the application router as a regular Node.js package or start the application router from a dedicated script. You can inject custom middleware into the application router using predefined slots, and ensure that proper control is maintained over any injected application-router middleware.

Procedure

1. Start the application router with your own JavaScript.

   Instead of starting the application router directly, you can configure your XS advanced application to use its own script to start the application router, as illustrated in the following example:

   Sample Code

   ```
   var approuter = require('approuter');
   var ar = approuter();
   ar.start();
   ```

2. Inject custom middleware into the application router.
   a. Inject middleware directly in the application-router startup script.

      The application router uses the connect framework, which enables you to reuse all injected “connect” middleware within the application router, for example, directly in the application start script:

      Sample Code

      ```
      var approuter = require('approuter');
      var ar = approuter();
      ar.beforeRequestHandler.use('/my-ext', function myMiddleware(req, res,
      next) {
        res.end('Request handled by my extension!');
      });
      ar.start();
      ```

> ### → Tip
>
> To facilitate troubleshooting, always provide a name for your custom middleware.

b. Add some `use` calls to middleware functions.

   The `path` argument is optional. You can also chain `use` calls, as illustrated in the following example:

   ### Sample Code

   ```
   var approuter = require('approuter');
   var morgan = require('morgan');
   var ar = approuter();
   ar.beforeRequestHandler
     .use(morgan('combined'))
     .use('/my-ext', function myMiddleware(req, res, next) {
       res.end('Request handled by my extension!');
     });
   ar.start();
   ```

   The application router also allows you to insert custom middleware into additional "slots": `first`, `beforeRequestHandler`, and `beforeErrorHandler`.

c. Return control to the application-router middleware, if necessary.

   If your middleware does not complete the request processing, call `next` to return control to the application router middleware, as illustrated in the following example:

   ### Sample Code

   ```
   ar.beforeRequestHandler.use('/my-ext', function myMiddleware(req, res,
   next) {
     res.setHeader('x-my-ext', 'passed');
     next();
   });
   ```

3. Configure application-specific extensions to the XS advanced application router.

   You can use an extension file, for example, `my-ext.js`) to insert middleware at the following points: `first`, `beforeRequestHandler`, and `beforeErrorHandler`.

   ### Sample Code

   Example Approuter Extension Configuration (`my-ext.js`)

   ```
   module.exports = {
     insertMiddleware: {
       first: [
         function logRequest(req, res, next) {
           console.log('Got request %s %s', req.method, req.url);
         }
       ],
       beforeRequestHandler: [
         {
           path: '/my-ext',
           handler: function myMiddleware(req, res, next) {
             res.end('Request handled by my extension!');
           }
         }
       ]
     }
   ```

```
};
```

Related Information

Custom Middleware Injection [page 953]
XS Advanced Application Router Extensions [page 951]
XS Advanced Application Router Extension API [page 954]
Maintaining XS Advanced Application Routes and Destinations [page 912]

10.2.1 XS Advanced Application Router Extensions

Configure application-specific extensions to the XS advanced application router.

An application-router extension is defined by an object with the following properties:

- `insertMiddleware` - describes the middleware provided by this extension
  - `first`, `beforeRequestHandler`, and `beforeErrorHandler`
    An array of middleware, where each one can be either of the following elements:
    - A middleware function (invoked on all requests)
    - An object with the following properties:
      - `path`
        Handle requests only for this path
      - `handler`
        The middleware function to invoke

Sample Code

Example Approuter Extension Configuration (`my-ext.js`)

```
module.exports = {
  insertMiddleware: {
    first: [
      function logRequest(req, res, next) {
        console.log('Got request %s %s', req.method, req.url);
      }
    ],
    beforeRequestHandler: [
      {
        path: '/my-ext',
        handler: function myMiddleware(req, res, next) {
          res.end('Request handled by my extension!');
        }
      }
    ]
  }
};
```

The extension configuration can be referenced in the corresponding application's start script, as illustrated in the following example:

Sample Code

```
var approuter = require('approuter');
var ar = approuter();
ar.start({
  extensions: [
    require('./my-ext.js')
  ]
});
```

Customize the Command Line

By default the application router handles its command-line parameters, but that can be customized as well.

An `<approuter>` instance provides the property `cmdParser` that is a commander instance. It is configured with the standard application router command line options. You can add custom options like this:

Sample Code

```
var approuter = require('approuter');
var ar = approuter();
var params = ar.cmdParser
  // add here custom command line options if needed
  .option('-d, --dummy', 'A dummy option')
  .parse(process.argv);
console.log('Dummy option:', params.dummy);
```

To disable the handling of command-line options in the application router, reset the `ar.cmdParser` property to "`false`", as illustrated in the following example:

```
ar.cmdParser = false;
```

Related Information

XS Advanced Application Router Extension API [page 954]
Extend the XS Advanced Application Router [page 949]
Custom Middleware Injection [page 953]
XS Advanced Application Router Extension API [page 954]

10.2.2 Custom Middleware Injection

Extend the application router by injecting so-called middleware plug-ins and using the “Connect” framework.

The application router uses the connect framework, for more information, see *Connect framework* in the *Related Links* below. You can reuse all injected “connect” middleware within the application router, for example, directly in the application start script:

Sample Code

```
var approuter = require('approuter');
var ar = approuter();
ar.beforeRequestHandler.use('/my-ext', function myMiddleware(req, res, next) {
  res.end('Request handled by my extension!');
});
ar.start();
```

→ Tip

To facilitate troubleshooting, always provide a name for your custom middleware.

The `path` argument is optional. You can also chain `use` calls.

Sample Code

```
var approuter = require('approuter');
var morgan = require('morgan');
var ar = approuter();
ar.beforeRequestHandler
  .use(morgan('combined'))
  .use('/my-ext', function myMiddleware(req, res, next) {
    res.end('Request handled by my extension!');
  });
ar.start();
```

The application router defines the following slots where you can insert custom middleware:

- `first` - right after the connect application is created, and before any application router middleware. At this point security checks are not performed yet.

  → Tip

  This is good place for infrastructure logic, for example, logging and monitoring.

- `beforeRequestHandler` - before standard application router request handling, that is static resource serving or forwarding to destinations.

  → Tip

  This is a good place to handle custom REST API requests.

- `beforeErrorHandler` - before standard application router error handling.

  → Tip

  This is a good place to capture or customize error handling.

If your middleware does not complete the request processing, call `next` to return control to the application router middleware, as illustrated in the following example:

Sample Code

```
ar.beforeRequestHandler.use('/my-ext', function myMiddleware(req, res, next) {
  res.setHeader('x-my-ext', 'passed');
  next();
});
```

Related Information

Middleware Connect Framework
Extend the XS Advanced Application Router [page 949]
XS Advanced Application Router Extensions [page 951]
XS Advanced Application Router Extension API [page 954]

10.2.3 XS Advanced Application Router Extension API

A detailed list of the features and functions provided by the application router extension API.

The application router extension API enables you to create new instances of the application router, manage the approuter instance, and insert middleware using the Node.js “connect” framework. This section contains detailed information about the following areas:

- Approuter Extension API Reference
- Middleware Slot

Approuter Extension API Reference

The application router uses the “Connect” framework for the insertion of middleware components. You can reuse all connect middleware within the application router directly.

Approuter Extension API Functions

API Function	Description
`approuter()`	Creates a new instance of the application router

API Function	Description
`first`	Defines a "middleware slot" (a slot for the insertion of middleware) immediately after the `connect` application is created, and before any application router middleware → Tip This is a good place to insert infrastructure logic, for example, logging and monitoring.
`beforeRequestHandler`	Defines a "middleware slot" before the standard application router request handling, that is; static resource serving or forwarding to destinations. → Tip This is a good place to handle custom REST API requests.
`beforeErrorHandler`	Defines a "middleware slot" before the standard application router error handling → Tip This is a good place to capture or customize error handling.

`start(options, callback)`	Starts the application router with the given options.
	• `options` this argument is optional. If provided, it should be an object which can have any of the following properties:
	  • `port` - a TCP port the application router will listen to (string, optional)
	  • `workingDir` - the working directory for the application router, should contain the `xs-app.json` file (string, optional)
	  • `extensions` - an array of extensions, each one is an object as defined in Application Router Extensions (optional)
	  • `xsappContext` - An object representing the content which is usually put in `xs-app.json` file. If this property is present it will take precedence over the content of `xs-app.json`.
	• `callback` - optional function with signature `callback(err)`. It is invoked when the application router has started or an error has occurred. If not provided and an error occurs (for example the port is busy), the application will abort.

`close(callback)`	Stops the application router. • `callback` - optional function with signature `callback(err)`. It is invoked when the application router has stopped or an error has occurred.

Middleware Slot

Manage the insertion of middleware slots with the application router.

Middleware Slot Management

Function	Description
`use(path, handler)`	Inserts a request handling middleware in the current slot. • `path` Handle only requests starting with this path (string, optional) • `handler` A middleware function to invoke (function, mandatory) Returns `this` for chaining.

Related Information

Maintaining XS Advanced Application Routes and Destinations [page 912]

XS Advanced Application Router Extensions [page 951]

Middleware Connect Framework

11 Maintaining Application Security in XS Advanced

Set up the components required in the context of application security.

XS Advanced applications contain content (for example, Web content and micro services), which is deployed to different containers. Static Web content is deployed to the application router; application logic is deployed to Node.js and Java containers; database logic is deployed to the SAP HANA database. Access to the content requires user identification and authentication as well as the appropriate authorization. Additional security can be built into the business logic code (Java or JavaScript) and the SQL connection to the database.

> **i** Note
>
> For more information about the tools available to enhance security in the application code, itself, see the sections describing the Java and JavaScript run-time environments in the *SAP HANA Developer Guide for XS Advanced*. For more information about enhancing security in SQL and SQLScript, see the corresponding SQL and SQLScript Reference Guides.

An XS advanced application must be able to identify and authenticate users, and ensure that the required authorizations are defined and granted, too.

The authentication process is triggered by the application router component, which is configured in the design-time artifact `xs-app.json`, often referred to as the Application Descriptor. Authorizations are defined in the security descriptor (`xs-security.json`) with so-called scopes; authorization scopes enable you to restrict access to resources based on user permissions specified in role templates. In the context of XS advanced applications, "resources" are services provided by a container (for example, an OData Web service) or SAP HANA database artifacts such as tables and views in an HDI container.

XS advanced applications perform the authentication process with OAuth 2.0; the applications use the XS advanced User Account and Authentication (UAA) service as an "OAuth 2.0 authorization server". The OAuth 2.0 client is the application router, and the role of the "OAuth 2.0 resource server" is performed by the application logic running in Node.js and Java back-end services.

The Security Descriptor

The central component for the configuration and maintenance of application security is the `xs-security.json` file, which is often referred to as the Security Descriptor. The contents of the `xs-security.json` file describe any required user authorizations (for example, with OAuth 2) using scopes (view, edit), attributes (dynamically defined cost center, department, etc.), and role templates (used to build roles).

> Sample Code

```
AppName
|- db/
|   |- package.json
|   |- src/
|       |- .hdiconfig
```

```
|     |- .hdinamespace
|     |- myEntity.hdbcds
|     |- myDataType.hdbcds
|     \- myDoc.hdbcds
|- web/
|  |- xs-app.json
|  \- resources/
|- js/
|  |- start.js
|  |- package.json
|  \- src/
|- xs-security.json        # Security artifacts/scopes/auths
|- manifest.yml
\- mtad.yaml
```

Related Information

Authorization in SAP HANA XS Advanced [page 961]
Set up Application Security in XS Advanced [page 978]
Create the Security Descriptor for an XS Advanced Application [page 995]

11.1 Identity, Authentication, and Authorization

Understand the difference between user identity, authentication, and authorization.

To enable access to an XS advanced application, and ensure this access is only granted to those people who can identify themselves and have the required authorization, it is important to ensure that the application can perform the following, basic, security-related tasks:

1. Identify a user.
   Retain user data for use during the identification process and distinguish between users by a unique ID, for example, with user name, e-mail address, etc.
2. Authenticate a user.
   Establish and prove the identity of a user based on a user-secret that is known to the application, for example, a password, a security certificate, a token, or Single Sign-On (SSO)
3. Assign authorizations to a user.
   Grant the defined authorizations to users who can identify and authenticate themselves and enforce the granted authorities according to defined system-usage policies.

> **i** Note
>
> How this functionality is provided differs according to the context in which the application is running, for example, on-premise of in the Cloud.

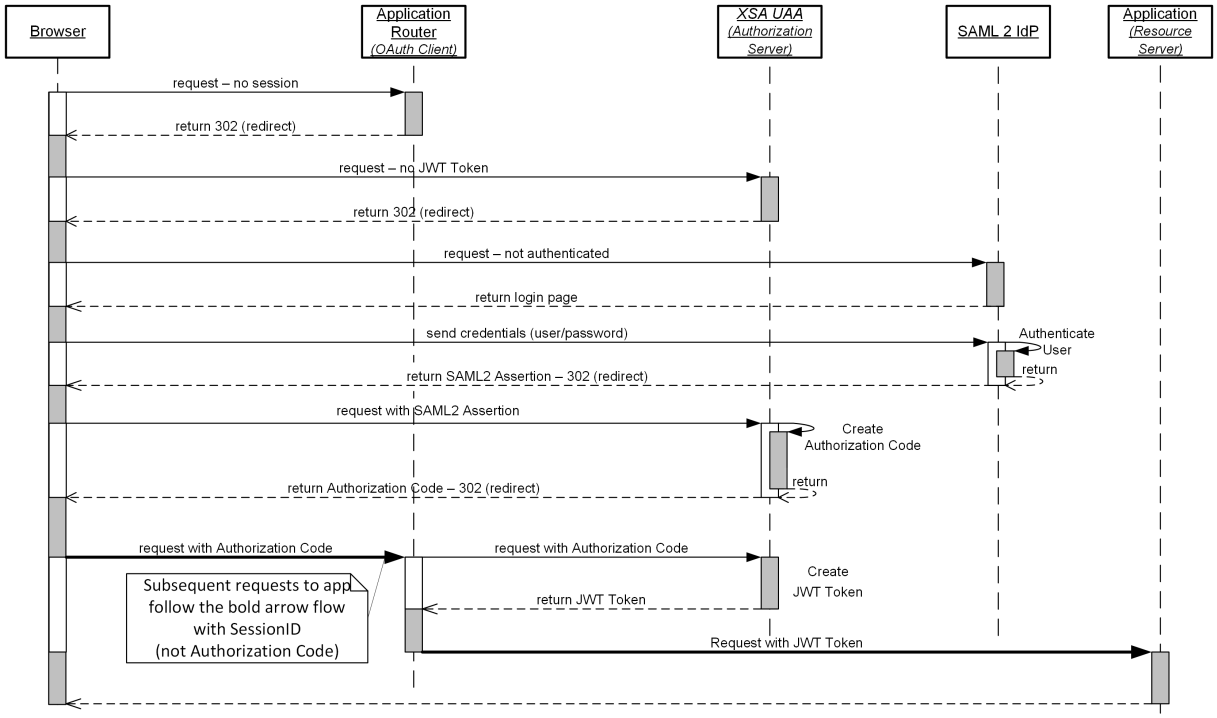
Today, the Internet enables us to use a huge variety of devices running multiple applications powered by different platforms. Despite the complexity, it is imperative that everything is able to interact securely, for example, so that logon is simplified and, as a result, security can be increased. This interaction requires protocols and technology that enable secure authentication and authorization between a wide variety of

components all over the Internet, for example: desktop devices, mobile devices, or Web-based applications. The problems caused by these competing requirements are addressed using the concept of Federated Identity, or distributed identity management.

Distributed Identity Management

Federated identity is the concept of linking and reusing electronic user identities across multiple identity-management systems and, as a result, freeing applications from the need to obtain and store a user´s credentials in order to be able to authenticate them on demand. Instead, the application can be configured to use an independent and central identity-management system that stores the user´s electronic identity, provided of course that the application is configured to trust the used identity-management system. This central identity-management system enables you to centralize authentication and authorization functionality. Federated Identity lead to the development of the following major protocols which are also supported by XS advanced applications:

- SAML 2.0 (authentication)
- OAuth 2.0 (authorization)



OAuth 2.0 Authentication and Authorization Flow in XS Advanced

Authentication with SAML 2.0

The Security Assertion Markup Language (SAML 2.0) is an XML-based open standard used to facilitate the exchange of authentication and authorization data of a principal (user) between an Identity Provider (IdP) and a Service Provider (SP). The exchange of date is performed with messages called "Bearer Assertions", where the "Bearer" is any party in possession of the SAML assertion. The integrity of the assertion is protected by means of XML encryption and an XML signature.

SAML addresses the requirement for Web browsers to enable and facilitate Single Sign-On (SSO) on the Internet. Authorizations are expressed through "User-Groups" to which the principal is assigned to. It is up

to the service provider and the respective implementation how the User-Groups are interpreted and the information translated into specific authorizations.

Authorization with OAuth 2.0

The OAuth 2.0 authorization framework is a protocol for delegating authorizations; it is not for use as an authentication mechanism. The OAuth 2.0 specifies the following basic roles:

- Resource Owner
  Typically, the end-user; the OAuth resource owner can grant access to a protected resource.
- Resource Server
  Typically a Cloud applications or microservice - hosts protected resources (REST endpoints). The OAuth resource server uses access tokens to accept (and respond to) requests for protected resources.
- Client
  Typically an `approuter`; an OAuth client is an application that sends requests for protected resources on behalf (and with the authorization) of the resource owner.
- Authorization server
  In XS advanced or Cloud Foundry, typically the XS User Account and Authentication (UAA) service. The OAuth Authorization Server issues access tokens for the OAuth client to obtain the authorizations of the OAuth resource owner after the client has been successfully authenticated by a SAML 2.0 compliant identity provider. An access token includes and represents the credentials used to access protected resources.

JSON Web Tokens

The JSON Web Token (JWT) is an open standard that defines a format that can be used for securely transmitting information between parties as a JSON object. The information included in the token can be verified and trusted because it is digitally signed. A JWT can be signed using a secret key (for example, using the HMAC algorithm) or a public-private key pair, for example, using the RSA algorithm, the de-facto standard for industrial-strength encryption.

In the context of XS advanced and CF, the JSON token is digitally signed with the private key of the Authorization Server (UAA). In addition to the header, the token contains information specifying the issuer, the subject, the expiration time, and any additional custom information required by the consumer.

In the context of XS advanced and SAP Cloud Platform (Cloud Foundry), developer writing business applications can validate the JWT token using the SAP Container Security API. You can also use the Security API to obtain user information, for example: logon name (`getLogonName`) or e-mail address (`getEmail`). The Security API also includes functions that enable you to check authorization scopes (`checkScope`), get the identity zone (`getIdentityZone`), get authentication attributes (`getAttribute`), and obtain tokens for use when authentication at the database is required (`getHdbToken`).

> → Tip
>
> The Security API is included in both the SAP Node.js package `@sap/xssec`, which is available on the public NPM Registry, and the Java library `java-container-security`, which is included in the `XS_JAVA` package that you can find on the SAP HANA media or download from SAP Marketplace.
>
> As of SPS05 the `java-container-security` library is deprecated and replaced by the open-source SAP Cloud Platform Java security client `java-security`, which is available on GitHub and on Maven Central. For more details, see *Related Information* below.

Related Information

XS Advanced Security Tools and Components [page 984]
Set up Authentication for an XS Advanced Application [page 965]
Set up Generic Authorization for an XS Advanced Application [page 969]
Standard Node.js Packages for XS Advanced [page 635]
SAP CP Java Security Library (GitHub)
SAP CP Spring XSUAA Library (GitHub)
Maven Central Repository

11.1.1 Authorization in SAP HANA XS Advanced

Authorization restricts access to resources and services based on defined user permissions.

SAP HANA XS advanced applications contain content that is deployed to different database containers. Access to the content requires not only user authentication but also the appropriate **authorization**.

The access-control process controlled by the authorization policy can be divided into the following two phases:

- Authorization
  Defined in the deployment security descriptor (`xs-security.json`), where access is authorized
- Policy enforcement
  The general rules by which requests for access to resources are either approved or disapproved.

Access enforcement is based on user identity and performed in the following distinct application components:

- Application router
  After successful authentication at the application router, the application router starts an authorization check (based on scopes).
- Application containers
  For authentication purposes, containers, for example the node.js and Java containers, receive an HTTP header `Authorization: Bearer <JWT token>` from the application router; the token contains details of the user and scope. This token **must** be validated using the XS advanced Container Security API. The validation checks are based on scopes and attributes defined in the XS advanced deployment security descriptor (`xs-security.json`).
- SAP HANA database
  Every XS advanced application uses its own dedicated database schema and its own dedicated (technical) database user for requests to connect to the database (for example, to manage persistent application data). This architecture ensures data isolation between the XS advanced applications. The technical database user is assigned the SAP HANA privileges required by the application; this defines the boundaries for all SAP HANA XS advanced users using that application.

> **i Note**
>
> Instance-based authorization checks using the data control language (DCL) are also supported; the checks are based on attributes, for example, those defined in the XS deployment security descriptor (`xs-security.json`).

Related Information

Maintaining Application Security in XS Advanced [page 957]
The Application Security Descriptor [page 996]
Scopes for Functional Authorization Checks [page 975]
The Data Control Language [page 976]
Set up Authentication for an XS Advanced Application [page 965]

11.1.2 OAuth 2.0 in XS Advanced

Since SAP HANA XS advanced uses OAuth 2.0 for authentication, the architecture needs to provide the OAuth 2.0 elements OAuth 2.0 client, OAuth 2.0 authorization server, and OAuth 2.0 resource server.

You want to integrate an application in the environment of XS advanced model. This means that the XS advanced platform needs to know your application. Using the User Account and Authentication (UAA) service of XS advanced, you initially integrate your application into the XS advanced platform environment. The application needs to authenticate against the User Account and Authentication service. The authentication concept XS advanced uses is OAuth 2.0.

In this context, the UAA acts as OAuth 2.0 authorization server. The application router itself is the OAuth 2.0 client. To integrate the application into XS advanced authentication, you must create a service instance of the `xsuaa` service and bind it to the application router and the application containers. From the OAuth 2.0 perspective, the containers (for example the node.js container) are OAuth 2.0 resource servers. Their container security API validates the access tokens against the UAA.

The UAA uses OAuth 2.0 access tokens with the JSON web token (JWT) format for authenticating at the containers, the OAuth 2.0 resource servers.

Principal propagation is needed in the SAP HANA database, as the business users are “virtual database users”, and the database connection uses a technical user. The SAP HANA database is a dedicated component that is called by SAP HANA XS advanced model. For using OAuth 2.0, the SAP HANA database needs to be able to receive and evaluate OAuth 2.0 access tokens. For this reason, the SAP HANA database uses SAML 2.0 bearer tokens.

Related Information

http://ietf.org/

11.1.3 Building an Authorization Concept for Users of an XS Advanced Application Instance

Business users in an SAP HANA XS advanced application should have different authorizations because they work in different jobs.

For example in the framework of a leave request process, there are employees who want to create and submit leave requests, managers who approve or reject, and payroll administrators who need to see all approved leave requests to calculate the leave reserve.

The authorization concept of a leave request application should cover the needs of these three employee groups. This authorization concept includes elements such as roles, scopes, and attributes.

Related Information

Example Scenario for Leave Request Authorizations [page 963]

11.1.4 Example Scenario for Leave Request Authorizations

You want to create roles and authorizations for the agents in a leave request work flow.

This work flow includes the following agents:

- Employee
- Manager (approver)
- Payroll administrator

Work Flow

The example used here to explain the different authorizations required for the different people in a leave-request scenario assumes the following work-flow:

Leave-Request Work Flow

Step	Action	Role
1	Create a leave request	Employee
2	Approve a leave request	Manager
3	Calculate available leave reserves	Payroll Administrator

Employee Authorizations

The employee John Doe needs the authorizations to enable him to create, edit, view, and delete his own requests for leave. He also needs to be able to delete an approved leave requests, for example, if due to a change of plans he decided to go to work on that day after all.

Employee Authorizations

Name	Role	Authorization	Attribute
John Doe	Employee	Create	Own
		Edit	Own
		View	Own
		Delete	Own

Manager Authorizations

John Doe's manager, Julia Moore, needs the authorizations that enable her to approve John's leave requests and edit them; she cannot, however, delete them. In addition, she needs the authorization that allows her (as a manager) to view, approve, and edit the leave requests of all employees in her department. Since she is also an employee, she needs to be able to create, view, edit, and delete her own leave requests.

Manager (Approver) Authorizations

Name	Role	Authorization	Attribute
Julia Moore	Employee	Create	Own
		Edit	Own
		View	Own
		Delete	Own
	Manager	Approve	Managed employees
		Edit	Managed employees
		View	Managed employees

Payroll Administrator Authorizations

A payroll administrators of the human resources department is only interested in viewing all approved leave requests. He needs the total number of approved leave request and of days off so that he can feed them into the payroll system, which will calculate the leave reserve. He is not involved in any way in the approval process.

Since the payroll administrator, Mark Smith, is an employee as well, he also needs the authorization to create, view, edit, and delete his own leave requests.

Payroll Administrator Authorizations

Name	Role	Authorization	Attribute
Mark Smith	Employee	Create	Own
		Edit	Own
		View	Own
		Delete	Own
	Payroll Administrator	View	All employees

As we can see, even a simple example such as the one described here involves a number of different authorizations defined in the corresponding attributes. However, the case could be extended further, for example, to enable the manager to view the leave of employees in related departments. With this additional authorization, the manager can see who is available as a substitute for the employees during their respective vacation.

An administrator in the XS advanced application can use role templates, scopes, and attributes to shape the roles so that they match the needs and requirements of your organization.

Related Information

Maintaining Application Security in XS Advanced [page 957]
Building an Authorization Concept for Users of an XS Advanced Application Instance [page 963]

11.2 Set up Authentication for an XS Advanced Application

Define the authentication model for your XS advanced application.

Prerequisites

- Ensure that Node.js and its package manager (NPM) is installed.
  - The XS advanced run time currently includes the following versions:
    Node.js 6.11.1
    NPM 3.10.10
  - For independent installations of Node.js or NPM:
    Open a command-line shell and run the following commands:
    ```
    node --version
    ```

```
npm --version
```

Context

In XS advanced, the Node.js application router (`approuter`) is used to connect your service to the centrally provided User Account and Authentication (UAA) service. This means that you need to deploy an `approuter` as part of your Multi-Target Application (MTA) and configure the `approuter` to manage the user authentication process.

Procedure

1. Set up the application router as part of your Multi-Target Application.

   The `approuter` is a Node.js package which you can download from the publicly available NPM Registry, for example by configuring the registry in your NPM resource-configuration file (`.npmrc`) as illustrated in the following example:

   ### Sample Code

   Sample NPM Registry Configuration (`.npmrc`)

   ```
   @sap:registry=https://registry.npmjs.org
   ```

   Add a dependency to the `approuter` package in your application's `package.json` file.

   ### Sample Code

   Typical `approuter` declaration in the `package.json`

   ```
   {
     "name": "approuter",
     "dependencies": {
        "@sap/approuter": "2.10.0"
     },
     "scripts": {
        "start": "node node_modules/@sap/approuter/approuter.js"
     },
     "engines": {
        "node": "6.11.1"
     }
   }
   ```

2. Update the application's `manifest.yml` file.

   Since the `approuter` is a Node.js application, it must be added to the XS advanced application's build and deployment manifest in the same way as any other application. This list and configuration is specified in the `manifest.yml` file, as illustrated in the following example:

   The `manifest.yml` file is formatted according to YAML rules, which requires strict and consistent adherence to indents. In the following example, the `<tenantID>` is only required if the application is deployed in a multi-tenant scenario.

Sample Code

```
applications:
- name: approuter
  host: approuter-<tenantID>
  path: <PATH>/approuter
  buildpack: nodejs_buildpack
  memory: 128M
  env:
    XSAPPNAME: myApp-<tenantID>
    TENANT_HOST_PATTERN: "^(.*)-approuter-<tenantID>.acme.com"
    destinations: >
      [
        {
         "name":"ads-destination",
         "url":"https://MyApp-ads-<tenantID>.acme.com",
         "forwardAuthToken": true
        }
      ]
  services:
    - applogs-myApp
    - uaa-myApp
```

Note

Since the `manifest.yml` contains multiple applications, it is necessary to specify the name of the `"host"` where the application is running. With multiple hosts, it is not possible to use other means to specify the host name, for example, during application deployment using the command `xs push -n <HOST>`.

3. Configure the application router.

   You must define `"routes"` for an application; the specified path is used to direct incoming requests to the correct destination, as illustrated in the following example:

   Sample Code

   Route Configuration in `xs-app.json`

   ```
   {
     "welcomeFile": "index.html",
     "routes": [{
       "source": "^/myApp",
       "target": "/",
       "destination": "myApp-destination"
     }]
   }
   ```

4. Deploy the application router and the corresponding application to the desired target.

   Before deploying the application, it is necessary to ensure that a instance of the UAA service is available for use by the deployed application.

   → Tip

   For `[xs|cf]` in the following steps, use either the `cf` command (for SAP Cloud Platform/CF environments) or the `xs` command (for on-premise environments).

a. Create the UAA service instance for your application.

```
[xs|cf] create-service xsuaa application uaa-myApp -c
'{"xsappname":"myApp"}'
```

b. Deploy your application.

```
[xs|cf] push
```

> **i** Note
>
> The host names for the deployment targets are specified in the application manifest (`manifest.yml`).

c. Check the information available for the UAA service connection information in the `<VCAP_SERVICES>` environment variable.

The UAA service broker generates the service-connection information when the `approuter` is bound to the corresponding UAA service instance. The JSON array following the string `"XSUAA":` contains the client credentials for the UAA service instance, which are used to establish a trusted relationship between the `approuter` and the UAA service. The client credentials string also contains the URL of the UAA service, which the `approuter` uses to redirect unauthenticated calls to the UAA service.

```
[xs|cf] env approuter
```

> **i** Note
>
> Locate the property `xsappname`.

d. List the routes that are already mapped to the application.

```
[xs|cf] routes
```

If the `approuter` is to be used for other tenants, you must specify a route for each additional tenant, for example, to `myAppTrial`.

5. Access the application using the application router.

a. List the URL for the `approuter`.

```
[xs|cf] apps
```

b. Paste the URL for the `approuter` into a Web browser.

`https://myAppTrial-approuter.acme.com`

This should redirect you to XS UAA logon screen, where you can log on using the credentials required (for example, e-mail address and password). Both the authentication **method** (`"route"`, or `"none"`) and the authentication **type** (`"xsuaa"`, `"basic"`, or `"none"`) can be specified in the application router configuration (`xs-app.json`).

> **→** Tip
>
> The application router can be configured to implement Cross-Site Request Forgery (CSRF) protection. It is also possible to configure the `approuter` to initiate an automatic central log out if a user session is inactive for a specified period of time. The amount of time to wait is defined in the `SESSION_TIMEOUT` environment variable. To prevent unexpected behavior, you must ensure

that the timeout configuration for the application router and the XS UAA service are identical for **all** routes with the authentication type `"xsuaa"`.

Related Information

The NPM Registry [page 687]
XS Advanced Application Router Configuration Syntax [page 922]
Set up Authentication for your XS Advanced JavaScript Application [page 704]
Set up Authentication for XS Advanced Java Applications [page 754]
Set up Generic Authorization for an XS Advanced Application [page 969]

11.3 Set up Generic Authorization for an XS Advanced Application

Define an authorization model for your XS advanced application and configure generic authorization for any application end point (route path).

Prerequisites

The following files are required:

- `xs-app.json`
  Application descriptor (application-router configuration file)
- `xs-security.json`
  Security descriptor (security configuration file)

Context

To enable user access to an XS advanced application, it is necessary to declare a "start authorization" for the application's security descriptor (`xs-security.json`) and reference the start authorization from the application router's (`approuter`) configuration file (`xs-app.json`). This enables the XS application's application router to perform a generic authorization check on the declared start authorization. The authorization itself is based not only on scopes, which determine the type and extent of access allowed (for example, view, edit, manage), but also on roles, which are defined in templates. For example, the role template defines the authorization scopes that are associated with a given role. Users can then be assigned to a role that corresponds to the type of access authorization that they need.

To set up authorization for your XS advanced application, perform the following steps:

Procedure

1. Define authorization scopes and role templates.

   Define the role templates that specify the authorization scopes for each role. Users can then be assigned to one or more roles which ensure that the specified scopes are assigned as authorization keys.

   To declare the role templates, modify the security descriptor file `xs-security.json` by including the following authorization model for your application:

   Sample Code

   ```
   {
     "xsappname"     : "myApp-tenantID",
     "description"   : "Enable myApp for multi tenants",
     "tenant-mode"   : "shared",
     "scopes"        : [
                        {
                          "name"             : "$XSAPPNAME.Display",
                          "description"      : "Display Info."
                        },
                        {
                          "name"             : "$XSAPPNAME.Update",
                          "description"      : "Update Info."
                        }
                       ],
     "role-templates": [
                        {
                          "name"             : "Viewer",
                          "description"      : "View Info.",
                          "scope-references" : [
                                                "$XSAPPNAME.Display"
                                               ]
                        },
                        {
                          "name"             : "Editor",
                          "description"      : "Maintain Info",
                          "scope-references" : [ "$XSAPPNAME.Display",
                                                 "$XSAPPNAME.Update"
                                               ]
                        }
     ]
   }
   ```

   > **Note**
   >
   > For multi-tenant scenarios, the value of `"xsappname"` must be unique within the whole Cloud Foundry Organization, which is why the tenant ID here ensures that the value for `"xsappname"` refers to this unique instance. The shared `"tenant-mode"` makes it possible for the XS advanced UAA service instance to trust other tenants in addition to the one specified here.

2. Configure start conditions for application routes and end points.

   To define the start condition, we need to assign the required scope definition to the application's route(s). For example, in the approuter configuration file (`xs-app.json`), add an entry (`"scope:"`) that

references the authorization scope (`"Display"`) previously defined in the application's security descriptor `xs-security.json`, as illustrated in the following example:

> Sample Code

```
{
  "welcomeFile": "index.html",
  "routes": [{
     "source": "^/foo/api/",
     "target": "/api/",
     "destination": "foo-destination",
     "scope": "$XSAPPNAME.Display"
     },
     {
     "source": "^/foo",
     "target": "/",
     "destination": "foo-destination"
     }]
}
```

3. Inform the XS advanced User Account and Authentication (UAA) service of the security updates.

   The XS UAA service needs to know about any changes to the application's authorization model, which is described in the security descriptor (`xs-security.json`).

   > → Tip
   >
   > For `xs|cf` in the following steps, use either the `cf` command (for SAP Cloud Platform/CF environments) or the `xs` command (for on-premise environments).

   ```
   [xs|cf] update-service <myServiceInstance> -c <path>/xs-security.json
   ```

4. Bind your application to the XS UAA service.

   Add a service entry to the application's deployment manifest `manifest.yml`.

   > i Note
   >
   > The `manifest.yml` file is formatted according to YAML rules, which requires strict and consistent adherence to indents.

   > Sample Code

   ```
   - name: myApp
     ...
     services:
       - ...
       - uaa-myApp
   ```

5. Deploy the application.

   ```
   [xs|cf] push
   ```

6. Maintain the roles and authorizations defined in your application's security model.

   At this point, that, the XSUAA is aware of the role-templates defined in the application's security descriptor (`xs-security.json`). However, since the required scopes and roles are not yet assigned to a user, the user does not yet have the permissions required to access the advertised application end points.

In this step, you need to create a new role based on the role-template specified in the application's `xs-security.json` file; add the role to a **role collection** (either existing or newly created); and assign a user to the role collection. Logging into the application as the user with the assigned role collection ensures that the configured authorizations are assigned to the logged in user.

> → Tip
>
> You can set up roles and role collections, and assign users to role collections using the XS Advanced Admin Tools (on-premise) or the SAP Cloud Platform Cockpit (Cloud Foundry). Access to these tools requires administrator privileges.

Related Information

Application Security Descriptor Configuration Syntax [page 999]
XS Advanced Application Router Configuration Syntax [page 922]
The XS Command-Line Interface Reference
Set up Authorization for your XS Advanced JavaScript Application [page 708]
Set up Authorization for XS Advanced Java Applications [page 754]
Building Roles for XS Advanced Applications (Administration Guide)

11.3.1 Application Authorization Tasks, Roles, and Tools in XS Advanced

An overview of the steps required to set up user authorization for XS advanced applications.

Developers store authorization information as design-time role templates in the security descriptor file `xs-security.json`. Using the `xsuaa` service broker, they deploy the security information in a dedicated XS advanced application. The XS advanced administrators view the authorization information in role templates, which they use as part of the run-time configuration. The administrators use the role templates to build roles, which are aggregated in role collections. The role collections are assigned, in turn, to business users.

The tasks required to set up authorization artifacts in SAP HANA XS advanced are performed by two distinct user roles: the application developer and the SAP HANA administrator. After the deployment of the authorization artifacts as role templates, the administrator of the SAP HANA XS advanced application uses the role templates provided by the developers to build role collections and assign them to business users, for example, using in the *SAP HANA XS Advanced Cockpit* section of the *SAP HANA Administration Guide*.

> ℹ Note
>
> To test authorization artifacts after deployment, developers can use the role templates to build role collections and assign authorization to business users in the *SAP HANA XS Advanced Cockpit*.

Setting Up Authorization Artifacts

Step	Task	User Role	Tool
1	Specify the security descriptor file containing the functional authorization scopes for your application	Application developer	Text editor
2	Create role templates for the XS advanced application using the security descriptor file	Application developer	Text editor
3	Create a service instance from the `xsuaa` service in XS advanced using the service broker	Application developer	XS advanced CLI tool
4	Bind the service instance to the XS advanced application by including it into the manifest file	Application developer	Text editor
5	Deploy the XS advanced application	Application developer	XS advanced CLI tool
6	If required, create a new role in the XS advanced application role builder using role templates	XS advanced administrator	Application role builder
7	Create a role collection and assign roles to it	XS advanced administrator	Application role builder
8	Assign the role collection to a SAML 2.0 identity provider or to SAP HANA database users	XS advanced administrator	Application role builder, and SAML IDP Tool
9	Assign the users to roles using the role collections	XS advanced administrator	User interface of XS Advanced

Related Information

Set up Generic Authorization for an XS Advanced Application [page 969]
XS Advanced Security Tools and Components [page 984]
Building Roles for XS Advanced Applications (Administration Guide)
Maintaining the XS Advanced Run-time Environment (Administration Guide)

11.4 Assign Functional Authorization Scopes for an XS Advanced Application

Declare authorization scopes for your application in the security descriptor and then assign the security descriptor file to the application.

Context

The functional authorization scopes for your application must be known to the User Account and Authentication (UAA) service. To do this, you declare the scopes of your application in the security descriptor file named `xs-security.json` and then you assign the security descriptor file to the application.

Once you have created an OAuth 2.0 client using the service broker, you specify the `xs-security.json` file that is relevant for your application. Using this security descriptor file, the OAuth 2.0 client that has been created for your application receives permission to access the scopes of your application automatically.

Procedure

1. Define the authorization scopes for your application.

   Authorization scopes for XS advanced applications are defined in the application's security descriptor (`xs-security.json`). You also need to provide a reference for each authorization scope in a corresponding role template. The following example of an `xs-security.json` file defines scopes for viewing and creating data:

   Sample Code

   Authorization Scopes in the Security Descriptor (`xs-security.json`)

   ```
   {
     "xsappname": "tinyworld",
     "scopes": [ { "name": "$XSAPPNAME.view", "description": "View data" },
                 { "name": "$XSAPPNAME.create", "description": "Create data"}
               ],
     "role-templates": [ { "name": "tinyworldView",
                           "description": "Role for viewing data",
                           "scope-references": [ "$XSAPPNAME.view" ]
                         },
                         { "name": "tinyworldCreate",
                           "description": "Role for creating data",
                           "scope-references":
   [ "$XSAPPNAME.create","$XSAPPNAME.view" ]
                         }
                       ]
   }
   ```

2. Assign the `xs-security.json` file to the XS advanced application for which you defined the authorization scopes.

You can use the following command to :

```
xs create-service <service_name> <service_plan> -c <security_descriptor_file>
```

> **Example**
>
> ```
> xs create-service xsuaa default -c xs-security.json
> ```

This creates an instance of the UAA service with the scopes defined in the security descriptor and ensures it is bound to the corresponding application.

> **Note**
>
> After the developer has created the role templates and deployed them to the relevant application, it is the administrator's task to use the role templates to build roles, aggregate the roles into role collections, and assign the role collections to business users in the application. For more information, see the section about maintaining the XS advanced run-time environment in the *SAP HANA Administration Guide*.

Related Information

Create the Security Descriptor for an XS Advanced Application [page 995]

11.4.1 Scopes for Functional Authorization Checks

Authorization in the application router and container are handled by scopes. Scopes are groupings of functional organizations defined by the application.

From a technical point of view, the resource server (the relevant security container API) provides a set of services (the resources) for functional authorizations. The functional authorizations are organized in scopes.

Scopes are used to define the authorizations required by business users for access to the features provided by a specific SAP HANA XS advanced application, for example: view, change, or delete. Scopes are created when you deploy or update the application. Scopes are defined in an application's security descriptor file `xs-security.json`. It is possible to define so-called "local" (application-specific) scopes and, if necessary, also any "foreign" scopes, which are valid for other applications, too.

> **→ Tip**
>
> Scopes can be checked declaratively or programatically. For example, a declarative check of the scopes can be performed by the application router or in the Java run-time container; a programmatic check can be performed in either the Java or Node.js run-time containers.

The security descriptor is also used to define a set of role templates for a named application; the role templates contain the authorizations ("`scope-references`") for business users' actions, for example: viewing, editing, or deleting data. Information retrieved from the user's identity (such as location, department, or cost center) is stored in attributes.

After the developer has created the role templates and deployed them to the relevant application, it is the administrator's task to use the role templates to build roles, aggregate roles into role collections, and assign the role collections to business users in the application.

To assign scopes to an application, you need to perform the following steps:

1. Create an instance of the User Account and Authentication (UAA) service.
   You can use the service broker to create an instance of the UAA service. This ensures the creation of a new OAuth 2.0 client in the UAA.
2. Assign scopes for the application.
   Scopes are configured and assigned in the application descriptor file `xs-security.json`, which can be placed in the root folder of the application or in a `security` folder in your XS advanced project along with a `package.json` file.

> → Tip
>
> For access to SAP HANA objects, the privileges of the technical user (container owner) apply; for business data, instance-based authorizations using DCL are used.

The following example, of a simple security descriptor for an XS advanced application shows how to define authorization scopes and reference them in role templates:

> Sample Code

```
{
  "xsappname": "tinyworld",
  "scopes": [ { "name": "$XSAPPNAME.view", "description": "View data" },
              { "name": "$XSAPPNAME.create", "description": "Create data"}
            ],
  "role-templates": [ { "name": "tinyworldView",
                        "description": "Role for viewing data",
                        "scope-references": [ "$XSAPPNAME.view" ] },
                      { "name": "tinyworldCreate",
                        "description": "Role for creating data",
                        "scope-references":
[ "$XSAPPNAME.create","$XSAPPNAME.view" ] }
                    ]
}
```

Related Information

The Application Security Descriptor [page 996]

11.4.2 The Data Control Language

Control access to data with an instance-based authorization model.

The Data Control Language (DCL) is the language used in Core Data Services (CDS) to define instance-based authorization models which control access to data based on user attributes, for example: a user name, an organizational unit, or a country. Attributes can be defined by applications, and it is possible to map SAML 2.0

attributes with SAP HANA XS advanced attributes. To use them in the XS advanced application, the developers need to deploy the artifacts to the XS advanced application.

CDS DCL is designed as an extension to SQL's DCL; among other things, DCL introduces the notion of predicated privileges that allow dynamic grants based on the attribute values of entity instances. DCL uses **rules** to restrict access to data, as illustrated in the following example:

Sample Code

Access Rules in a Role with CDS DCL

```
role salesOrderOrgUnit {
  grant select on ddl.salesOrderView
    where customerOrgUnit in $env.user.orgunit; };
```

XS advanced applications declare attributes ("`orgunit`" in the example above) which they need for instance-based authorizations, so that only users registered in the named organizational unit are allowed (when authenticated) to access the information. These attributes take the form `$env.user.<attribute name>` and are integrated in rules using the syntax `customerOrgUnit in $env.user.orgunit`. The attributes are obtained from the Attribute Provider Service during the authentication process and passed to the SAP HANA database in a JSON Web Token (JWT).

Note

In XS advanced, this authorization concept coexists with the authorization concept based on role templates deployed in XS advanced applications. You can use the concepts together or independently of another. The application-based authorization concept is based on role templates. The authorization check of a business user occurs on the XS advanced application instance.

Multi-Value Attributes in XS Advanced Model

With earlier releases of SAP HANA, for example, SAP HANA 1.0, it is possible to define and consume user attributes with single values within DCL, for example, in grants and rules; the defined attributes are transferred in a JWT token (or a SAML assertion in older releases) to the SAP HANA database and stored in the session context. With SAP HANA 2.0 it is possible to use attributes that have multiple values. In both cases the syntax required to use the attributes within a grant (or rule) is illustrated in the following example:

Sample Code

User Attribute in a SELECT Rule

```
role salesOrderCountryOwn {
  grant select on ddl.salesOrderView
    where customerCountry in $env.user.country; };
```

The notation defining the attribute on the right-hand side of the IN condition always takes the following form: `$env.user.<attribute name>.` So, the attribute name in the example above is "`country`", and any user with the role `salesOrderCountryOwn` will be able to see the entries of the `salesOrderView` that correspond to the country of his country attribute within the session context.

> ! Restriction
>
> With SAP HANA 2.0, it is not possible to use the `SESSION_CONTEXT` syntax to access attribute values in SAP HANA CDS DCL. In an SQL artifact, however, you **can** use `SESSION_CONTEXT` to extract attribute values, for example, when defining a structured privileges or inserting an instance filter into an SQL view definition.

Although the following example is syntactically correct and can be activated, it is not allowed in CDS DCL rules in SAP HANA 2.0; the rule does not work at run time when the data is selected.

> Sample Code
>
> SESSION_CONTEXT not allowed in SAP HANA 2.0 CDS DCL Rules

```
role salesOrderCountryOwn {
  grant select on ddl.salesOrderView
    where customerCountry in SESSION_CONTEXT('XS_COUNTRY'); };
```

Related Information

Create a CDS Access-Policy Document (XS Advanced) [page 439]

11.5 Set up Application Security in XS Advanced

Help ensure an XS advanced application is protected from Web-based attacks.

Context

In addition to enabling an application to identify, authenticate, and grant the appropriate authorization to users logging on to an XS advanced application, it is also essential to ensure that the application itself uses all available tools to guard against Web-based attacks, for example, by setting a session timeout and closing unused and unwanted sessions; using CORS to enable or disable the shared access to resources between domains; enabling protection against cross-site request forgery (CSRF); and configuring protection against click-jacking attempts.

The following steps provides useful tips and tricks to help you understand what needs to be done to set up application security and where to find the information to help you do it:

Procedure

1. Set up user authentication for your XS advanced application.

   You can choose between various authentication methods and types for users logging into an XS advanced application. It is also recommended to set up the OAuth 2.0 client.

   - Authentication method and type
     The **method** used to authenticate the user of an XS advanced application defined in the application-router configuration file (`xs-app.json`), as illustrated in the following (incomplete) example:
     `"authenticationMethod": ["route"|"none"]`

     > **i** Note
     >
     > If `authenticationMethod` is set to "`none`", logon with the User Account and Authentication (UAA) service is disabled.

     The **type** of authentication used for user logon to an XS advanced application is defined in the individual route specified in the application-router configuration file (`xs-app.json`), as illustrated in the following (incomplete) example:
     `"routes": [{...},{[...]"authenticationType": ["xsuaa"|"basic"|"none"]}]`

     > **i** Note
     >
     > If `authenticationType` is set to "`none`", then no authentication is required for the specified application route.

   - OAuth 2.0 configuration
     The OAuth 2.0 client configuration is defined in the XS advanced application's security descriptor (`xs-security.json`), as illustrated in the following (incomplete) example:
     `"oauth2-configuration": {"token-validity": "900", "autoapprove": "false"}`
   - Single Sign-on (SSO)
     The XS advanced run-time platform supports Single Sign-On (SSO) with SAP HANA OpenID Connect (OIDC), for example, using X.509 certificates or SPNego with Kerberos, provided that the corresponding back-end services are configured and enabled in SAP HANA. The XS advanced UAA server enables support for X.509 certificates by default (`uaa.oidc.enablex509 = true`); support for SPNego/Kerberos on the other hand is not enabled by default (`uaa.oidc.enablespnego = false`) but can be enabled in the XS UAA server configuration file (`xsuaaserver.ini`) manually, if required.

     > **i** Note
     >
     > XS administrator privileges are required to maintain security certificates in the XS advanced run-time; SAP HANA administrator privileges are required to maintain database users.

     For more information about setting up SSO for SAP HANA users and XS advanced applications, see the section *Maintaining Application Run-Time Services* in the *SAP HANA Administration Guide* in *Related Information* below.

2. Set up cross-origin resource sharing (CORS).

   The CORS environment variable enables you to provide support for cross-origin requests, for example, by allowing the modification of the request header, which is normally disabled by default. Cross-origin resource sharing (CORS) permits Web pages from other domains to make HTTP requests to your application domain, where normally such requests would automatically be refused by the Web browser's security policy.

   In XS advanced, you can enable cross-origin support in the application router by setting the `<CORS>` environment variable.

   ⁖≡ Sample Code

   ```
   xs set-env <AppName> CORS {<CORS_Configuration>}
   ```

   When setting the CORS environment variable, the properties `"uriPattern"` and `"allowedOrigin"` are mandatory.

   → Tip

   If you use a simple comma-separated list or a JSON structure to define `<CORS_Configuration>`, the rules that apply for escaping quotes or spaces included in the string differ according to the command shell you are using (Windows or Unix-like). The following example shows a Unix example using single quotes (`'{}'`) to enclose the CORS configuration values. For more information, see the `xs set-env` command in the *XS Command-Line Interface Reference* listed in *Related Information*.

   ⁖≡ Sample Code

   ```
   xs set-env myApp1 CORS '{"uriPattern": "^\route1$", "allowedOrigin":
   [ { "host": "www.acme.com", "protocol": "https", "port": 345 } ]}'
   ```

3. Enable protection against cross-site request forgery (CSRF).

   In the application router configuration, you can specify if a route needs protection for CSRF tokens. The default value is “true” where the application router enforces CSRF protection for any HTTP request that changes state on the server side, for example: `PUT`, `POST`, or `DELETE`.

   → Tip

   The SAP Node.js package `@sap/xss-secure` is a utility that provides additional protection against cross-site scripting as described in a subsequent step.

   The following example shows how to enable CSRF protection for an application route. For more information about configuring the application router in XS advanced, see the *Related Information*.

   ⁖≡ Sample Code

   ```
   {
   ...
     "routes": [
         {
           "source": "^/sap/ui5/1(.*)$",
           "target": "$1",
           "destination": "ui5",
           "csrfProtection": true
   ```

```
      },
...
}
```

4. Set up protection against click-jacking attempts.

   In the application router configuration, you can enable an allowlist service to help prevent click-jacking attacks. Enabling the service opens an endpoint that accepts `GET` requests at the relative path configured in the `endpoint` property, as illustrated in the following example:

   Sample Code

   ```
   {
     "whitelistService": {
       "endpoint": "/allowlist/service"
     }
   }
   ```

   If the service is enabled in the application router, each time an HTML page needs to be rendered in a frame, the service is used to check if the parent frame is allowed to render the content in another frame. For more information about configuring the application router, see *Related Information*.

   Tip

   You can use the `<CJ_PROTECT_WHITELIST>` environment variable to define a list of allowed host or domain names that do not need to be protected against click-jacking attacks.

   The following example shows how to add a host name to the application router's allowlist to ensure protection against click-jacking:

   Sample Code

   ```
   xs set-env myApp1 CJ_PROTECT_WHITELIST {<protocol>, <hostname>, <portNr>}
   ```

5. Set up protection against cross-site scripting (XSS).

   Cross-site Scripting (XSS) is the name of a class of security vulnerabilities that can occur in Web applications, for example, when these applications dynamically generate HTML, JavaScript, or even CSS. Attackers can use these vulnerabilities to include arbitrary HTML, JavaScript, or CSS into an application's client front end; this code is, in turn, rendered by the victim's Web browser and, in this way, interpreted in the victim's current authentication context. To help you protect your application against XSS attacks, SAP provides the Node.js package `@sap/xss-secure` which includes APIs to encode CSS, HTML, JavaScript, URLs, and even XML code.

   The following example shows how to start using `@sap/xss-secure`:

   Sample Code

   ```
   var xsssecure = require('@sap/xss-secure');
   ```

   For more information about standard SAP Node.js packages, see *Related Information*.

6. Set up timeouts for application user sessions.

   You can configure the triggering of an automatic central log-out from the User Account and Authentication (UAA) service if an application session is inactive for a specified time. A session is considered to be inactive

if no requests are sent to the application router. The following command shows how to set the environment variable `<SESSION_TIMEOUT>` to 10 (ten) minutes for the application `<myApp>`:

> Sample Code

```
xs set-env <myApp> SESSION_TIMEOUT 10
```

> → Tip
>
> If the authentication type for a route is set to `"xsuaa"` (for example, `"authenticationType": "xsuaa"`), the application router depends on the UAA server for user authentication, and the UAA server might have its own session timeout defined. To avoid problems caused by unexpected timeouts, it is recommended that the session timeout values configured for the application router and the UAA are identical."

7. Set up anti-virus scanning for XS advanced applications.

   The `@sap/node-vsi` package contains the Virus Scan Interface (VSI) binding for Node.js. The package also includes the native libraries required to run on Windows, Linux, and MacOS operating systems.

   > i Note
   >
   > The `@sap/node-vsi` package does not perform any virus scanning; the virus-scanning tools must be installed and configured independently.

   The following example shows how to start using `@sap/node-vsi`:

   > Sample Code

   ```
   var vsi = require('@sap/node-vsi');
   ```

   For more information about standard SAP Node.js packages, see *Related Information*.

8. Secure your Node.js application in XS advanced.

   SAP HANA includes a selection of standard Node.js packages that provide help when enabling and configuring security for your XS advanced application. The packages are available for download and use either from the public NPM registry at `https://registry.npmjs.org` or, for customers and partners who have the appropriate access authorization, from the SAP Service Marketplace (SMP).

   The following security-related Node.js/JavaScript libraries are available:

   - `@sap/audit-logging`
     A Node.js client that enables applications to make use of the features and functionality provided by the XS advanced audit-log service
   - `@sap/hdbext`
     A JavaScript client for Node.js which implements the SAP HANA Database SQL Command Network Protocol
   - `@sap/xsenv`
     A utility that allows the easy setup of (and access to) XS Advanced environment variables and services (for example, CORS, CSRF, ...)
   - `@sap/xssec`
     XS advanced security API for Node.js; this includes the XS advanced **container** security API for Node.js described in the *SAP HANA Administration Guide*

- `@sap/xss-secure`
  A utility that provides protection against cross-site scripting

> → Tip
>
> For more information about which Node.js packages are available, see *Related Information* below. For more details about the contents of an individual package, see the `README` file in the corresponding package after download.

9. Secure your Java application in XS advanced.

   SAP HANA includes a selection of standard Java libraries, which are available for download from the SAP Service Marketplace (SMP) to customers and partners who have the appropriate access authorization and from the Maven Central repository. The following security-related Java libraries are included:

   - `xs-env`
     A utility for setting up and accessing XS advanced environment variables and services
   - `audit-java-client-api`
     A client that enables applications to use the XS advanced audit-log service
   - `java-container-security`
     A convenient Java wrapper for the security context of an authentication

     > → Tip
     >
     > As of SPS05 the Java library `java-container-security` is deprecated and replaced by the open-source SAP Cloud Platform Java security client `java-security` or `spring-xsuaa`, both of which are available on GitHub and on Maven Central. For more details, see *Related Information* below.

   For more information, see *Related Information* below.

   > → Tip
   >
   > For more details of the contents of an individual package, see the `README` file in the corresponding package after download or, if available, the JavaDoc reference guide.

10. Secure your Python application.

    SAP HANA includes a selection of standard Python libraries, which are available for download from the SAP Service Marketplace (SMP) to customers and partners who have the appropriate access authorization. The following security-related Python libraries are included:

    - `sap_xssec`
      XS Advanced Container Security API for Python.
    - `sap_audit_logging`
      Audit-logging functionality for XS Advanced Python applications

11. Configure data privacy and anonymity in views.

    You can use so-called view nodes in calculation views to ensure the anonymity of private data in result sets. The graphical calculation view editor in the SAP HANA Web IDE provides an **anonymize** view node, which you can use to select the required anonymization method and to define the level of privacy for a data set. The “anonymize” view node must always be used as a leaf node in the calculation view. You can preview the output of the calculation view modeled with anonymize view nodes only if the query on the calculation view includes all output columns of the calculation view.

→ Tip

For more information about data-anonymity in calculation views, see the *SAP HANA Modeling Guide for SAP Web IDE for SAP HANA* in *Related Information*.

Related Information

XS Advanced Security Tools and Components [page 984]
XS Advanced Application Router Configuration Syntax [page 922]
XS Advanced Application Router Environment Variables [page 938]
Standard Node.js Packages for XS Advanced [page 635]
Standard Java Client Libraries for XS Advanced [page 720]
Tutorial: Setting up your JavaScript Application in XS Advanced [page 699]
Tutorials: Setting Up the Java Run-Time Environment in XS Advanced [page 735]
The XS Command-Line Interface Reference
SAP HANA Modeling Guide for SAP Web IDE for SAP HANA (Data Anonymization)
Standard Python Packages for XS Advanced [page 783]
SAP CP Java Security Library (GitHub)
SAP CP Spring XSUAA Library (GitHub)
Maven Central Repository

11.5.1 XS Advanced Security Tools and Components

Setting up security for XS advanced applications involves multiple tasks and multiple tools and components.

The following table lists the tasks you need to perform to set up security for your XS advanced application, shows which tools or components are involved, and indicates the privileges and permissions required by the persona who typically performs the task, for example, "developer" or "administrator".

ℹ Note

The tasks and tools listed in the following table are not exhaustive; some tools are not included. The aim of the table is to provide guidance concerning where to find information about the more typical areas of concern for application security. For more information about the individual tools mentioned in this topic, see *Related Information*.

Security-Related Tasks and Tools in XS Advanced

Task	Tool	Persona/Permissions
Application container: access and security	Container Security API • JavaScript/Node.js: `@sap/xssec` • Java: `java-container-security` • Python: `sap_xssec`	Developer
Application Security	Java: • Spring Security • `web.xml` (standard non-proprietary authentication method) • `@ServletSecurity` restraint (standard non-proprietary method) • `audit-java-client-api` (included in SAP's XS_JAVA package) • `java-container-security` (included in SAP's XS_JAVA package) **Note** As of SPS05 the Java library `java-container-security` is deprecated and replaced by the open-source SAP Cloud Platform Java security client `java-security`, which is available on GitHub and on Maven Central. For more details, see *Related Information* below.	Developer
	JavaScript/Node.js: • `@sap/xssec` (security APIs) • `@sap/xss-secure` (cross-site scripting) • `@sap/node-vsi` (anti-virus) **Note** The `@sap/node-vsi` package does not perform any virus scanning; the virus-scanning tools must be installed and configured independently.	Developer
	Python • `sap_xssec` (Security API; included in SAP's `XS PYTHON package`) • `sap_audit_logging` (included in SAP's `XS PYTHON package`)	Developer

Task	Tool	Persona/Permissions
Application authentication (method, type, ...)	Application router configuration (`xs-app.json`)	Developer
	OAuth 2.0 configuration (`xs-security.json`)	Developer
	JSON Web Tokens (JWT)	Developer
	User Account and authentication service (**UAA**) XS CLI: • `create-service` • `bind-service`	Developer: during development and testing XS advanced administrator: for installation and update on productive systems
	Single Sign-on (SSO) with SAP HANA OpenID Connect (OIDC). The default XS advanced UAA server configuration (`xsuaaserver.ini`) enables support for X.509 certificates (`uaa.oidc.enablex509 = true`). Support for SPNego/Kerberos on the other hand is not enabled by default (`uaa.oidc.enablespnego = false`) but can be enabled in the XS UAA server configuration file manually, if required. **i Note** For more information, see the section *Maintaining Application Run-Time Services* in the *SAP HANA Administration Guide* in *Related Information* below.	XS advanced administrator: maintaining run-time certificates SAP HANA administrator: maintaining database users
Application authorization (scopes)	Application security descriptor (`xs-security.json`) Application router configuration (`xs-app.json`)	Developer
Application routes	Application router configuration (`xs-app.json`)	Developer
Application destinations (login/logout)	Application router configuration (`xs-app.json`) Application deployment manifest (`manifest.yml`) **i Note** Destinations specified in the `manifest.yml` must match the destinations defined the application-router configuration (`xs-app.json`). For more information, see the section application routes and destinations.	Developer

Task	Tool	Persona/Permissions

Click Jacking protection	Application router configuration (`xs-app.json`):
	`"whitelistService": {"endpoint":"<path>"}`
	Application router environment variable:
	`CJ_PROTECT_WHITELIST {<protocol>, <hostname>, <portNr>}`
		Developer

Cross-Origin Resource Sharing (CORS) configuration	Application router environment variable (`<CORS>`), for example: `[{"uriPattern": "^\route1$", "allowedMethods": ["GET"], "allowedOrigin": [ { "host": "host.acme.com", "protocol": "https", "port": 345 }] }]` → Tip CORS configuration is also possible in the application `manifest.yml` file. For more information, see *Related Information* below.	Developer
Cross-Site Resource Forgery (CSRF) protection	Application router configuration (`xs-app.json`)	Developer
Cross-site scripting (XSS) protection	Node.js package `@sap/xss-secure`	Developer
Data anonymity	Calculation views with the *k-anonymity* (and differential privacy) anonymization methods. → Tip For more information, see the *SAP HANA Modeling Guide for SAP Web IDE for SAP HANA*.	Developer

Task	Tool	Persona/Permissions
Secure Store	An application bound to an instance of the XS advanced platform services `hana` or `managed-hana` with the service plan `securestore` can use the following stored procedures to access to the SAP HANA Secure Store: • `SYS.USER_SECURESTORE_INSERT` Insert an encrypted entry into the SAP HANA Secure Store. • `SYS.USER_SECURESTORE_RETRIEVE` Retrieve an encrypted entry from the SAP HANA Secure Store. • `SYS.USER_SECURESTORE_DELETE` Delete an encrypted entry from the SAP HANA Secure Store.	Developer
User identification	User accounts and subaccounts (SAP Cloud Platform Cockpit) *SAML ID Provider Configuration* (XS Advanced Admin Tools)	SAP Cloud Platform administrator XS advanced administrator
User roles	Application security descriptor (`xs-security.json`) • Role Templates • Scopes/Attributes	Developer
User role collections	*Application Role Builder* (XS advanced tools) SAP Cloud Platform Cockpit	XS advanced administrator SAP Cloud Platform administrator
User organizations/spaces	XS CLI: • `create-[org|space]` • `set-[org|space]-role` • `create-role-collection` • `assign-role-collection`	orgManager spaceManager
	XS administrator tools: • *Organization and Space Management* • *Application Role Builder* • *SAML ID Provider Configuration* • *User Management*	XS advanced administrator

Task	Tool	Persona/Permissions
Virus protection	Anti-virus scanning configuration in Node.js package `@sap/node-vsi` **i Note** The `@sap/node-vsi` package does not perform any virus scanning; the virus-scanning tools must be installed and configured independently.	Developer

Related Information

The XS Advanced Security Descriptor (xs-security.json) [page 996]
XS Advanced Application Router Environment Variables [page 938]
The XS Advanced Application Descriptor (xs-app.json) [page 917]
Download and Consume SAP Node.js Packages [page 633]
Download and Consume Java Libraries [page 719]
Download and Consume Python Libraries [page 782]
Creating Data Persistence Artifacts with CDS in XS Advanced [page 310]
SAP HANA Administration Guide (Application Run-Time Services)
The XS Command-Line Interface Reference
SAP HANA Modeling Guide for XS Advanced (Data Anonymization)
SAP HANA and HDI Services in XS Advanced [page 834]
Maintain Values in the SAP HANA Secure Store [page 1020]
SAP CP Java Security Library (GitHub)
SAP CP Spring XSUAA Library (GitHub)
Maven Central Repository

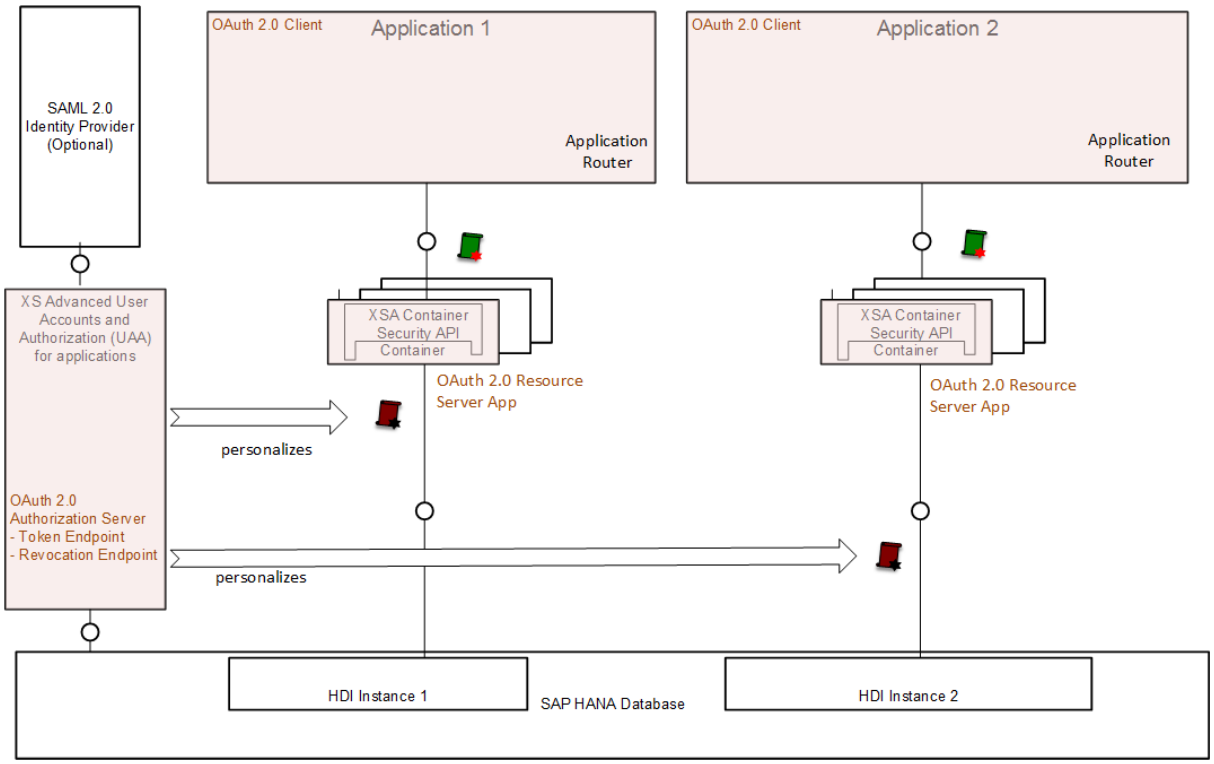
11.5.2 Personalizing the Database Connection with JSON Web Tokens

Secure the technical-user connection between an XS advanced application and the SAP HANA database.

XS Advanced applications contain content which is deployed to different containers. Access to the content requires user identification and authentication as well as the appropriate authorization. Additional security can be built into the business logic code (Java or JavaScript) and the SQL connection to the database.

Connections to the SAP HANA database are established with the credentials of the technical user provided by the service instance. When the connection personalization feature is enabled, the connections to the database are still established using the credentials of the technical user, however, in addition, the JWT token of the application business user is propagated to the SAP HANA database layer. In this way the user-specific SAP

HANA privileges of the business user (if any are configured at the database level) are applied, which provides the possibility of limiting access to the database based on the authorizations of the business user.



As illustrated in the diagram above, a JSON Web Token (JWT) can be used to secure the technical-user connection between an XS advanced application and the SAP HANA database. After it is set, the JWT can be consumed at the database level using the Data Control Language (DCL), for example, with a user attribute such as `$env.user.<AttributeName>`, or by accessing the propagated session attributes using plain SQL.

> → Tip
>
> The database JWT token can be retrieved using the standard security library provided with XS advanced model and set for all supported XS advanced model run-time environments, for example, JavaScript/ Node.js, Java, or Python.

Every XS advanced application has the option of declaring user attributes in the `attributes` section of the application's security descriptor, `xs-security.json`. A user attribute is a key-value pair, which is used by the application to control and restrict access, for example, when used in a DCL aspect that restricts access to data depending on attribute values. Attribute values can also be retrieved using the XS advanced Container Security API, for example, `SecurityContext.getUserInfo()`. The Attribute Provider Service can extract the user attributes from a JWT received during authentication (if available), a local persistence artifact, or a combination of both. The retrieved attributes might be processed further, for example, renaming or mapping, recombination, derivation, and so on.

The following table shows the high-level steps required in the end-to-end process of using a JSON Web Token (JWT) to personalize the technical-user connection between an XS advanced application and the SAP HANA database:

Secure the Database Connection with a JSON Web Token

Step	Action	User/Role
1	Retrieve the database JWT token by means of the XS advanced security library	Developer
2	Set a JWT for the database connection for the run-time environment	Developer
3	Consume the JWT at the database level, for example, in DCL with `SESSION_CONTEXT('XS_*')` or in plain SQL with a JSON query	Developer
4	Pass attributes from the Identity Provider (IdP) down to the database via this JWT	Administrator

Retrieve the Database JSON Web Token

The XS advanced security API provides the method "`getHdbToken`" which returns a token that can be used, for example, to help improve the security of the technical-user connections to the SAP HANA database.

Sample Code

Using the `getHdbToken` Method

```
function getRequestOptions(req) {
  var opt = { session: {} };
  var authInfo = req.authInfo;
  if (authInfo && authInfo.getGrantType && authInfo.getGrantType() !==
internalConsts.GRANTTYPE_CLIENTCREDENTIAL) {
    if (authInfo.getHdbToken) { opt.session.XS_APPLICATIONUSER =
authInfo.getHdbToken();
    }
  }
}
```

Note

Note that if the returned token used to instantiate the security context is a foreign token (the OAuth client contained in the token and the OAuth client of the current application do not match), null is returned instead of a token.

Set a JWT for the Database Connection

You can use a JSON Web Token to secure the technical-user connection between an XS advanced application and the SAP HANA database. One important prerequisite is that a trusted relationship exists between the User

Account and Authentication (UAA) service and the SAP HANA database. More specifically, this means that the UAA is registered as trusted JWT provider inside SAP HANA database; this is true by default for XS advanced run time in on-premise scenarios, but requires some manual work in Cloud environments.

The method used to set the Web token depends on the context and the programming language used for your XS advanced application, as illustrated in the following short examples:

- SQL
  `SET XS_APPLICATIONUSER = <JWT TOKEN>`
- `@sap/hdbext`
  The Node.js package for database connections sets the JWT automatically.
- `@sap/hana-client`
  With the `connection.session.XS_APPLICATIONUSER = <JWT TOKEN>;`
- For Java applications, you use can activate the personalization feature to set `XS_APPLICATIONUSER` automatically to the JWT, as follows:

  > **i Note**
  >
  > The feature only works for protected users; public resources do not have a user context.

  - For Apache Tomcat use the `META-INF/context.xml` file.

    ```
    <?xml version='1.0' encoding='utf-8'?>
    <Context>
    <Resource id="jdbc/my-hdi-container" provider="xs.openejb:XS Default JDBC
    Database" type="javax.sql.DataSource" >
        service=my-hdi-container
        personalize=true
    </Resource>
    ```

  - For Apache TomEE use the `WEB-INF/resources.xml` file.

    ```
    <?xml version='1.0' encoding='utf-8'?>
    <resources>
        <Resource id="jdbc/my-hdi-container" provider="xs.openejb:XS Default
    JDBC Database" type="javax.sql.DataSource" >
            service=my-hdi-container
            personalize=true
        </Resource>
    </resources>
    ```

- In the SQL console in the Database Explorer in Web IDE for SAP HANA
  Check the option *Set XS_APPLICATIONUSER*

  > **! Restriction**
  >
  > This feature is currently only available for use in SAP HANA on-premise scenarios.

Consume the JWT at the Database Level

To better understand the context in which the JSON Web Token used to personalize the database connection works, it is important to see how user attributes are represented at the database, for example:

- As session variables (`<XS_*>`)

- As variables in a JSON array: `"['us', 'de', 'en']"`

The information set in the session variables (for example, `XS_APPLICATIONUSER` or `$env.user.value`) can be extracted using SQL, for example, and used in other design-time artifacts, as shown in the following list:

- In an SQL view (`hdbview`) artifact:

  Sample Code

  `v_items.hdbview`

  ```
  VIEW V_ITEMS
  AS
  SELECT ID, NAME, OWNER
  FROM T_ITEMS
  WHERE OWNER = SESSION_CONTEXT('XS_APPLICATIONUSER')
  ```

- In an SAP HANA CDS DCL artifact:

  Sample Code

  `dcl.hdbcds`

  ```
  using "V_T_ITEMS" as V_T_ITEMS;
  accesspolicy dcl {
     role R_V_T_ITEMS {
         grant
             select on V_T_ITEMS
         where
             A in $env.user.value;
     };
  };
  ```

- The `dcl.hdbcds` artifact renders the `where` condition as follows in plain SQL:

  Sample Code

  ```
  WHERE  ( "V_T_ITEMS"."A" )  IN  (  ( SELECT "ALIAS"."VALUE" FROM
  JSON_TABLE  (  (  ( '{"values" : ' || SESSION_CONTEXT ( 'XS_VALUE' )  )
  || '}' )  , '$.values[*]' COLUMNS  ( "VALUE" VARCHAR  ( 5000 )  PATH
  '$' )  )  AS "ALIAS" )  )  )  )  )
  ```

Pass Attributes from the Identity Provider (IdP) to the Database

To read attributes from a JWT token, you can use the functions and libraries provided with the XS advanced container security API:

- Node.js: `@sap/xssec`
- Java: `java-container-security`

  → Tip

  As of SPS05 the `java-container-security` library is deprecated and replaced by the open-source SAP Cloud Platform Java security client `java-security`, which is available on GitHub and on Maven Central. For more details, see *Related Information* below.

- Python: `sap_xssec`

To configure attributes in role collections, you can define them in an application's security descriptor (`xs-security.json`), as illustrated in the example below. Role templates can be used to build roles, which can be assigned to role collections, for which attributes can also be defined.

Sample Code

Example `xs-security.json` File

```
{
  "xsappname" : "node-hello-world",
  "scopes"     : [ {
                    "name" : "$XSAPPNAME.Display",
                    "description" : "display" },
                  {
                    "name" : "$XSAPPNAME.Edit",
                    "description" : "edit" },
                  {
                    "name" : "$XSAPPNAME.Delete",
                    "description" : "delete"  }
                ],
  "attributes" : [ {
                    "name" : "Country",
                    "description" : "Country",
                    "valueType" : "string" },
                  {
                    "name" : "CostCenter",
                    "description" : "CostCenter",
                    "valueType" : "int" }
                ],
  "role-templates": [ {
                       "name"                : "Viewer",
                       "description"         : "View all books",
                       "scope-references"    : [
                                               "$XSAPPNAME.Display" ],
                       "attribute-references": [ "Country" ]
                      },
                      {
                       "name"                : "Editor",
                       "description"         : "Edit, delete books",
                       "scope-references"    : [
                                               "$XSAPPNAME.Edit",
                                               "$XSAPPNAME.Delete" ],
                       "attribute-references" : [
                                                 "Country",
                                                 "CostCenter"]
                      }
                     ]
}
```

Note

You can also define attributes in the Identity Provider (IdP), for example, while configuring the UAA's trust configuration for an application using the *Identity Authentication Administration Console*. For each application's IdP trust configuration, you can select from a list of assertion attributes and configure as required.

Related Information

Create the Security Descriptor for an XS Advanced Application [page 995]
SAP HDI Artifact Types and Build Plug-ins Reference
Download and Consume SAP Node.js Packages [page 633]
Download and Consume Java Libraries [page 719]
Download and Consume Python Libraries [page 782]
SAP CP Java Security Library (GitHub)
SAP CP Spring XSUAA Library (GitHub)
Maven Central Repository

11.6 Create the Security Descriptor for an XS Advanced Application

The security descriptor defines details of an application's security-related dependencies.

Prerequisites

- File name
  `xs-security.json`
- Location
  Either of the following folders:
  - Application root folder, for example, `/path/appname/`
  - Application security folder, for example, `/path/appname/security`
- Format
  JavaScript Object Notation (JSON)

Context

The `xs-security.json` file defines the security and deployment options for an application; the information in the application-security file is used at application-deployment time, for example, for authentication and authorization.

Procedure

1. Create the application resource-file structure.

For example, `/path/appname/`

2. Create a sub-folder for the JavaScript security files.

   The security sub-folder should be located in the application root folder, for example, `/path/appname/security`.

3. Create the application security file.

   The application security file `xs-security.json` can be located either in the application root folder `/path/appname/` or the security folder `/path/appname/security`.

4. Define details of the application's security-related dependencies.

   The contents of the `xs-security.json` file must adhere to the standards defined in the XS advanced application's JSON-formatted security-descriptor syntax.

Related Information

The Application Security Descriptor [page 996]
Application Security Descriptor Configuration Syntax [page 999]

11.6.1 The Application Security Descriptor

A file that defines the details of the authentication methods and authorization types to use for access to your application.

The `xs-security.json` file uses JSON notation to define the security options for an application; the information in the application-security file is used at application-deployment time, for example, for authentication and authorization. Applications can perform scope checks (functional authorization checks with Boolean result) and checks on attribute values (instance-based authorization checks). The `xs-security.json` file declares the scopes and attributes on which to perform these checks. This enables the User Account and Authentication (UAA) service to limit the size of an application-specific JSON Web Token (JWT) by adding only relevant content.

Scope checks can be performed by the application router (declarative) and by the application itself (programmatic, for example, using the container API). Checks using attribute values can be performed by the application (using the container API) and on the database level using the data control language SAP HANA DCL.

The contents of the `xs-security.json` are used to configure the OAuth 2.0 client; the configuration is shared by all components of an SAP multi-target application. The contents of the `xs-security.json` file cover the following areas:

- Authorization scopes
  A list of limitations regarding privileges and permissions and the areas to which they apply
- Attributes
  A list of as-yet undefined information or sources (for example, the name of a cost center)
- Role templates
  A description of one or more roles to apply to a user and any attributes that apply to the roles

The following example shows a simple application-security file:

Sample Code

Example `xs-security.json` File

```
{
  "xsappname" : "node-hello-world",
  "scopes"     : [ {
                    "name" : "$XSAPPNAME.Display",
                    "description" : "display" },
                  {
                    "name" : "$XSAPPNAME.Edit",
                    "description" : "edit" },
                  {
                    "name" : "$XSAPPNAME.Delete",
                    "description" : "delete"  }
                ],
  "attributes" : [ {
                    "name" : "Country",
                    "description" : "Country",
                    "valueType" : "string" },
                  {
                    "name" : "CostCenter",
                    "description" : "CostCenter",
                    "valueType" : "int" }
                ],
  "role-templates": [ {
                       "name"                : "Viewer",
                       "description"         : "View all books",
                       "scope-references"    : [
                                             "$XSAPPNAME.Display" ],
                       "attribute-references": [ "Country" ]
                      },
                      { "name"               : "Editor",
                        "description"        : "Edit, delete books",
                        "scope-references"   : [
                                               "$XSAPPNAME.Edit",
                                               "$XSAPPNAME.Delete" ],
                        "attribute-references" : [
                                               "Country",
                                               "CostCenter"]
                      }
                     ]
}
```

Scopes

Scopes are functional authorizations that are assigned to users by means of security roles, which are mapped to the user group(s) to which the user belongs. Scopes are used for authorization checks by the application router and the application's HDI container. The authorization checks can be performed declaratively (for example, by the application router or in the Java run-time container) or programmatically, for example, in either the Java or the Node.js run-time containers.

- Application router
  URL prefix patterns can be defined and a scope associated with each pattern. If one or more matching patterns are found when a URL is requested, the application router checks whether the OAuth 2.0 access token included in the request contains at least the scopes associated with the matching patterns. If not all scopes are found, access to the requested URL is denied. Otherwise, access to the URL is granted.

- Application container
  The container security API includes the `hasScope` method that allows you to programmatically check whether the OAuth 2.0 access token used for the current request has the appropriate scope. Among other things, the access token contains a list of scopes that the user is allowed to access in the current context. Each scope name must be unique in the context of the current XS advanced UAA installation.

Attributes

You can define attributes so that you can perform checks based on a source that is not yet defined. Attributes are used to define **instance**-based authorizations, which change according to the context, for example, location, time-zone, or cost center. In the example `xs-security.json` file included here, the check is based on a cost center, whose name is not known at design time, because the name of the cost center differs according to context.

Role Templates

A role template describes a role and any attributes that apply to the role.

It is important to remember that a role template must be instantiated; you can do this with the role-building tools provided in the *XS Administration Tools*. This is especially true with regards to any attributes defined in the role template and their concrete values, which are subject to customization and, as a result, cannot be provided automatically. Role templates that only contain local scopes can be instantiated without user interaction. The same is also true for foreign scopes where the scope "owner" has declared his consent in a kind of allow list (for example, either for "public" use or for known "friends").

> **i** Note
>
> If a role template does not contain any attributes, then a role is generated automatically with the same name as the role template. Otherwise, a role based on a role template can be generated using the XS administration tools for XS advanced.

The version information defined in the role template can be used to check if an instantiated role needs to be updated. This assumes that the instantiated role contains not only a reference to the role template from which it was derived but also the corresponding version information.

> **i** Note
>
> The resulting application-specific role instance needs to be assigned to one or more user groups.

Related Information

Create the Security Descriptor for an XS Advanced Application [page 995]
Application Security Descriptor Configuration Syntax [page 999]

11.6.2 Application Security Descriptor Configuration Syntax

The syntax required to set the properties and values defined in the `xs-security.json` application-security description file.

→ Tip

The following code-syntax example is for illustration purposes only; it contains links to further information. Click an attribute, for example, `"scopes"` to navigate to the section where the selected attribute is described in more detail.

Sample Code

XS Advanced Security Descriptor (`xs-security.json`)

```
{
  "xsappname" : "node-hello-world",
  "tenant-mode"   : "shared",
  "scopes"     : [ {
                     "name" : "$XSAPPNAME.Edit",
                     "description" : "edit",
                     "granted-apps":
["$XSAPPNAME","$XSAPPNAME(application,zone-guid-1,$XSAPPNAME)"],
                     "grant-as-authority-to-apps" : ["$XSAPPNAME"]}
                 ],
  "foreign-scope-references": ["$ACCEPT_GRANTED_SCOPES"],
  "authorities":["$ACCEPT_GRANTED_AUTHORITIES"],
  "attributes" : [ {
                     "name" : "Country",
                     "description" : "Country",
                     "valueType" : "string" },
                   {
                     "name" : "CostCenter",
                     "description" : "CostCenter",
                     "valueType" : "string" }
                 ],
  "role-templates": [ {
                     "name"               : "Editor",
                     "description"        : "Edit, delete books",
                     "scope-references"   : [ "$XSAPPNAME.Edit",
                                              "$XSAPPNAME.Delete" ],
                     "attribute-references" : ["Country", "CostCenter"] }
                    ],
  "oauth2-configuration": {
                     "token-validity": 900,
                     "refresh-token-validity": 1800,
                     "redirect-uris": [
                                       "https://<host_name1>",
                                       "https://<host_name2>"],
                     "autoapprove": "false",
                     "system-attributes": ["groups", "rolecollections"],
                     "allowedproviders": ["useridp1", "useridp2"] }
}
```

xsappname

Use the `xsappname` property to specify the name of the SAP HANA XS application(s) to which the security description applies.

Sample Code

```
"xsappname" "<app-name>",
```

The `xsappname` property represents a developer-defined part of a system-generated OAuth Client ID. Since an OAuth Client ID must be unique within a given subaccount (pka "identity zone"), the `xsappname` property must be defined in such a way as to guarantee the uniqueness of the corresponding OAuth Client ID.

→ Tip

It is important to ensure that the name defined in `xsappname` can be easily recognized by other users when configuring additional security components, for example, when XS advanced administrators are using the SAP HANA XS Advanced Admin Tools to create roles from role templates. See also *Unsupported Application Names* below for a list of application names that are not permitted, for example, "zones" or "oauth".

Naming Conventions

Bear in mind the following restrictions regarding the length and content of an application name in the `xs-security.json` file:

- The following characters can be used in an XS advanced application name: “aA”-“zZ”, “0”-“9”, “-” (dash), “_” (underscore), “/” (forward slash), and “\” (backslash)
- The maximum length of an XS advanced application name is 100 characters.

Since the `xsappname` that is written into the credentials section of the environment must be unique within the XS advanced or Cloud Foundry landscape where the application is deployed, it is possible to combine multiple service plans for the same application. The result is a list of “xsappnames” with different suffixes that correspond to the various service plans used.

If the application `myApp` is deployed once with the service plan `"default"`, twice with service plan `"space"`, and three times with service plan `"devuser"`, the resulting “xsappnames” for `myApp` might look like the names included in the following list:

- Service plan `"default"`:
  - myApp
- Service plan `"space"`:
  - myApp**!i5**
  - myApp**!i11**
- Service plan `"devuser"`:
  - myApp**!u5**
  - myApp**!u8**
  - myApp**!u9**

You can also include the other Cloud-Foundry-related plan: `application`. The suffix added to the has the `xsappname` has the format: `"!b<tenant index>"`, where `<tenant index>` is a number that the UAA maintains internally and differs from landscape to landscape, for example, `myapp!i8`.

⚠ Caution

It is not recommended to rely on the application-name format used in the examples above, for example, by parsing code in order to retrieve the service plan used with a particular application instance. However, the

general format of the application name is relevant, for example, when using the *Application Role Builder* tool to maintain application roles in the XS Advanced Cockpit.

Unsupported Application Names

It is not permitted to use any of the following names in the `xsappname` property:

- "zones"
- "clients"
- "scim"
- "password"
- "oauth"
- "approvals"
- "groups"
- "uaa"
- "sap_system"

References to the Application Identity

You can use any of the following methods to refer to the application identity (appid) defined in the `xsappname` property:

- `$XSAPPNAME`
  Used for references to the `"xsappname"` specified in the local `xs-security.json` security-descriptor file
- `$XSAPPNAME(<plan>,<xsappname>)`
  Used for references to the specified `"xsappname"` in the subaccount of the current application

  ! Restriction

  For use in Cloud-Foundry environments only; the XSUAA service plan must be a PaaS-enabled plan, for example, “application”.

tenant-mode

In multi-tenant environments, tenants subscribe to and consume applications, which are registered as clients at the XS UAA. XS UAA creates a new OAuth2 client per application for each tenant. You can use the custom `tenant-mode` property to define the way the tenant's OAuth clients obtain their respective client secrets.

During the binding of the `xsuaa` service, the service broker writes the specified tenant mode into the service-broker credentials section. The application router uses the tenant-mode information for the implementation of multi-tenancy with the application's bound service plan.

Sample Code

```
{
  "xsappname"   : "<application_name>",
  "tenant-mode" : "shared",
  "scopes"      : [
                   {
                    "name"         : "$XSAPPNAME.Display",
                    "description"  : "display" }
```

```
                    ],
[...]
}
```

The following tenant modes are available:

Tenant Modes

Value	Description
`dedicated`	The OAuth client receives a separate client secret for each subaccount. (Default) If no value is specified for `tenant-mode`, the default value is assumed.
`shared`	The OAuth client uses the same client secret in all subaccounts that subscribe to the application configured with this tenant mode. The `shared` tenant mode is mandatory for an application router configured for multi-tenancy applications, and the application router variable `TENANT_HOST_PATTERN` and a corresponding route must be mapped. → Tip The `shared` tenant mode is used by the UAA `application` service plan intended for use with PaaS-enabled applications and in combination with Cloud Foundry resources.
`external`	For tenants with multiple subscriptions to applications. For each subscription to an application the tenant receives an OAuth client with a client secret.

scopes

In the application-security file (`xs-security.json`), the `scopes` property defines an array of one or more security scopes that apply for an application. You can define multiple `scopes`; each scope has a name and a short description. The list of scopes defined in the `xs-security.json` file is used to authorize the OAuth client of the XS application with the correct set of **local** and **foreign** scopes; that is, the permissions the application requires to be able to respond to all requests.

Sample Code

Defining Local Scopes in the Security Descriptor

```
"scopes": [
    {
     "name" : "$XSAPPNAME.Display",
     "description" : "display" },
    {
     "name" : "$XSAPPNAME.Edit",
     "description" : "edit" },
    {
     "name" : "$XSAPPNAME.Delete",
     "description" : "delete"  }
],
```

Scopes are mostly “local”; that is, application-specific. Local scopes are checked by the application's own application router or checked programmatically within the application's run-time container. In the event that an application needs access to other SAP HANA XS advanced services on behalf of the current user, the provided access token needs to contain the necessary “foreign” scopes. Foreign scopes are not provided by the application itself; they are checked by other sources outside the context of the application. For more information about how and where to use foreign scopes, see `foreign-scope-references` and `authorities`.

In the `xs-security.json` file, “local” scopes must be prefixed with the variable `<$XSAPPNAME>`; at run time, the variable is replaced with the name of the corresponding local application name. Using a static application name could lead to deployment problems where the name's index suffix changes depending on the landscape. For more information, see *References to Application Identity* in the property "xsappname" [page 999] above.

> → Tip
>
> The variable `<$XSAPPNAME>` is defined in the application's deployment manifest description (`mtad.yaml` or `manifest.yml`).

The following table lists the parameters and values that can be used with the `scopes` property:

Parameters and Values for "scopes"

Key	Mandatory	Description	Example
`name`	Yes	Name of the local scope, which must be prefixed with the name of the XS advanced application defined in the property `"xsappname"`. It is highly recommended to prefix the scope name with the variable $XSAPPNAME in order to reference the element "xsappname" of the `xs-security.json` file. The `"name"` can contain up to 193 characters and must not start with a period '.' (dot), and cannot include any unsupported characters. Permitted characters include: 'a'-'z', 'A'-'Z', '0'-'9', '_', '-', '\', '/', ':', and '.'	`"$XSAPPNAME.LRApprove"`
`description`	Yes	A short summary of the specified scope; `"description"` must not be longer than 1000 characters.	

Key	Mandatory	Description	Example
`granted-apps`	No	Array of application-id values. It is highly recommended to use the `$XSAPPNAME` functions to reference the `appid` value of other XS advanced or Cloud Foundry applications. The value '*' is allowed if the scope is intended to be granted to all applications, and for this reason should only be used with extreme care. `granted-apps` is only relevant for user scenarios. **! Restriction** Value '*' does not work if the receiving application defines `"$ACCEPT_GRANTED_SCOPES"` with the property `"foreign-scope-references"` in its `xs-security.json` file.	`["LR2","$XSAPPNAME(application,LR3)"],` `["*"],`
`grant-as-authority-to-apps`	No	Array of application-id values. It is highly recommended to use the `$XSAPPNAME` functions to reference the application Id value of other XS advanced or Cloud Foundry applications. `grant-as-authority-to-apps` is only relevant for client-credential scenarios. **! Restriction** Value '*' is not permitted.	`["LR2","LR3"],`

i Note

The use of scope-granting features establish a strong dependency between the involved applications and should be avoided wherever possible. Instead of using the granting functionality, try to see if the features provided by the XSUAA broker plan supports your application's authorization needs in a better way.

Granting Authorization Scopes to other Applications

If two XS advanced applications are bound to the User Account and Authentication (UAA) service, an authorization scope defined in one application can be granted to the second application on request. The application requesting the scope authority must specify this acceptance (of externally defined scopes) in the

`"authorities"` section of its own security descriptor, and the application granting the scope authority must flag the specified scope as “grantable” in its own security descriptor, as illustrated in the following example:

Sample Code

Defining Local Scopes in the Security Descriptor

```
"scopes": [
    {
     "name" : "$XSAPPNAME.Display",
     "description" : "display" },
    {
     "name" : "$XSAPPNAME.Edit",
     "description" : "edit" },
    {
     "name" : "$XSAPPNAME.Delete",
     "description" : "delete"  },
    {
     "name" : "$XSAPPNAME.ForeignCall",
     "description" : "Enable calls into scope-granting app",
     "granted-apps": [*],
     "grant-as-authority-to-apps" : ["RequestingApp"]
    }
],
```

Naming Conventions

Bear in mind the following restrictions regarding the length and content of a scope name:

- The following characters can be used in a scope name: “aA”-“zZ”, “0”-“9”, “-” (dash), “_” (underscore), “/” (forward slash), “\” (backslash), “:” (colon), and the “.” (dot)
- Scope names cannot start with a leading dot “.” (for example, `.myScopeName1`)
- The maximum length of a scope name, including the fully qualified application name, is 193 characters.

! Restriction

Some scope names are already in use; these names are specified in a default deny list.

It is not permitted to use any of the following unsupported names in the `scopes` property:

- "zones.read", "zones.write"
- "clients.admin", "clients.write", "clients.secret"
- "scim.write", "scim.read", "scim.create", "scim.userids", "scim.zones"
- "password.write"
- "oauth.approval", "oauth.login"
- "approvals.me"
- "groups.update"
- "uaa.resource", "uaa.admin", "uaa.none"

attributes

In the application-security file (`xs-security.json`), the `attributes` property enables you to define an array listing one or more attributes that are applied to the role templates also defined in the `xs-security.json` file. You can define multiple `attributes`.

Sample Code

Defining Attributes in the XS Advanced Application's Security Descriptor

```
"attributes" : [
     {
      "name" : "Country",
      "description" : "Country",
      "valueType" : "string" },
     {
      "name" : "CostCenter",
      "description" : "Cost Center ID",
      "valueType" : "string" }
],
```

The `"attributes"` property is only relevant for user scenarios. Each element of the attributes array defines an individual, named, attribute, which can be referenced by one or more role-templates. For this reason, if the user in this scenario has been assigned the corresponding roles, the defined attributes are added to the JSON Web Token and used to enable instance-based authorization checks for access to the SAP HANA database. The value of the attributes is assigned in the administrator's SAP HANA or SAP Cloud Cockpit, either by defining a static value or by referencing an attribute of the SAML assertion that is issued by the configured identity provider during authentication.

The `attributes` key can take the following parameters and values:

Parameters and Values for Authorization-Scope "attributes"

Key	Description	Example
`name`	The name of the attribute with a value to apply when building the role template	`Country` `Costcenter`
`description`	A short summary of the attribute defined	`Country` `Cost Center ID`
`valueType`	The type of value expected for the defined attribute; possible values are: “`string`”, “`int`” (integer), or “`date`”	`int`

Naming Conventions

Bear in mind the following restrictions regarding the length and content of attribute names in the `xs-security.json` file:

- The following characters can be used to declare an attribute name in the `xs-security.json` file for XS advanced: “aA”-“zZ”, “0”-“9”, “_” (underscore)
- The maximum length of an XS advanced security attribute name is 64 characters.

role-templates

In the application-security file (`xs-security.json`), the `role-templates` property enables you to define an array listing one or more roles (with corresponding scopes and any required attributes), which are required to access a particular application module. You can define multiple `role-templates`, each with its own scopes and attributes.

> ! Restriction
>
> It is not permitted to use the property `role-templates` in combination with the XS UAA `broker` "clone" service plan. For more information see *Service Plans and Resources in XS Advanced* in *Related Information*.

> Sample Code
>
> Defining Role Templates in the XS Advanced Application's Security Descriptor

```
"role-templates": [
     {
      "name"                : "Editor",
      "description"         : "View, edit, delete books",
      "scope-references"    : [
                              "$XSAPPNAME.Edit", "$XSAPPNAME.Delete"
                              ],
      "attribute-references": [
                              "Country", "CostCenter"
                              ]
     },
]
```

A role template must be instantiated. This is especially true with regards to any attributes defined in the role template and the concrete attribute values, which are subject to customization and, as a result, cannot be provided automatically. Role templates that only contain "local" scopes can be instantiated without user interaction. The same is also true for "foreign" scopes where the scope **owner** has declared consent in a kind of allow list (for example, either for "public" use or for known "friends").

> i Note
>
> The resulting (application-specific) role instances need to be assigned to the appropriate user groups.

Role-Template Parameters

Key	Description	Example
`name`	The name of the role to build from the role template	`"Viewer"`
`description`	A short summary of the role to build	`"View all books"`
`scope-references`	The security (authorization) **scope** to apply to the application-related role	`"$XSAPPNAME.Display"`
`attribute-references`	One or more attributes to apply to the built role	`["Country", "CostCenter"]`

Naming Conventions

Bear in mind the following restrictions regarding the length and content of role-template names in the `xs-security.json` file:

- The following characters can be used to declare the name of a role-template in the `xs-security.json` file in XS advanced: “aA”-“zZ”, “0”-“9”, “_” (underscore)
- The maximum length of an XS advanced role-template name is 64 characters.

foreign-scope-references

The element `"foreign-scope-references"` defines whether an application accepts authorization scopes that where granted to it by another XS advanced or Cloud Foundry application. Only those scopes that are listed as `"foreign-scope-references"` will be accepted by an application; the accepted scopes are then added to the local application's own OAuth client. If the value `$ACCEPT_GRANTED_SCOPES` is used, the local application will accept all scopes that are granted to it by other applications.

> **i** Note
>
> It is not permitted to use the property `"foreign-scope-references"` in combination with the XS UAA `broker` “clone” service plan. For more information see *Service Plans and Resources in XS Advanced* in *Related Information*.

The element `"foreign-scope-references"` is only relevant for a user scenario where the roles are not defined in the application's own service instance. For example, an administrator application integrates different application components, each of which has its own `xs-security.json` security descriptor. The administrator application uses the values defined in `"foreign-scope-references"` in the application's security descriptor to request the specified scopes, while the roles are defined by the components.

The following code illustrates a simple example of the `"foreign-scope-references"` property as defined in an application's security descriptor:

> Sample Code

```
"foreign-scope-references": ["$ACCEPT_GRANTED_SCOPES"]
```

```
"foreign-scope-references": ["$XSAPPNAME(application,otherApp).scope1"]
```

The `“foreign-scope-references”` property specifies a list of scopes, with the following rules and restrictions:

- It is highly recommended to prefix the scope value with the `$XSAPPNAME` function to reference the application Id inside the scope value, as illustrated in the example above.
- References to a scope of the local XS advanced or Cloud Foundry application are not permitted
- `$ACCEPT_GRANTED_SCOPES` is an allowed value with a special meaning.
  If the value`$ACCEPT_GRANTED_SCOPES` is used, the local application where it is specified accepts **all** scopes that are granted to it by other (foreign) applications. Only those scopes listed in `"foreign-scope-references"` are accepted by the local application and added to the local application's OAuth client.

> **Note**
>
> The use of scope-granting features establish a strong dependency between the involved applications and should be avoided wherever possible. Instead of using the granting functionality, see if the features provided by the XSUAA broker plan supports your application's authorization needs in a better way.

authorities

To enable one (requesting) application to accept and use the authorization scopes granted by another application, each application must be bound to its own instance of the XS UAA service; this is required to ensure that the applications are registered as OAuth 2.0 clients at the UAA. You must also add an `"authorities"` section to the security descriptor file of the application that is requesting an authorization scope. In the `"authorities"` section of the requesting application's security descriptor, you can either specify that **all** scope authorities configured as grantable in the granting application should be accepted by the application requesting the scopes, or alternatively, only individual, named, scope authorities:

- Request and accept **all** authorities flagged as "grantable" in the granting application's security descriptor

  > **Sample Code**
  >
  > Specifying Scope Authorities in the XS Advanced Requesting Application's Security Descriptor
  >
  > ```
  > "authorities":["$ACCEPT_GRANTED_AUTHORITIES"]
  > ```

- Request and accept only the **specified** scope authority that is flagged as grantable in the specified granting application's security descriptor:

  > **Sample Code**
  >
  > Specifying Scope Authorities in the XS Advanced Requesting Application's Security Descriptor
  >
  > ```
  > "authorities":["<GrantingApp>.ForeignCall", "<GrantingApp2>.ForeignCall",]
  > ```

Since both the requesting application and the granting application are bound to UAA services using the information specified in the respective application's security descriptor, the requesting application can accept and use the specified authority from the granting application. The requesting application is now authorized to access the granting application and perform some tasks.

> **Note**
>
> The granted authority always includes the prefixed name of the application that granted it. This information tells the requesting application which granting application has granted which set of authorities.

The scope authorities listed in the requesting application's security descriptor must be specified as "grantable" in the granting application's security descriptor.

> **→ Tip**
>
> To flag a scope authority as "grantable", add the `"grant-as-authority-to-apps"` property to the respective scope in the granting application's security descriptor.

oauth2-configuration

Use the custom `oauth2-configuration` property to define custom values for the OAuth 2.0 clients, such as the amount of time for which an OAuth token is valid or the Universal Resource Indicators (URI) to which you want to redirect application requests. The `xsuaa` service broker registers and uses the values defined in `oauth-configuration` to configure the OAuth 2.0 clients.

The following code illustrates a simple example of the OAuth 2.0 client configuration:

Sample Code

```
"oauth2-configuration": {
   "token-validity": 900,
   "refresh-token-validity": 1800,
   "redirect-uris": [
                    "https://<host_name1>",
                    "https://<host_name2>"],
   "autoapprove": "false",
   "system-attributes": ["groups", "rolecollections"],
   "allowedproviders": ["useridp1", "useridp2"]
}
```

The following configuration keys are available:

oauth2-configuration Parameters

Key	Mandatory	Description	Example
`token-validity`	No	The amount of time (in seconds) for which the application's JSON Web Token (JWT) is valid. If no value is defined, the default value of 12 hours (43,200 seconds) is used.	`900` (15 minutes)
`refresh-token-validity`	No	The amount of time (in seconds) for which the application's refresh JSON Web Token (JWT) is valid. If no value is defined, the default value (30 days) is used.	`1800` (30 minutes)

Key	Mandatory	Description	Example
`redirect-uris`	No	Array defining a list of the allowed redirect URIs. This allow list overwrites the default URIs provided by the XS advanced User Account and Authentication service (UAA). Default: localhost, `[xs | cf ] apps domain`	`["https://<host_name1>","https://<host_name2>"]`
`autoapprove`	No	Determines whether the user has to approve the requested scopes (“false”) or if the scopes are assigned without approval (“true”) during the token retrieval. Allowed values are “true” (default) and “false”	`“false”`

Key	Mandatory	Description	Example
`system-attributes`	No	Array of system attributes to add to the JSON Web Token. Allowed values are: “groups”, “role-collections” **! Restriction** Only for use in Cloud-Foundry environments: not in XS advanced.	`["groups", "rolecollections"]`
`allowedproviders`	No	Array listing the allowed identity providers (IdP) to use for authentication purposed. By default **all** providers are allowed.	`["useridp1", "useridp2"]`

Related Information

Create the Security Descriptor for an XS Advanced Application [page 995]
XS Advanced Application Router Configuration Syntax [page 922]
Service Plans and Resources in XS Advanced [page 856]
User Account and Authentication Services in XS Advanced [page 841]

11.7 Create an Instance of the XS UAA Service

Use the service broker to create an instance of the `xsuaa` service in XS advanced.

Context

If you want to grant users access to an XS advanced application, you must create an instance of the XS advanced User Account and Authentication service (XSUAA) for this application; the XSUAA service instance acts as an OAuth 2.0 client for the bound application.

Procedure

1. Create a service instance based on the `xsuaa` service and using the service plan “default” and the security descriptors defined in the `xs-security.json` file.

   ```
   xs create-service <service _name> <service_plan> <service_instance_name> -c xs-security.json
   ```

   > Example
   >
   > ```
   > xs create-service xsuaa default authorizationtest-uaa -c xs-security.json
   > ```

   The name of the OAuth 2.0 client you created is similar to the following example: `sb-577a76db-21f9-412b-ad4e-a740f3991136`.

2. Display a list of the OAuth 2.0 clients that are available.

   To check that your new service instance is among the list of currently available OAuth 2.0 clients, log on to the UAA using the following URL and a known UAA user:

   ```
   http://<host_name>:<uaa port>
   ```

   a. Log on to the XS controller.

      ```
      xs login -a <api_endpoint> -u <username> -p <password>
      ```

   b. List the applications and services currently running.

      The information displayed in the console includes URL and port numbers.

      ```
      xs apps
      ```

      ```
      xs services
      ```

3. Update details of an existing XSUAA service's configuration.

   The XSUAA service for user authentication is configured using the security descriptor `xs-security.json`. You can change selected details of the security configuration and update the XSUAA service with the modified configuration.

To modify details of an existing XSUAA service's security configuration, use the `xs update-service` command as follows:

```
xs update-service <SERVICE_INSTANCE> -c xs-security.json
```

> ! Restriction
>
> It is not possible to update the service plan assigned to an instance of the XS User Account and Authentication (XSUAA) service.

When updating the configuration of the XSUAA service, you can add or delete any defined scopes, security-related attributes, or role templates. However, **modifications** are only permitted to the following elements of an existing service's security configuration:

- A security scope's description and “granted applications”
- A security attribute's description
  When updating the XSUAA service, it is not permitted to change the value type of an attribute defined in the `xs-security.json` application security descriptor.
- A role template's description, scope reference, or attribute reference

> ! Restriction
>
> When updating the configuration of an XSUAA service instance, it is only possible to delete an attribute reference; it is not possible to add or modify them.

Related Information

Create a Service Instance [page 861]
Modifications and Restrictions for XSUAA Service Updates [page 1014]

11.7.1 Modifications and Restrictions for XSUAA Service Updates

Restrictions apply to which parts of the XS advanced security description you can change and apply when updating an instance of the XSUAA service.

The XS advanced service broker allows you to update the configuration of an existing service, for example, to change the service plan associated with a service. You can also use a service update to modify the XSUAA service's security configuration, for example, by adding, deleting, or changing the artifacts defined in the `xs-security.json` security descriptor. If you want to use a service-update operation to make changes to

the XSUAA service's security configuration, bear in mind that certain restrictions apply, as illustrated in the following table:

Permitted Changes to xs-security.json when Updating the XSUAA Service

Security Artifacts	Add	Delete	Change
Authorization Scope	Yes	Yes	You can only modify the scope's: • `description` • `granted-apps`
Attribute	Yes	Yes	You can only modify the attribute's: • `description` **i Note** It is not possible to change an attribute's `valueType` during a service-update operation.
Role template	Yes	Yes	You can only modify a role template's: • `description` • `scope-references` • `attribute-references` **! Restriction** An attribute reference can only be deleted; it cannot be modified during a service-update operation.

Related Information

Create the Security Descriptor for an XS Advanced Application [page 995]
The Application Security Descriptor [page 996]
Application Security Descriptor Configuration Syntax [page 999]

11.8 Bind the XS UAA Service Instance to the XS Advanced Application

You must bind the UAA service instance you created to the XS advanced application that is going to use it.

Context

You must bind the service instance that acts as OAuth 2.0 client to your XS advanced application. The name of the service is defined in the `services` property in the application's manifest file (`manifest.yml`). The manifest file is located in the root folder of your XS advanced application, for example `node-hello-world`.

Procedure

Bind an application to a service instance:

```
xs bind-service <APP_NAME> <SERVICE_INSTANCE>
```

> Example
>
> ```
> xs bind-service hello-world xsuaa
> ```

You can now deploy your application with the authorization artifacts that were created earlier.

Related Information

Create a Service Instance [page 861]
XS Advanced Application Routes and Destinations [page 936]
XS CLI: Services Management

11.9 Enable Applications to Perform Tasks with the Authority of Other Applications

You can use authorization scopes to enable an application to perform tasks on behalf of another application and in the second application's container.

Context

In XS advanced, an application can be configured to grant the authorization defined in its own scopes to another application on request, for example, to allow the requesting (external) application to perform tasks on behalf of the local application; that is, with the authority of the local application, and in the local application's container. To configure one application to perform tasks with the authority of another application, perform the following steps:

Procedure

1. Enable an XS advanced application to grant scope authorities to another (requesting) XS advanced application.

   In the security descriptor of the application granting the use of a scope authority, use the `"grant-as-authority-to-apps"` property to configure individual scopes to be "grantable" to another authorized application on request.

   ```
   "scopes": [
          {
            "name": "$XSAPPNAME.foreigncall",
            "description": "Enable Calls into Granting App",
            "grant-as-authority-to-apps": ["RequestingApp"]
          },
   ]
   ```

2. Bind the application granting the scope authority to its respective UAA service.

   After you make the necessary changes to the scopes in the security descriptor of the application that is granting the scope authorities, you must rebind the application to its UAA service instance.

   ```
   cf bind-service GrantingApp xsuaa –c xs-security.json
   ```

   > → Tip
   >
   > If no such instance of the `xsuaa` service exists, then create it using the `xs create-service` command.

3. Enable the requesting application to accept the scope authorities granted by another application.

   In the security descriptor of the application requesting the authorization scopes granted by another application, list the requested scopes and the application granting them in the `“authorities”` property.

You can configure **all** granted scopes to be accepted and used, or individual named scopes, as indicated below:

- Accept all granted authorities:

```
"authorities":["$ACCEPT_GRANTED_AUTHORITIES"],
```

- Accept the specified authorities from the named applications:

```
"authorities":["<GrantingApp>.ForeignCall", "<GrantingApp2>.ExternalCall"],
```

4. Bind the requesting application to its respective UAA service instance.

   After you have made the necessary changes to the security descriptor of the application requesting, accepting, and using the scope authorities granted by the first application, you must rebind the requesting application to its UAA service instance so that it updates its configuration with the scope changes made to the security descriptor (`xs-security.json`).

```
cf bind-service RequestingApp xsuaa –c xs-security.json
```

   > → Tip
   >
   > If no such instance of the `xsuaa` service exists, then create it using the `xs create-service` command.

5. Test the requesting and granting of the configured scopes and authorities.

   Start all applications and test that the requesting application receives the granted authority and can perform the planned task with the required authorization in the granting application's container.

Related Information

Application Security Descriptor Configuration Syntax [page 999]
Create an Instance of the XS UAA Service [page 1013]
Bind the XS UAA Service Instance to the XS Advanced Application [page 1016]

11.9.1 Authorization Scopes and Authorities

An overview of the configuration required to enable one XS advanced application to perform a task with the authority of another XS advanced application.

In the XS advanced run-time environment, it is possible to authorize one application to make calls to other applications to trigger an action on the other application's behalf. For example, you could configure one application to trigger a scheduled job that cleans up unwanted logs and collects run-time "garbage" on behalf (and with the authority) of another application. In this scenario, one application grants the authorities, and another application requests the authorities, accepts them, and uses the authorities to access the granting application and perform tasks on the granting application's behalf. By enabling applications to grant and use authorities in this way, it is possible to ensure that not only the communication between the applications but also the execution of the task itself run securely; that is, with the required permissions and in the correct application run-time container.

To enable the exchange and use of authorities between applications, at least two applications are required: one that grants the authority, and a second, named, application that performs the desired action on behalf of the application granting the authority. In the security descriptor (`xs-security.json`) of the application granting a named scope as an authority, you need to flag the specified scope as "grantable", for example, using the property `"grant-as-authority-to-apps"`, as illustrated in the following example:

Code Syntax

Granting Scopes as Authorities (`xs-security.json` of Granting Application)

```
"scopes": [
        {
          "name": "$XSAPPNAME.foreigncall",
          "description": "Enable Calls into Granting App",
          "grant-as-authority-to-apps": ["RequestingApp"]
        },
]
```

In the application that requires the granted authority, you need to add the property `"authorities"` to the requesting application's security descriptor and either list the individual, named authorities that can be used, or specify that **all** authorities granted by another application can be accepted and used, as illustrated in the following example:

Code Syntax

Requesting and Using Scopes as Authorities (`xs-security.json` of Requesting Application)

```
"authorities":["$ACCEPT_GRANTED_AUTHORITIES"],
```

Each of the applications involved in the exchange of authorities must be bound to its own an instance of the XS advanced User Account and Authentication (UAA) service; the service binding creates the respective OAuth 2.0 clients and enables secure communication between the applications involved in the exchange of authorities. The UAA binding also enables the application requesting an authority to authenticate itself at the granting application **on its own behalf** using the OAuth 2.0 client credentials.

→ Tip

When an OAuth 2.0 client acts on behalf of a user, it requires authorization scopes, which are defined in its own security descriptor, (`xs-security.json`). When an OAuth 2.0 client acts on its **own** behalf and no user is involved, the client requires "authorities", which are defined in the security descriptor of another application - the granting application. An authority is a scope that is flagged as "grantable" so that it can be granted to another application in order to enable the requesting application to perform tasks with the authority of the granting application.

Related Information

Enable Applications to Perform Tasks with the Authority of Other Applications [page 1017]
Application Security Descriptor Configuration Syntax [page 999]
The Application Security Descriptor [page 996]

11.10 Maintain Values in the SAP HANA Secure Store

Insert entries into (and retrieve and remove entries from) the SAP HANA Secure Store.

Prerequisites

To maintain content in the SAP HANA Secure Store, bear in mind the following conditions:

- The XS advanced application must be bound to an instance of the `hana` (or `managed-hana`) service using the `securestore` service plan.
- For XS advanced applications, the resource type for the database connection specified in the MTA's deployment descriptor (`mta.yaml` file) must be `com.sap.xs.hana-securestore`.

Context

In the context of XS advanced, the SAP HANA Secure Store is used to maintain sensitive information such as user credentials. The stored procedures listed in the following table enable you to insert, retrieve, and delete entries in the Secure Store. Any XS advanced application that is bound to an instance of the `hana` (or `managed-hana`) service with the service plan `securestore` has access to these stored procedures and can use them to maintain content stored in the Secure Store:

Stored Procedures for Secure Store Maintenance

Stored Procedure	Description	Required Parameters

`SYS.USER_SECURESTORE_INSERT`	Insert an encrypted entry into the SAP HANA Secure Store.	`STORE_NAME`
		`FOR_XS_APPLICATIONUSER`
		`KEY`
		`VALUE`

`SYS.USER_SECURESTORE_RETRIEVE`	Retrieve an encrypted entry from the SAP HANA Secure Store.	`STORE_NAME`
		`FOR_XS_APPLICATIONUSER`
		`KEY`
		`VALUE`

Stored Procedure	Description	Required Parameters

`SYS.USER_SECURESTORE_DELETE`	Delete an encrypted entry from the SAP HANA Secure Store.	`STORE_NAME`
		`FOR_XS_APPLICATIONUSER`
		`KEY`

→ Tip

For more information about the parameters required when calling the Secure-Store procedures, see *Related Information* below.

You can use XS advanced applications or SAP HANA SQL to maintain entries in the SAP HANA Secure Store:

Procedure

1. Insert encrypted values into the SAP HANA Secure Store.

   You can use an XS advanced application or SAP HANA SQL to insert encrypted values into the SAP HANA Secure Store by calling the stored procedure `SYS.USER_SECURESTORE_INSERT`.
2. Retrieve encrypted values from the SAP HANA Secure Store.

   You can use an XS advanced application or SAP HANA SQL to request entries from the SAP Secure Store by calling the stored procedure `SYS.USER_SECURESTORE_RETRIEVE`.
3. Delete encrypted values from the SAP HANA Secure Store.

   You can use an XS advanced application or SAP HANA SQL to remove entries from the SAP HANA Secure Store by calling the stored procedure `SYS.USER_SECURESTORE_DELETE`.
4. Set up a secure connection to the SAP HANA Secure Store.

   To avoid authorization problems when the XS advanced application tries to connect to the database, it is essential to ensure that the resource type for the database connection specified in the Multi-Target Application's development descriptor (`mta.yaml` file) is `com.sap.xs.hana-securestore`.

Related Information

SAP HANA Secure-Store Interface Procedures and Parameters [page 1022]
Access the SAP HANA Secure Store with Node.js [page 1025]
Access the SAP HANA Secure Store with Java [page 1031]
Access the SAP HANA Secure Store with SQL [page 1041]

11.10.1 SAP HANA Secure-Store Interface Procedures and Parameters

A list of the parameters available for interaction with the SAP HANA Secure Store using the dedicated stored procedures.

An application that is bound to an instance of the `hana` or `managed-hana` service with the service plan `securestore` has access to the following stored procedures, which enable encrypted access to the SAP HANA Secure Store:

- `SYS.USER_SECURESTORE_INSERT`
- `SYS.USER_SECURESTORE_RETRIEVE`
- `SYS.USER_SECURESTORE_DELETE`

The information in the following sections describes the parameters available to make use of these interfaces to the SAP HANA Secure Store.

> → Tip
>
> The information in this section is displayed in the `DEFINITION` column of the SAP HANA System view `PROCEDURES`.

SYS.USER_SECURESTORE_INSERT

The following table lists the parameters used to describe an entry inserted into the SAP HANA Secure Store with the procedure `SYS.USER_SECURESTORE_INSERT`:

Parameter	Direction	Type	Description
STORE_NAME	IN	NVARCHAR(530)	The name of the target secure store for the insert operation
FOR_XS_APPLICATIONUSER	IN	BOOLEAN	Indicates if the inserted value is stored for the XS advanced application user (“true”) or the technical database user (“false”). **i Note** If `FOR_XS_APPLICATIONUSER` is set to “true”, the session variable `<XS_APPLICATIONUSER>` must be set, too.

Parameter	Direction	Type	Description
KEY	IN	NVARCHAR(1024)	The key to be inserted into the target secure store (`STORE_NAME`)
VALUE	IN	VARBINARY(5000)	The binary value of the entry to be inserted into the secure store. Non-binary values (numbers, dates, strings) must be converted, for example, using `TO_BINARY` in the database or by using a JavaScript Buffer to maintain the request body content.

SYS.USER_SECURESTORE_RETRIEVE

The following table lists the parameters used to describe an entry retrieved from the SAP HANA Secure Store with the procedure `SYS.USER_SECURESTORE_RETRIEVE`:

Parameter	Direction	Type	Description
STORE_NAME	IN	NVARCHAR(530)	The name of the target secure store for the retrieve operation
FOR_XS_APPLICATIONUSER	IN	BOOLEAN	Indicates if the retrieved value is stored for the XS advanced application user (“true”) or the technical database user (“false”). **i Note** If `FOR_XS_APPLICATIONUSER` is set to “true”, the session variable `<XS_APPLICATIONUSER>` must be set, too.
KEY	IN	NVARCHAR(1024)	The key to be retrieved from the target secure store (`STORE_NAME`)

Parameter	Direction	Type	Description
VALUE	OUT	VARBINARY(5000)	The binary value of the entry to be retrieved from the secure store. Binary values can be converted for further use, for example, with `TO_NVARCHAR` in the database or by using a JavaScript Buffer to maintain the request body content.

SYS.USER_SECURESTORE_DELETE

The following table lists the parameters used to describe an entry removed from the SAP HANA Secure Store with the procedure `SYS.USER_SECURESTORE_DELETE`:

Parameter	Direction	Type	Description
STORE_NAME	IN	NVARCHAR(530)	The name of the target secure store for the delete operation
FOR_XS_APPLICATIONUSER	IN	BOOLEAN	Indicates if the value to be deleted is stored for the application user (“true”) or the technical database user (“false”). **i Note** If `FOR_XS_APPLICATIONUSER` is set to “true”, the session variable `<XS_APPLICATIONUSER>` must be set, too.
KEY	IN	NVARCHAR(1024)	The key to be deleted within the target secure store (`STORE_NAME`)

Related Information

Maintain Values in the SAP HANA Secure Store [page 1020]

SAP HANA and HDI Services in XS Advanced [page 834]

11.10.2 Access the SAP HANA Secure Store with Node.js

Use a Node.js application to maintain entries in the SAP HANA Secure Store.

Prerequisites

To maintain content in the SAP HANA Secure Store, bear in mind the following conditions:

- The XS advanced application must be bound to an instance of the `hana` (or `managed-hana`) service using the `securestore` service plan.
- For XS advanced applications, the resource type for the database connection specified in the MTA's deployment descriptor (`mta.yaml` file) must be `com.sap.xs.hana-securestore`.

Context

In the context of XS advanced, the SAP HANA Secure Store is used to maintain sensitive information such as user credentials. Access to the SAP HANA Secure Store is enabled by stored procedures which you can use to insert, retrieve, and delete entries in the Secure Store. Any XS advanced application that is bound to an instance of the `hana` (or `managed-hana`) service with the service plan `securestore` has access to these stored procedures and can use them to maintain content stored in the Secure Store.

> → Tip
>
> For more information about the parameters required when calling the Secure-Store procedures, see *Related Information* below.

The following examples show how to use XS advanced Node.js applications to maintain entries programmatically in the SAP HANA Secure Store:

Procedure

1. Insert encrypted values into the SAP HANA Secure Store.

The following example shows how to use a Node.js application to insert encrypted entries into the SAP HANA Secure Store; the example uses `app.post()` to insert the entry and includes the content in the request body:

Sample Code

Call SYS.USER_SECURESTORE_INSERT (Node.js Application)

```
/*eslint no-console: 0, no-unused-vars: 0, no-shadow: 0, new-cap: 0*/
"use strict";
var express = require("express");
module.exports = () => {
     var app = express.Router();

     var bodyParser = require("body-parser");
     app.use(bodyParser.raw({
             type: "text/plain"
     }));  //Process request Body and return a Buffer
     //Secure Store Insert
     app.post("/:key?", async(req, res) => {
         const hdbext = require("@sap/hdbext");
         const key = req.params.key;
         let inputParams = "";
         if (typeof key === "undefined" || key === null) {
                 res.type("text/plain").status(500).send("ERROR: No Secure
Store Key Input Parameter Specified");
                  return;
         } else {
                 inputParams = {
                       KEY: key };
         }
         inputParams.STORE_NAME = "MY_SECURE_STORE";
         inputParams.FOR_XS_APPLICATIONUSER = true;
         inputParams.VALUE = req.body;
          const client = await getSecureStore();
         //(client, Schema, Procedure, callback)
         hdbext.loadProcedure(req.db, "SYS", "USER_SECURESTORE_INSERT",
(err, sp) => {
             if (err) {
                 res.type("text/plain").status(500).send(`ERROR: $
{err.toString()}`);
                 return;
             }
             //(Input Parameters, callback(errors, Output Scalar
Parameters, [Output Table Parameters])
             sp(inputParams, (err, parameters, results) => {
                 if (err) {
                     res.type("text/plain").status(500).send(`ERROR: $
{err.message.toString()}`);
                     return;
                 }
                 res.type("application/json").status(200).send(`Entry in
secure store successfully created for key: ${key} and value: $
{req.body.toString("utf8")} `);
             });
         });
     });
     //Secure Store Retrieve
     app.get("/:key?", (req, res) => {
     [...]
     });
     //Secure Store Delete
     app.delete("/:key?", (req, res) => {[...]
     });
     return app;
```

```
};
```

Bear in mind that, in the Node.js example above, the `KEY` parameter requires a `VALUE` of type `VARBINARY`, so any hard-coded values (numbers, dates, strings) in the `INSERT` call need to be converted, for example, using `TO_BINARY` on the database side or by using a JavaScript Buffer. In this case, we specify a body-parsing module earlier in the code which sets the body content used to fill the `VALUE` parameter to `raw` and ensures that no conversion is required.

2. Retrieve encrypted values from the SAP HANA Secure Store.

   The following example shows how to use a Node.js application to retrieve encrypted entries from the SAP HANA Secure Store:

   Sample Code

   Call SYS.USER_SECURESTORE_RETRIEVE (Node.js Application)

```
//Secure Store Retrieve
app.get("/:key?", async(req, res) => {
     const hdbext = require("@sap/hdbext");
     const key = req.params.key;
     let inputParams = "";
     if (typeof key === "undefined" || key === null) {
            res.type("text/plain").status(500).send("ERROR: No Secure
Store Key Input Parameter Specified");
            return;
     } else {
            inputParams = {
                   KEY: key };
    }
    inputParams.STORE_NAME = "MY_SECURE_STORE";
    inputParams.FOR_XS_APPLICATIONUSER = true;
     const client = await getSecureStore();
    //(client, Schema, Procedure, callback)
    hdbext.loadProcedure(req.db, "SYS", "USER_SECURESTORE_RETRIEVE", (err,
sp) => {
        if (err) {
            res.type("text/plain").status(500).send(`ERROR: $
{err.toString()}`);
            return;
        }
        //(Input Parameters, callback(errors, Output Scalar Parameters,
[Output Table Parameters])
        sp(inputParams, (err, parameters, results) => {
            if (err) {
                res.type("text/plain").status(500).send(`ERROR: $
{err.message.toString()}`);
                return;
            }
            res.type("application/json").status(200).send(`Entry in secure
store successfully retrieved for key: ${key} and value: $
{results.VALUE.toString("utf8")} `);
        });
    });
});
```

Bear in mind that, in the Node.js example above, the `KEY` parameter returns a `VALUE` of data type `VARBINARY`, which if required can be converted to a string, for example, by using `TO_NVARCHAR` in the database or by using a JavaScript Buffer for the content of the request body.

3. Delete encrypted values from the SAP HANA Secure Store.

The following example shows how to use a Node.js application to remove parameter required for inserting entries into and retrieving entries from the Secure Store expects a

Sample Code

Call SYS.USER_SECURESTORE_DELETE (Node.js Application)

```
//Secure Store Delete
app.delete("/:key?", async(req, res) => {
     const hdbext = require("@sap/hdbext");
     const key = req.params.key;
     let inputParams = "";
     if (typeof key === "undefined" || key === null) {
           res.type("text/plain").status(500).send("ERROR: No Secure Store
Key Input Parameter Specified");
           return;
     } else {
            inputParams = {
                  KEY: key
            };
     }
     inputParams.STORE_NAME = "MY_SECURE_STORE";
     inputParams.FOR_XS_APPLICATIONUSER = true;
     const client = await getSecureStore();
     //(client, Schema, Procedure, callback)
     hdbext.loadProcedure(req.db, "SYS", "USER_SECURESTORE_DELETE", (err,
sp) => {) in the database or
                        by ensuring that the content of the request body
is written to a JavaScript
          if (err) {
               res.type("text/plain").status(500).send(`ERROR: $
{err.toString()}`);//Secure Store Delete
                        ensure that the content used to fill the
app.delete("/:key?", async(req, res) =
               return;>
          } {
          //(Input Parameters, callback(errors, Output Scalar Parameters,
[Output Table Parameters
     const hdbext = require("@sap/hdbext"); ]
     const key = req.params.key; )
     let inputParams = "";
          sp(inputParams, (err, parameters, results) =
     if (typeof key === "undefined" || key === null) { >
           res.type("text/plain").status(500).send("ERROR: No Secure Store
Key Input Parameter Specified"); {
           return;
               if (err) {
                    res.type("text/plain").status(500).send(`ERROR: $
{err.message.toString()}`);
                    return;
               }
     } else {
               res.type("application/json").status(200).send(`Entry in
secure store successfully deleted for key: ${key} `);
              inputParams = {
          });
                    KEY: key
     });
            };
     }
     inputParams.STORE_NAME = "MY_SECURE_STORE";
});
     inputParams.FOR_XS_APPLICATIONUSER = true;
```

4. Encode the content of the application's request body so that it is compatible with the data type used for `INSERT` and `RETRIEVE` operations that require a `VARBINARY` value for the entry's `KEY` parameter.

> **Note**
>
> This example applies to Node.js applications only.

For operations where the `KEY` expects a `VALUE` with the data type `VARBINARY`, any hard-coded values (numbers, dates, strings) in the `INSERT` or `RETRIEVE` calls need to be converted, for example, using `TO_BINARY` or `TO_VARCHAR`.

> **Sample Code**
>
> Process Request Body and Return a JavaScript Buffer

```
"use strict";
const express = require("express");
module.exports = () => {
     const app = express.Router();
     const bodyParser = require("body-parser");
       app.use(bodyParser.raw({
               type: "text/plain"
       }));
```

5. Set up a secure connection to the SAP HANA Secure Store.

   To avoid authorization problems when the XS advanced application tries to connect to the database, it is essential to ensure that the resource type for the database connection specified in the MTA's development descriptor (`mta.yaml` file) is `com.sap.xs.hana-securestore`.

   a. Add a new secure-connection resource type to the MTA's development descriptor.

      To ensure a secure connection to the database, we use an instance of the `hana` service with the service plan `securestore`. For this, we need to define a corresponding resource in the application's development descriptor (`MTA.yaml`), and specify that it is required by the Node.js module using it, as illustrated in the following (**incomplete**) code snippet:

      > **Sample Code**
      >
      > Resources in the Development Descriptor (`MTA.yaml`) of a Node.js Application

```
ID: MySecureStoreApp
_schema-version: '2.0'
version: 1.0.0
modules:
  - name: srv
    type: nodejs
    path: srv
    requires:
      - name: MyDBConnection-secure
    provides:
      - name: srv_api
        properties:
          url: '${default-url}'
resources:
  - name: myHANA-hdi-container
    properties:
      hdi-container-name: '${service-name}'
    type: com.sap.xs.hdi-container
  - name: mySAP-ex-uaa
    type: com.sap.xs.uaa-space
    parameters:
      path: ./xs-security.json
```

```
  - name: MyDBConnection-secure
    type: com.sap.xs.hana-securestore
```

b. Initialize a connection to the Secure Store resource.

The following Node.js code shows a function (`getSecureStore()`) that uses two SAP Node.js modules (`@sap/xsenv` and `@sap/hdbext`) to request information about the `securestore` service instance required for the connection and then use this information to help set up a secure connection to the database:

Sample Code

Connect the XS Application's Database Module to the Secure-Store Resource (Node.js)

```
/*eslint no-console: 0, no-unused-vars: 0, no-shadow: 0, new-cap: 0*/
/*eslint-env node, es6 */
"use strict";
const express = require("express");
module.exports = () => {
    const app = express.Router();
    const bodyParser = require("body-parser");
    app.use(bodyParser.raw({
        type: "text/plain"
    })); //Process request Body and return a Buffer
    function getSecureStore() {
        return new Promise((resolve, reject) => {
            const xsenv = require("@sap/xsenv");
            let hanaOptions = xsenv.getServices({
                secureStore: {
                    plan: "securestore"
                }
            });

            var hdbext = require("@sap/hdbext");
            hdbext.createConnection(hanaOptions.secureStore, (error,
client) => {
                if (error) {
                    reject(error);
                } else {
                    resolve(client);
                }
            });
        });
    }
     //Secure Store Insert
     app.post("/:key?", async(req, res) => {[...]};
     //Secure Store Retrieve
     app.get("/:key?", async(req, res) => {[...]};
     //Secure Store Delete
     app.delete("/:key?", async(req, res) => {[...]};
     return app;
};
```

Related Information

Maintain Values in the SAP HANA Secure Store [page 1020]
SAP HANA Secure-Store Interface Procedures and Parameters [page 1022]

11.10.3 Access the SAP HANA Secure Store with Java

Use a Java application to maintain entries in the SAP HANA Secure Store.

Prerequisites

To maintain content in the SAP HANA Secure Store, bear in mind the following conditions:

- The XS advanced application must be bound to an instance of the `hana` (or `managed-hana`) service using the `securestore` service plan.
- For XS advanced applications, the resource type for the database connection specified in the MTA's deployment descriptor (`mta.yaml` file) must be `com.sap.xs.hana-securestore`.

Context

In the context of XS advanced, the SAP HANA Secure Store is used to maintain sensitive information such as user credentials. Access to the SAP HANA Secure Store is enabled by stored procedures which you can use to insert, retrieve, and delete entries in the Secure Store. Any XS advanced application that is bound to an instance of the `hana` (or `managed-hana`) service with the service plan `securestore` has access to these stored procedures and can use them to maintain content stored in the Secure Store.

> → Tip
>
> For more information about the parameters required when calling the Secure-Store procedures, see *Related Information* below.

The following examples show how to use an XS advanced Java application to maintain entries programmatically in the SAP HANA Secure Store:

Procedure

1. Insert encrypted values into the SAP HANA Secure Store.

   The following example shows how to use a Java application to insert encrypted entries into the SAP HANA Secure Store; the `SecureStoreService` is configured in another file:

   > Sample Code
   >
   > Call SYS.USER_SECURESTORE_INSERT (Java Application)

   ```
   package bla.bla.test.securestore.service;
   import java.sql.CallableStatement;
   import java.sql.Connection;
   import java.sql.SQLException;
   import java.sql.Types;
   import javax.sql.DataSource;
   ```

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Service;
@Service
public class SecureStoreServiceImpl implements SecureStoreService {
    private static final Logger LOGGER =

LoggerFactory.getLogger(SecureStoreService.class);
    @Autowired
    @Qualifier("secureStoreDataSource")
    private DataSource dataSource;

    @Override
    public void writeValue(String storeName, String key, String value)
                                                          throws SQLException
{
       LOGGER.info("secure store write value: " + storeName + " / "
                                                        + key + " / " +
value);

       this.deleteValue(storeName, key);
       try (Connection connection = dataSource.getConnection()) {

       CallableStatement callableStatement = connection
                 .prepareCall("{call
SYS.USER_SECURESTORE_INSERT(?,?,?,?)}");
       callableStatement.setString(1, storeName);
       callableStatement.setBoolean(2, false);
       callableStatement.setString(3, key);
       callableStatement.setBytes(4, value.getBytes());
       callableStatement.executeUpdate();
       }
    }
```

In the Java example above, the `KEY` parameter for the inserted entry expects a `VALUE` (4) with the BINARY data type, so any hard-coded values (numbers, dates, strings) in the `INSERT` call need to be converted. In this case, we ensure the data type is correctly set with `setBytes(4, value.getBytes())`.

2. Retrieve encrypted values from the SAP HANA Secure Store.

   The following example shows how to use a Java application to retrieve encrypted entries from the SAP HANA Secure Store:

Sample Code

Call SYS.USER_SECURESTORE_RETRIEVE (Java Application)

```
@Override
public void getValue(String storeName, String key) throws SQLException {
       LOGGER.info("secure store read value: " + storeName + " / " + key);
       try (Connection connection = dataSource.getConnection()) {

       CallableStatement callableStatement = connection
             .prepareCall("{call SYS.USER_SECURESTORE_RETRIEVE(?,?,?,?)}");
       callableStatement.setString(1, storeName);
       callableStatement.setBoolean(2, false);
       callableStatement.setString(3, key);
       callableStatement.registerOutParameter(4, Types.VARBINARY);

       callableStatement.executeUpdate();
       byte[] bytes = callableStatement.getBytes(4);

       if (null == bytes) {
```

```
                LOGGER.info("value is null");
                return null;
        } else {
                String returnValue = new String(bytes);
                LOGGER.info("return value is :" + returnValue);
                return returnValue; 68 }
        }
    }
}
```

In the Java example above, the `KEY` parameter returns a `VALUE` (4) with the VARBINARY data type. In this example, we ensure the data type for the OUT parameter is correctly set with `registerOutParameter(4, Types.VARBINARY)`.

3. Delete encrypted values from the SAP HANA Secure Store.

   The following example shows how to use a Java application to remove encrypted entries from the SAP HANA Secure Store:

   Sample Code

   Call SYS.USER_SECURESTORE_DELETE (Java Application)

   ```
   @Override
   public void deleteValue(String storeName, String key) throws SQLException {
           LOGGER.info("secure store delete value: " + storeName + " / " +
   key);
           try (Connection connection = dataSource.getConnection()) {

           CallableStatement callableStatement = connection
                       .prepareCall("{call
   SYS.USER_SECURESTORE_DELETE(?,?,?)}");
           callableStatement.setString(1, storeName);
           callableStatement.setBoolean(2, false);
           callableStatement.setString(3, key);

           callableStatement.executeUpdate();
           }
       }
   ```

4. Set up a secure connection to the SAP HANA Secure Store.

   To avoid authorization problems when the XS advanced application tries to connect to the database, it is essential to ensure that the resource type for the database connection specified in the MTA's development descriptor (`mta.yaml` file) is `com.sap.xs.hana-securestore`.

   a. Add a new secure-connection resource type to the MTA's development descriptor.

      To ensure a secure connection to the database, we use an instance of the `hana` service with the service plan `securestore`. For this, we need to define a corresponding resource in the application's development descriptor (`MTA.yaml`), and specify that it is required by the Java module using it, as illustrated in the following (**incomplete**) code snippet:

      Sample Code

      Resources in the Development Descriptor (`MTA.yaml`) of a Java Multitarget Application

      ```
      ID: MySecureStoreApp
      _schema-version: '2.0'
      version: 1.0.0
      modules:
        - name: srv
      ```

```
    type: java
    path: srv
    requires:
      - name: MyDBConnection-secure
    provides:
      - name: srv_api
        properties:
          url: '${default-url}'
resources:
  - name: openSAPHANA-hdi-container
    properties:
      hdi-container-name: '${service-name}'
    type: com.sap.xs.hdi-container
  - name: openSAP-ex-uaa
    type: com.sap.xs.uaa-space
    parameters:
      path: ./xs-security.json
  - name: MyDBConnection-secure
    type: com.sap.xs.hana-securestore
```

b. Initialize a connection to the Secure Store resource.

The following Java code example shows to use the name of the `securestore` service instance to create the data source for the connection to the SAP HANA secure store:

Sample Code

Connect the XS Java Application to the Secure-Store Service-Instance

```
package bla.bla.test.securestore.configuration;
import javax.sql.DataSource;
import org.springframework.cloud.config.java.AbstractCloudConfig;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
@Configuration
public class CloudServiceConfig extends AbstractCloudConfig {
       @Bean public DataSource secureStoreDataSource() {
              // use service-instance name to create the data source
              return connectionFactory().dataSource("hana-securestore");
}
       @Bean public DataSource schemaDataSource() {
              // use service-instance name to create the data source
              return connectionFactory().dataSource("hana-schema");
       }
}
```

c. Manage the requests to access the secure store service.

The following code example shows how the Java application controls the `POST`, `GET`, and `DELETE` requests used to insert, retrieve, and delete entries in the SAP HANA secure store:

Sample Code

Control Access to the Java Application's Secure-Store Service Instance

```
package bla.bla.test.securestore.controller;
import java.sql.SQLException;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import bla.bla.test.securestore.service.SecureStoreService;
```

```
@RestController
@RequestMapping("/securestore")
public class SecureStoreController {
    @Autowired
    private SecureStoreService secureStoreService;
    @PostMapping
    public void setValue() throws SQLException {
        secureStoreService.writeValue("store", "key", "value");
    }
    @GetMapping
    public String get() throws SQLException {
        return secureStoreService.getValue("store", "key");
    }
    @DeleteMapping
    public void delete() throws SQLException {
        secureStoreService.deleteValue("store", "key");
    }
}
```

Related Information

SAP HANA Secure-Store Interface Procedures and Parameters [page 1022]
Access the SAP HANA Secure Store with Node.js [page 1025]
Access the SAP HANA Secure Store with SQL [page 1041]

11.10.4 Access the SAP HANA Secure Store with Python

Use an XS advanced Python application to maintain entries in the SAP HANA Secure Store.

Prerequisites

To maintain content in the SAP HANA Secure Store, bear in mind the following conditions:

- The XS advanced application must be bound to an instance of the `hana` (or `managed-hana`) service using the `securestore` service plan.
- For XS advanced applications, the resource type for the database connection specified in the MTA's deployment descriptor (`mta.yaml` file) must be `com.sap.xs.hana-securestore`.
- The Python driver for SAP HANA is required; it contains the `dbapi` module that must be imported to enable connections to the database. For more information, see *SAP HANA Client Interface Programming Reference* in *Related Information*.
- This tutorial requires the following Python libraries:
  - `hdbcli`, which is included in the SAP archive `XS PYTHON 1`
  - `sap_cf_logging`, an open-source library that is developed by SAP and published on the Python Package Index (Pypi)

  For more information about downloading these libraries, see *Related Information*.

Context

In the context of XS advanced, the SAP HANA Secure Store is used to maintain sensitive information such as user credentials. Access to the SAP HANA Secure Store is enabled by stored procedures which you can use to insert, retrieve, and delete entries in the Secure Store. Any XS advanced application that is bound to an instance of the `hana` (or `managed-hana`) service with the service plan `securestore` has access to these stored procedures and can use them to maintain content stored in the Secure Store.

The following examples show how to use an XS advanced Python application to maintain entries in the SAP HANA Secure Store:

> → Tip
>
> The sources used for the code samples in this tutorial are also available on GitHub. For more information, see *Related Information* below.

Procedure

1. Request details of the XS advanced service bound to the Python application that needs access to the SAP HANA Secure Store.

   You can read the port number from the relevant environment variable or choose port "9099" as the local default. In this example, we also use the module `cfenv` to import `AppEnv()`, which we use to list details of the Python application's environment and retrieve information about the configuration of the XS advanced service `hana` bound to the Python application.

   > Sample Code
   >
   > SAP HANA database connection credentials (Python Application)

   ```
   from flask import Flask
   from flask import request
   from flask import Response
   from flask import send_from_directory

   import os
   import json
   from cfenv import AppEnv
   from hdbcli import dbapi
   app = Flask(__name__)
   env = AppEnv()
   port = int(os.getenv("PORT", 9099))
   hana = env.get_service(label='hana')
   [. . .]
   ```

2. Set up routes for requests coming through the application router.

   This module's `Flask` Web server is configured to respond to these three routes (URL paths): insert, retrieve, and delete.

   a. If there is no path, then the Web server just returns "Hello World" and the module's instance number. Any requests passed through the application router will never hit this route.

Sample Code

```
@app.route('/')
def hello_world():
    output = '<strong>Python SecureStore! Instance ' +
str(os.getenv("CF_INSTANCE_INDEX", 0)) + '</strong> Try these links.</
br>\n'
    output += '<a href="/python/links">/python/links</a><br />\n'
    return output
[. . .]
```

b. Ensure that the Web browser's request for `favicon.ico` is answered so that the browser does not return a “404” (not found) error.

Sample Code

```
@app.route('/favicon.ico')
def favicon():
    return send_from_directory(os.path.join(app.root_path,
'static'),'favicon.ico', mimetype='image/vnd.microsoft.icon')
[. . .]
```

c. Define requests passed through the application router.

The following code snippet shows how to set up routes (URL paths) that are resolved when requests are made to the Python application to perform an operation in the SAP HANA Secure Store, for example: "insert", "retrieve", or "delete":

Sample Code

```
@app.route('/python/links')
def python_links():
    output = '<strong>Python SecureStore! Instance ' +
str(os.getenv("CF_INSTANCE_INDEX", 0)) + '</strong> Try these links.</
br>\n'
    output += '<a href="/python/insert">/python/insert</a><br />\n'
    output += '<a href="/python/retrieve">/python/retrieve</a><br />\n'
    output += '<a href="/python/delete">/python/delete</a><br />\n'
    return output
[. . .]
```

3. Define the credentials to use when establishing a secure connection to the database.

In this step, we provide the credentials used for the database connection and use the credentials along with the Python driver to establish a connection to the SAP HANA database:

Note

The credentials defined here are valid for the database connection used for all of the Secure Store operations described in this tutorial: `INSERT`, `RETRIEVE`, and `DELETE`.

a. Provide the credentials to use for anonymous access in a database connection.

This information is required for each of the connection operations: insert, retrieve, and delete. For encrypted connections, provision is also provided for a certificate. The following example shows how to

ensure that an XS advanced Python application uses the appropriate credentials to establish a secure connection to a specific SAP HANA database host:

> **i Note**
>
> The `output` variable is used to collect expected values and return them to the browser for confirmation.

> **‹≡› Sample Code**
>
> SAP HANA database connection credentials (Python Application)

```
@app.route('/python/insert')
def securestore_insert():
    output = 'Python SecureStore Insert. \n'
    schema = hana.credentials['schema']
    host = hana.credentials['host']
    port = hana.credentials['port']
    user = hana.credentials['user']
    password = hana.credentials['password']
    # Certificate for SAP HANA service instances that require an
encrypted connection
    if 'certificate' in hana.credentials:
        haascert = hana.credentials['certificate']

    output += 'schema: ' + schema + '\n'
    output += 'host: ' + host + '\n'
    output += 'port: ' + port + '\n'
    output += 'user: ' + user + '\n'
    output += 'pass: ' + password + '\n'
[. . .]
```

b. Configure the secure connection to the SAP HANA database.

The following section of the code sample stores the information obtained in the previous step (a.) in local variables so that they can be easily returned in the output response body in the browser. If necessary the certificate provided in the header is also obtained for use where a secure connection is required.

> **‹≡› Sample Code**
>
> Secure connection to the SAP HANA Database (Python Application)

```
@app.route('/python/insert')
def securestore_insert():
[...]
   if 'certificate' in hana.credentials:
      connection = dbapi.connect(
         address=host,
         port=int(port),
         user=user,
         password=password,
         currentSchema=schema,
         encrypt="true",
         sslValidateCertificate="true",
         sslCryptoProvider="openssl",
         sslTrustStore=haascert
      )
   else:
      connection = dbapi.connect(
         address=host,
         port=int(port),
         user=user,
```

```
            password=password,
            currentSchema=schema
        )
```

4. Insert encrypted values into the SAP HANA Secure Store.

   The following example shows how to use a Python application to insert encrypted entries into the SAP HANA Secure Store; the example uses `dbapi.connect` function shown in the previous sub-step to insert the entry defined in the parameters of the SQL call:

   ‹≡› Sample Code

   Call SYS.USER_SECURESTORE_INSERT (Python Application)

   ```
   @app.route('/python/insert')
   def securestore_insert():
   #    # Provide connection details and certificates
   [...]
   #    # Prepare a cursor for SQL execution
       cursor = connection.cursor()
   #    # Form an SQL statement to retrieve some data
       string2store = 'Whatever!'
       import codecs
       hex2store = (codecs.encode(str.encode(string2store), "hex")).decode()
       try:
           cursor.callproc("SYS.USER_SECURESTORE_INSERT", ("TestStoreName",
   False, "TestKey", hex2store))
           output += 'key TestKey with value ' + string2store + '=' +
   hex2store + ' was inserted into store TestStoreName.' + '\n'
       except:
           output += 'key TestKey likely already exists. Try deleting first.'
   + '\n'
   #    # Close the DB connection
       connection.close()
       # Return the results
       return Response(output, mimetype='text/plain' , status=200,)
   ```

   Bear in mind that, in the Python example above, the `TestKey` parameter requires a `VALUE` of type "`hex`", so any hard-coded values (numbers, dates, strings) in the `INSERT` call need to be converted to the `hex` format. Here, several Python encoding functions are chained together and the resulting value stored in the `hex2store` variable.

5. Retrieve encrypted values from the SAP HANA Secure Store.

   The following example shows how to use a Python application to retrieve encrypted entries from the SAP HANA Secure Store:

   ‹≡› Sample Code

   Call SYS.USER_SECURESTORE_RETRIEVE (Python Application)

   ```
   @app.route('/python/retrieve')
   def securestore_retrieve():
   #    # Provide connection details and certificates
   [...]
   #    # Prepare a cursor for SQL execution
       cursor = connection.cursor()
   #    # Form an SQL statement to retrieve some data
       hexvalue = cursor.callproc("SYS.USER_SECURESTORE_RETRIEVE",
   ("TestStoreName", False, "TestKey", None))
   #    # Close the DB connection
       connection.close()
       import codecs
   ```

```
    if hexvalue[3] is None:
        output += 'key TestKey does not exist in store TestStoreName.  Try
inserting a value first.' + '\n'
    else:
        retrieved = codecs.decode(hexvalue[3].hex(), "hex").decode()
        output += 'key TestKey with value ' + retrieved + ' was retrieved
from store TestStoreName.' + '\n'
    # Return the results
    return Response(output, mimetype='text/plain' , status=200,)
```

Bear in mind that, in the Python example above, the `TestKey` parameter returns a `VALUE` of type "`hex`", so any hard-coded values (numbers, dates, strings) in the `RETRIEVE` results need to be converted with Python functions. In this case, the conversion results in the retrieved variable.

6. Delete encrypted values from the SAP HANA Secure Store.

   The following example shows how to use a Python application to remove encrypted entries from the SAP HANA Secure Store:

   Sample Code

   Call SYS.USER_SECURESTORE_DELETE (Python Application)

```
@app.route('/python/delete')
def securestore_delete():
#    # Provide connection details and certificates
[...]
#    # Prepare a cursor for SQL execution
    cursor = connection.cursor()
#    # Form an SQL statement
    cursor.callproc("SYS.USER_SECURESTORE_DELETE", ("TestStoreName",
False, "TestKey"))
#    # Close the DB connection
    connection.close()

    output += 'key TestKey was deleted from store TestStoreName.' + '\n'
    # Return the results
    return Response(output, mimetype='text/plain' , status=200,)
```

7. Set up a secure connection to the XS advanced service instance that enables access to the SAP HANA Secure Store.

   To avoid authorization problems when the XS advanced application tries to connect to the database, it is essential to ensure that the resource type for the database connection specified in the XS advanced MTA's development descriptor (`mta.yaml` file) is `com.sap.xs.hana-securestore`.

   To ensure a secure connection to the database, we use an instance of the `hana` service with the service plan `securestore`. For this, we need to define a corresponding resource in the application's development descriptor (`MTA.yaml`), and specify that it is required by the Python module using it, as illustrated in the following (**incomplete**) code snippet:

   Sample Code

   Resources in the Development Descriptor (`MTA.yaml`) of a Python Application

```
ID: MySecureStoreApp
_schema-version: '2.0'
version: 1.0.0
modules:
  - name: srv
    type: python
```

```
    path: srv
    requires:
      - name: MyDBConnection-secure
    provides:
      - name: srv_api
        properties:
          url: '${default-url}'
resources:
  - name: myHANA-hdi-container
    properties:
      hdi-container-name: '${service-name}'
    type: com.sap.xs.hdi-container
  - name: mySAP-ex-uaa
    type: com.sap.xs.uaa-space
    parameters:
      path: ./xs-security.json
  - name: MyDBConnection-secure
    type: com.sap.xs.hana-securestore
```

Related Information

Python Samples for SAP HANA Secure-Store on GitHub

Maintain Values in the SAP HANA Secure Store [page 1020]
SAP HANA Secure-Store Interface Procedures and Parameters [page 1022]
Download and Consume Python Libraries [page 782]
Connect to SAP HANA from Python (Client Programming Reference)

11.10.5 Access the SAP HANA Secure Store with SQL

Use SQL to maintain entries the SAP HANA Secure Store.

Context

In the context of XS advanced, the SAP HANA Secure Store is used to maintain sensitive information such as user credentials. Access to the SAP HANA Secure Store is enabled by stored procedures which you can use to insert, retrieve, and delete entries in the Secure Store.

→ Tip

For more information about the parameters required when calling the Secure-Store procedures, see *Related Information* below.

The following examples show how to use SQL to maintain entries in the SAP HANA Secure Store:

Procedure

1. Insert encrypted values into the SAP HANA Secure Store.

   The following example shows how to use SQL to call the stored procedure that inserts encrypted entries into the SAP HANA Secure Store:

   > ‹≡› Sample Code
   >
   > Call SYS.USER_SECURESTORE_INSERT (SQL Application)
   >
   > ```
   > call SYS.USER_SECURESTORE_INSERT ('TestStoreName', False, 'TestKey',
   > 'ab2467cdef');
   > ```

   > **i** Note
   >
   > When using SQL to insert entries into the SAP HANA Secure Store, key values must be provided in hexadecimal form.

2. Retrieve encrypted values from the SAP HANA Secure Store.

   The following example shows how to use SQL to call the stored procedure that retrieves encrypted entries from the SAP HANA Secure Store:

   > ‹≡› Sample Code
   >
   > Stored Procedure USER_SECURESTORE_RETRIEVE (SQL Application)
   >
   > ```
   > call SYS.USER_SECURESTORE_RETRIEVE ('TestStoreName', False, 'TestKey', ?);
   > ```

3. Delete encrypted values from the SAP HANA Secure Store.

   The following example shows how to use SQL to call the stored procedure that deletes encrypted entries from the SAP HANA Secure Store:

   > ‹≡› Sample Code
   >
   > Stored Procedure SYS.USER_SECURESTORE_DELETE (SQL Application)
   >
   > ```
   > call SYS.USER_SECURESTORE_DELETE ('TestStoreName', False, 'TestKey');
   > ```

Related Information

Maintain Values in the SAP HANA Secure Store [page 1020]
SAP HANA Secure-Store Interface Procedures and Parameters [page 1022]

12 Migrating XS Classic Applications to XS Advanced Model

SAP HANA provides tools to help you migrate an XS classic application to the XS advanced run-time environment.

From HANA 1.0 SPS 11, SAP HANA includes an additional run-time environment for application development: SAP HANA extended application services (XS), **advanced model**. SAP HANA XS advanced model represents an evolution of the application server architecture within SAP HANA by building upon the strengths (and expanding the scope) of SAP HANA extended application services (XS), classic model. If you are developing new applications, it is recommended to use SAP HANA XS advanced model.

> → Tip
>
> If you want to migrate existing XS classic applications to run in the new XS advanced run-time environment, SAP recommends that you first check the features available with the installed version of XS advanced. If the XS advanced features match the requirements of the XS classic application you want to migrate, then you can start the migration process, for example, using the *XS Advanced Migration Assistant*, which automates many of the steps in the migration process and is described in more detail in this section.

Before starting the migration process, it is recommended to review the design of your XS classic application and compare it with the design required for applications running in the XS advanced-model run-time environment. It is important to bear in mind the fundamental differences in design between the XS classic an XS advanced run-time environments.

For example, the security concept underpinning XS advanced is different to the one used in XS classic. In XS classic, business users are also database users, to whom privileges can be assigned which grant or restrict access to data. In XS advanced, business users are not linked to database access privileges, and it is no longer possible to differentiate between users by assigning different privileges to database-related objects or schemas. Instead, authorization scopes are defined in the application layer.

You can also optimize some of the components used in the XS classic application to make an automatic migration easier and faster. Alternatively, you can rewrite the application, to take advantage of the new features XS advanced provides.

SAP HANA XS, Classic Model

In XS classic applications, data-intensive and model-based calculations must be close to the data and, therefore, need to be executed in the index server, for example, using SQLScript or the code of the specialized functional libraries. The presentation (view) logic runs on the client – for example, as an HTML5 application in a Web browser or on a mobile device. Native application-specific code, supported by SAP HANA Extended Application Services, classic model, can be used to provide a thin layer between the clients on one side, and the views, tables and procedures in the index server on the other side. A typical XS classic application contains control-flow logic based on request parameter; invokes views and stored procedures in the index server; and transforms the results to the response format expected by the client.

SAP HANA XS, Advanced Model

SAP HANA Extended Application Services advanced model (XS advanced) adds an application platform to the SAP HANA in-memory database. In the Cloud, this platform is provided by Cloud Foundry. An SAP-developed run-time environment is bundled with SAP HANA on-premise which provides a compatible platform that enables applications to be deployed to both worlds: the Cloud and on-premise. XS advanced is optimized for simple deployment and the operation of business applications that need to be deployed in both worlds. For this reason, the XS advanced programming model fully embraces the Cloud Foundry model and leverages its concepts and technologies. In areas where Cloud Foundry as an intentionally generic platform for distributed Web applications does not address relevant topics or offers choice, the XS advanced programming model provides guidance that is in line with the general Cloud programming model.

In simple terms, XS advanced is basically the Cloud Foundry open-source Platform-as-a-Service (PaaS) with a number of tweaks and extensions provided by SAP. These SAP enhancements include amongst other things: an integration with the SAP HANA database, OData support, compatibility with XS classic model, and some additional features designed to improve application security. XS advanced also provides support for business applications that are composed of multiple micro-services, which are implemented as separate Cloud Foundry applications, which combined are also known as Multi-Target Applications (MTA). A multi-target application includes multiple so-called "modules" which are the equivalent of Cloud Foundry applications.

Related Information

The XS Advanced Application-Migration Process [page 1044]
The XS Advanced Migration Assistant [page 1046]
The XS Application Migration Report [page 1048]
SAP HANA XS Advanced Migration Guide

12.1 The XS Advanced Application-Migration Process

The high-level steps required to migrate an XS classic application to the XS advanced run-time environment.

The XS Advanced Migration Assistant enables you to automate most of the tasks required to migrate an XS classic application to the XS advanced run-time environment. The following table lists the main steps required to complete the migration process:

> → Tip
>
> For more information about the migration assistant and a step-by-step guide to the migration process, see the *SAP HANA XS Advanced Migration Guide*.

Application Migration Steps

Steps	Action	Comments
1	Prepare for the migration operation	Some XS classic artifacts have been deprecated or are no longer used in XS advanced applications, for example: • Application Function Library (AFL) models (`*.aflpmml`) • Decision Tables (`*.decrule`) All deprecated object types used in the migrated XS classic application must be migrated manually, for example, using the view-migration tool included in SAP HANA Studio's *Modeler* perspective. For more information about this tool and the migration process, see the *SAP HANA XS Advanced Migration Guide*.
2	Prepare the SAP HANA source systems and required users	Since the application-migration process requires access to the SAP HANA XS classic Repository, it is necessary to configure the source system that hosts the Repository where the legacy XS classic application artifacts are located. **i Note** If your source system is older than SAP HANA SPS 11, you also need an "external parse system". Your target XS advanced system can be configured to play the role of external parser.
3	Migrate the XS classic application	Install and run the *XS Advanced Migration Assistant* and then start the migration process. **! Restriction** The target XS advanced system where you plan to deploy the migrated application must be running HANA 2.0. You cannot use the *XS Advanced Migration Assistant* to help you migrate your XS classic application to an SAP HANA 1.0 system, for example, SPS 11 or SPS 12.
4	Locate and read the report generated by the *XS Advanced Migration Assistant*.	The report generated by the *XS Advanced Migration Assistant* lists the problems found during the migration process; the problems must be fixed before running the migration again. Typically, the problems concern the following artifacts: • Deprecated attribute views • Incorrectly formatted calculation views • Security artifacts (`.hdbroles` and `.xsprivileges`) • Database artifacts (`.hdbtable`, `.hdbview`, etc.) • XS JavaScript artifacts (`.xsjs/.xsjslib`)

Steps	Action	Comments
5	Deploy the migrated application to XS advanced	Upload the successfully migrated design-time application artifacts to XS advanced. Build the new XS advanced application using the migrated artifacts. Set up the target system (including any required user-provided services). Deploy the migrated and built application to the XS advanced run time.

Related Information

The XS Advanced Migration Assistant [page 1046]
The XS Application Migration Report [page 1048]
Legacy Object Types not Supported in XS Advanced [page 1049]
SAP HANA XS Advanced Migration Guide

12.2 The XS Advanced Migration Assistant

The migration assistant helps you to migrate an XS classic application to the XS advanced model run time environment.

SAP HANA XS advanced model includes the XS Advanced Migration Assistant - a tool that is designed to help you migrate an XS classic application to the XS advanced run-time environment. After downloading and installing the XS Advanced Migration Assistant, you can use it to automate large parts of the migration process, for example, the conversion of many design-time artifacts from the XS classic file format and syntax to the syntax, format, and file extension required to build and deploy the migrated application in the XS advanced run-time environment.

> → Tip
>
> The *XS Advanced Migration Assistant* is available for download from the SAP Software Download Center listed in *Related Information* below. Search for the component *XSAC MIGRATION 1* in *SUPPORT PACKAGES AND PATCHES*.

The XS Advanced Migration Assistant scans the XS classic application and generates a report that indicates which design-time artifacts can be automatically migrated to the format required in XS advanced and which artifacts require attention before the migration process can be started. For example, scripted (text-based) calculation views, attribute views, and analytic views and privileges can all now be converted automatically into graphical calculation views using tools included in the Migration Assistant; some deprecated artifacts, for example, decision tables, must be manually converted using the XS classic admin tool SAP HANA Studio.

When all the issues listed in the migration report have been addressed and fixed, the XS Advanced Migration Assistant must be run again to perform the migration. During the migration process, the migration assistant converts the XS classic application's design-time artifacts to the appropriate format for XS advanced and places the migrated artifacts in the application-project containers required to be able to build and deploy the migrated application in the XS advanced run-time environment.

During the application-migration process, SAP HANA XS classic Repository objects are migrated to their equivalent counterparts in XS advanced and placed in the appropriate project container, for example:

- *web/*
  All user-interface objects and any static Web content
- *xsjs/*
  Application code, XS jobs and schedules, and HTTP destinations
- *db/*
  Database objects such as tables, views, procedures, and so on

The migration assistant puts objects whose destination cannot be automatically established into the *migration* container. For these objects, it is necessary to decide how each file must be processed. Objects in the *migration* container must be copied manually by a developer into the appropriate target container in the XS advanced application project, for example, *web*, *xsjs*, or *db*. This includes (but is not limited to) the following file types:

> **i Note**
>
> For more details about which design-time artifacts are copied to which container during the migration process, see the *SAP HANA XS Advanced Migration Guide* in *Related Information*.

- `.properties`
  In XS classic applications, the `.properties` file typically contains translation text. However, the XS Advanced Migration Assistant is not able to establish if a `.properties` file really is a translation-text file. The migration assistant also does not know to which target application container to copy the `.properties` file, for example, the *web*, *xsjs*, or *db* container.
- `.xml`
  The XS Advanced Migration Assistant checks for known structures in `.xml` files and, depending on what it finds and recognizes, copies the file to the appropriate application project container (*web*, *xsjs*, or *db*). If the migration assistant is not able to determine the type of XML file to be migrated, the `.xml` file is copied to the *migration* container from where it must be moved manually to the appropriate application-project container.
- `.json`
  The XS Advanced Migration Assistant checks for known structures in `.json` files and copies the file to the appropriate XS advanced application project container. If it is not possible to determine the type of file to be migrated, the migration assistant copies the file to the *migration* container from where it must be moved manually to the appropriate XS advanced application-project container.

Related Information

The XS Application Migration Report [page 1048]
SAP HANA XS Advanced Migration Guide
SAP Software Download Center (Logon required)

12.3 The XS Application Migration Report

The XS Advanced Migration Assistant generates a report, which lists details of the migration operation including any problems that need to be fixed.

The report generated by the *XS Advanced Migration Assistant* is in the form of an HTML file that is split into convenient categories, for example, “Security”, “Database”, or “XS JavaScript”. The information generated for each category includes a summary of the migration as well as a comprehensive list of any problems that occurred during the migration. The location of the migration report is specified in the options you can define before you run the *XS Advanced Migration Assistant*.

The source objects from the provided XS classic application's delivery units (DU) are exported from the source XS classic system, analyzed, migrated, and then written into the folder structure required for an XS advanced application.

> → Tip
>
> For more detailed information about the steps required to migrate XS classic applications to XS advanced, see the SAP HANA XS Advanced Migration Guide in *Related Information*.

The format of the information in the report matches the order and scope of the phases the migration process goes through, for example, preparation, security, database, etc. The following table lists the sections included in migration report generated by the *XS Advanced Migration Assistant* and provides a brief description of each section's contents:

XS Advanced Migration Report Elements

Report Section	Content
Preparation	A list of the XS classic artifacts that require **manual** migration **before** you start the migration of the XS classic application. For example, using the tools provided in SAP HANA Studio, you need to manually migrate decision tables and files generated by the Application Function Modeler (`*.aflpmml`) artifacts. After the manual migration of the listed artifacts is complete, it is necessary to run the XS Advanced Migration Assistant again.
Security	Since the security concept used in XS advanced is incompatible with the one used in XS Classic, manual migration steps are required in order to complete the migration of an XS classic application's security artifacts to XS Advanced. The *Warnings* section includes a list of the roles and access privileges that must be checked using the XS JavaScript command `getConnection()` in the XS JS `$.hdb` API. ! Restriction Due to the change in security concepts between XS classic and XS advanced, not all the features configured in the XS classic application's access-configuration file (`*.xsaccess`) can be migrated to the XS advanced application router's configuration file (`xs-app.json`).
Unsupported Features	A list of the XS classic objects that are no longer supported or have been discontinued. These are objects which the migration assistant cannot migrate automatically. Any problems listed in this section must be fixed by hand.

Report Section	Content
Database Artifacts	Information about design-time database artifacts which created warnings during the migration process. **i Note** The technical HDI user in XS advanced must be granted permission to access the SAP HANA database in order to be able to read and retrieve the XS classic database artifacts, for example, tables, views, procedures, and so on.
XS JavaScript Migration	A list of the statements in XS JavaScript code (for example, in the application's code `*.xsjs` and libraries `*.xsjslib` which cannot be migrated automatically; you must migrate this XS JavaScript code manually.
Translation	A list of the XS classic application's `*.hdbtextbundle` files, which contain UI texts and possibly existing translation texts. The contents of the `.hdbtextbundle` files are migrated to `.properties` files using the rules concerning references to (and from) `.hdbtextbundle` files.
Unknown File Types or Content	A list of the components that the *XS Advanced Migration Assistant* was not able to identify. For each unidentified component the file type is specified. You must decide whether these objects are relevant, and copy them into the appropriate target container directory if needed.
Objects not Migrated	Details of the objects which the *XS Advanced Migration Assistant* was not able to migrate, for example, because the objects are either not relevant (do not belong to the application project) or have been successfully migrated to another object type.

Related Information

SAP HANA XS Advanced Migration Guide

12.4 Legacy Object Types not Supported in XS Advanced

Some XS classic object types cannot be migrated to XS advanced.

The files listed in the following table cannot be migrated to XS advanced and are not moved to the XS advanced application project that is generated by the migration process.

i Note

Strictly speaking, not all of the object types listed in the following table are XS classic objects, for example: Eclipse-related files such as `*.project` and `*.classpath` or the Windows `*.bat` file, and Python `*.pyc` files. The one thing in common that **all** the files in this table share is that they are no longer relevant for XS advanced applications, and are not migrated by the XS Advanced Migration Assistant.

Non-Migratable Object Types

Object Type	Description
`*.xsapp`	An application-specific file in an XS classic repository package that defines the root folder of a native SAP HANA application. All files in that package (and any subpackages) are available to be called via URL.
`*.project`	An Eclipse project descriptor used in SAP HANA studio for . The `*.project` file is a design-time artifact that is stored in the SAP HANA XS classic repository.
`*.pyc`	Compiled Python files
`*.regiignore`	A file containing ignore of exclude patterns for XS Classic repository used by HANA Studio Repository Team Provider
`*.DS_Store`	MacOS Desktop Services Store used for custom file attributes, icons, etc. Should not be committed to any project repository.
`*.bat`	MS DOS Batch files
`*.classpath`	Eclipse JDT project Classpath configuration
`*.jdtscope`	Eclipse JDT project configuration
`*.gitignore`	A file containing ignore (exclude) patterns for Git client
`*.db`	A database file used by Sybase SQL Anywhere, SQLite, and others
`*.xsprivileges`	A file that defines a privilege that can be assigned to an SAP HANA Extended Application Services application, for example, the right to start or administer the application.
`*.xssecurestore`	The design-time file that creates an application-specific secure store; the store is used by the application to store data safely and securely in name-value form.
`*.xssqlcc`	A file that enables execution of SQL statements from inside server-side JavaScript code with credentials that are different to those of the requesting user

Related Information

SAP HANA XS Advanced Migration Guide

13 HDI Artifact and Build Plug-in Configuration

In XS advanced, design-time artifacts are distinguished by means of a unique file suffix that must be mapped to an HDI build plug-in.

The following example of an abbreviated HDI configuration file (`.hdiconfig`) illustrates how the design-time artifact types `.hdbcalculationview` and `.hdbcds` are mapped to their corresponding HDI build plug-in:

> **! Restriction**
>
> All "hdi*" file-suffixes (for example, "`.hdiconfig`" or "`.hdinamespace`") are reserved by the SAP HANA Deployment Infrastructure (HDI) and cannot be (re)configured.

Code Syntax

`.hdiconfig`

```
{
  "plugin_version" : "0.0.0.0",   // optional, defaults to 0.0.0.0
  "file_suffixes" : {
     "hdbcalculationview" : {
       "plugin_name" : "com.sap.hana.di.calculationview",
       "plugin_version": "2.0.0.0" //optional, default=top-level version
     },
     "hdbcds" : {
       "plugin_name" : "com.sap.hana.di.cds",
       "plugin_version": "2.0.0.0"
     },
     "<file suffix 2>" : {
      "plugin_name"   : "<plugin name>",
      "plugin_version": "<plugin_version>" // optional

     }
}
```

The values of the `plugin_version` fields are interpreted as the minimal required SAP HANA version, which follows the format `<major>.<minor>.<revision>.<patch>`, for example, `2.0.0.0` for SAP HANA 2.0 SPS00 revision 0 patch 0. The three level plugin version format used in SAP HANA 1.0, for example. `13.1.0`, is still valid; values in the old format will be internally converted to the appropriate four level format for SAP HANA 2.0 revisions.

In SAP HANA 1.0 there was no "top-level" `plugin_version` entry, and the "bottom-level" `plugin_version` entries were mandatory.

From SAP HANA 2.0, the version of all plugins shipped with SAP HANA is the same as (and equal to) the SAP HANA version. Consequently, there should normally be no need for the bottom level `plugin_version` entries, since the result of the compatibility check would depend on the highest entry in use only. To summarize, if no particular minimum SAP HANA version is required, it is sufficient to provide the top-level entry or no entry at all. However, if present, bottom-level entries override the top-level entry or the default `plugin_version` of "`0.0.0.0`", respectively.

The following table lists in alphabetical order the design-time artifacts you can develop and deploy with the SAP HANA Deployment Infrastructure (HDI). For more information about the syntax required for a particular type of design-time artifact and the configuration of the corresponding build plug-in, choose the artifact type in the following table.

> **i Note**
>
> Some plug-in libraries are not available by default, for example, for public and `_SYS_BIC` synonyms. The list of plug-in libraries that are available by default in an HDI container can be viewed and, if necessary, configured by the HDI Container administrator, as described in *Maintaining HDI Containers* in the *SAP HANA Administration Guide*. For more information, see *Related Information* below.

Related Information

HDI Artifact Types and Build Plug-ins [page 1052]
Maintaining HDI Containers (SAP HANA Admin Guide)

13.1 HDI Artifact Types and Build Plug-ins

The XS advanced deployment infrastructure supports a wide variety of database artifact types, for example, tables, types, views, and procedures.

The following table lists in alphabetical order the design-time artifacts you can develop and deploy with the SAP HANA Deployment Infrastructure (HDI). For more information about the syntax required for a particular type of design-time artifact and the configuration of the corresponding build plug-in, choose the artifact type in the following table.

> **i Note**
>
> Some plug-in libraries are not available by default, for example, for public and `_SYS_BIC` synonyms. The list of plug-in libraries that are available by default in an HDI container must be configured by the HDI Container administrator, as described in *Maintaining HDI Containers* in the *SAP HANA Administration Guide*. For more information, see *Related Information* below.

HDI Design-Time Artifact Types and Run-Time Plug-Ins

Artifact Type	Artifact Suffix	Description
text, copy only	.? (.txt, ...)	Transform an arbitrary design-time resource into a deployed object
AFL Lang Procedure	`.hdbafllangprocedure`	Transform a design-time Application Function Library (AFL) language procedure resource into a corresponding database procedure object.

Artifact Type	Artifact Suffix	Description
Analytic Privilege	`.hdbanalyticprivilege`	Transform a design-time XML-based analytic-privileges resource into a analytic-privileges object in the database
Application Time-Period Table	`.hdbapplicationtime`	Transform a design-time application time-period table into a database table object with application-time period
Calculation View	`.hdbcalculationview`	Transform a design-time calculation view description into a set of view database objects
Constraint	`.hdbconstraint`	Transform a design-time constraint into a constraint on database tables
Core Data Services (CDS)	`.hdbcds`	Transform a design-time SAP HANA CDS resource into the database objects defined in the CDS document
CSV	-	Source file containing comma-separated values (CSV) for data import with `.hdbtabledata`
DocStore Collection	`.hdbcollection`	Transforms a design-time document-collection resource into a JSON collection database object
Drop/Create Table	`.hdbdropcreatetable`	Transforms a design-time table resource into a table database object. See also `.hdbtable`
Flowgraph	`.hdbflowgraph`	Transform a design-time flowgraph description into a corresponding set of database procedure or task objects
Full-Text Index	`.hdbfulltextindex`	Transform a design-time full-text index resource into a full-text index on a database table.
Function	`.hdbfunction`	Transform a design-time function resource into a function database object
Graph Workspace	`.hdbgraphworkspace`	Transform a design-time graph-workspace resource into a graph-workspace object in the database
Index	`.hdbindex`	Transform a design-time index resource into an index on a database table
Map Reduce Job ***	`.hdbmrjob`	Transform a design-time Hadoop map-reduce job resource into a virtual-function-package database object. See also `.jar`. **Note** The `.hdbmrjob` artifact has been deprecated; use Virtual Package (`.hdbvirtualpackage`) instead.

Artifact Type	Artifact Suffix	Description
Jar	`.jar`	Transform a design-time Hadoop map-reduce job resource into a virtual-function-package database object. See also `.hdbmrjob`.
Library	`.hdblibrary`	Transform a design-time library resource into a library database object
Logical Schema	`.hdblogicalschema`	Transforms a design-time logical-schema definition into database objects that can be consumed by synonyms and so on.
Migration Table	`.hdbmigrationtable`	Transform a design-time table resource into a table database object for migration purposes
Procedure	`.hdbprocedure`	Transform a design-time procedure resource into a procedure database object
Projection View	`.hdbprojectionview`	Transforms a design-time projection-view definition into a database object
Projection View Configuration	`.hdbprojectionviewconfig`	An explicit configuration of the projection view's target. A projection view configuration file can contain multiple configurations
(Table Data) Properties	`.properties`	Insert data from files referenced in the ".properties" file format into database tables which are managed by SAP HANA HDI. See also CSV and Table Data
Public Synonym	`.hdbpublicsynonym`	Create public synonyms that refer to database objects located in the target schema of the current container
Replication Task	`.hdbreptask`	Transform a design-time replication task description into a corresponding set of database procedure or task operations
Result Cache	`.hdbresultcache`	Transform a DDL-based definition of a result cache into the corresponding catalog object
Role Configuration	`.hdbrole`	Transform a design-time role resource into a run-time role object.
Search Rule Set	`.hdbsearchruleset`	Creates search configurations that can be consumed with a built-in database procedure
Sequence	`.hdbsequence`	Transforms a design-time sequence resource into a sequence database object
SQL View	`.hdbview`	Transforms a design-time view resource into an SQL view database object.
Statistics	`.hdbstatistics`	Transforms a design-time statistics resource into a statistics object on a database table.

Artifact Type	Artifact Suffix	Description
Structured Privilege	`.hdbstructuredprivilege`	Transforms a design-time DDL-based structured privilege resource into a structured privilege object
Synonym	`.hdbsynonym`	Transforms a design-time synonym definition into a database synonym object.
Synonym Configuration	`.hdbsynonymconfig`	An explicit (but optional) configuration of the synonym's target; the configuration can also be included in the synonym definition (`.hdbsynonym`).
_SYS_BIC Synonym	`.hdbsysbicsynonym`	Create synonyms in the _SYS_BIC schema that refer to database objects located in the target schema of the current container
System Version Table	`.hdbsystemversioning`	Transforms a design-time, system-versioned table that refers to a current and history table into a system-versioned table database object
Table	`.hdbtable`	Transforms a design-time table resource into a table database object. See also `dropcreatetable`.
Table Data	`.hdbtabledata`	Insert data from other files (for example, CSV, `.properties`, or `.tags` files) into database tables.
Table Data Properties	`.properties`	Insert data from files referenced in the ".properties" file format into database tables which are managed by SAP HANA HDI. See also `.hdbtabledata` and `.tags`.
Table Type	`.hdbtabletype`	Transforms a design-time table type resource into a table type database object.
Tags	`.tags`	Insert data from other files (for example, CSV, `.properties`, or `.tags` files) into database tables.
Text Analysis Configuration	`.hdbtextconfig`	Customize the features and options to be used for text analysis
Text Analysis Dictionary	`.hdbtextdict`	Define custom entity types and entity names for use with text analysis
Text Analysis Extraction Rule	`.hdbtextrule`	Specify rules used to extract complex entities and relationships during text analysis
Text Analysis Extraction Include	`.hdbtextinclude`	Define rule definitions to be used in one or more top-level, extraction-rule sets for text-analysis
Text Analysis Extraction Lexicon	`.hdbtextlexicon`	Define lists of words for use in sets of text-analysis rules

Artifact Type	Artifact Suffix	Description
Text Mining Configuration	`.hdbtextminingconfig`	Customize the features and options used for text mining
Trigger	`.hdbtrigger`	Transforms a design-time trigger resource into a trigger on a database table.
View	`.hdbview`	Transforms a design-time view resource into an SQL view database object.
Virtual Function	`.hdbvirtualfunction`	Transform a design-time virtual function resource into a virtual function database object
Virtual Function Configuration	`.hdbvirtualfunctionconfig`	
Virtual Package	`.hdbvirtualpackage`	Transform a design-time Hadoop Map Reduce Job (`.hdbvirtualpackagehadoop`) or SparkSQL resource (`.hdbvirtualpackagesparksql`) into a virtual-package database object
Virtual Procedure	`.hdbvirtualprocedure`	Transform a design-time virtual-procedure resource into a virtual-procedure database object
Virtual Procedure Configuration	`.hdbvirtualprocedureconfig`	
Virtual Table	`.hdbvirtualtable`	Transform a design-time virtual table resource into a virtual table database object
Virtual Table Configuration	`.hdbvirtualtableconfig`	

> **i** Note
>
> * AFL Lang Procedure: Application function library language procedure
>
> ** CSV: Comma-separated-values file
>
> *** Map Reduce Job: Deprecated, use Virtual Package instead

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference

14 The XS Command-Line Interface

A list of all the categories and areas covered by the `xs` command-line interface (CLI).

The XS CLI enables you to maintain not only the applications that are deployed to the XS advanced run-time environment, but also the run-time environment itself, and the users who access and use it. The XS advanced command-line client is included not only in the full XS advanced run-time instance installed on the server but in a separate "client" package for installation on a remote machine.

→ Tip

The XS advanced command-line client is available for download on SAP Service Marketplace at the following location for those people with the required S-User ID:

Service Marketplace > *Software Downloads [Downloads]* > *SUPPORT PACKAGES & PATCHES* > *By Alphabetical Index (A-Z)* > *H* > *SAP HANA PLATFORM EDITION*

- *SAP HANA PLATFORM EDIT. 1.0* > *XS ADVANCED RUNTIME* > *XS RUNTIME 1*
- *SAP HANA PLATFORM EDITION 2.0* > *XS RUNTIME 1*

Usage

```
xs <command> [<ARGUMENTS>] [<OPTIONS>]
```

To display information about a specific `xs` command, run the following command:

```
xs help <command>
```

To display information about **all** available `xs` commands, run the following command:

```
xs help -a
```

Environment Variables

Sample Code

```
XS_TRACE=true
```

XS CLI Environment Variables

Variable	Value	Description
XS_TRACE	`true`	Print verbose information to `stdout`
	`path/to/trace.log`	Append verbose information to the specified log file
XSCLIENT_CONTEXTFILE	`path/to/config.cfg`	Override the path to the default configuration file

Global XS Command-Line Options

XS CLI Environment Variables

Variable	Value	Description
`--verbose`	-	Print verbose information to `stdout`
`--context-file`	`path/to/config.cfg`	Override the path to the default configuration file

Related Information

SAP Software Download Center (Logon required)
SAP HANA Administration Guide

14.1 The XS Command Overview

A list of the categories covered by the `xs` command-line interface (CLI) for SAP HANA XS, advanced model.

Command Overview

XS Command Overview

Command Category	Description
Logon and setup	User logon, viewing user-organization (and space) targets, and setting API URLs
Application Management	Maintain SAP HANA XS applications: listing, deploying, starting, stopping, staging, [...]

Command Category	Description
Services Management	Maintain SAP HANA XS services: listing, creating, deleting, binding, updating, [...]
Organizations	Maintain user organizations: create, list, rename, delete, [...]
Spaces	Manage user spaces: create, list, rename, delete, [...]
Domains	Manage domains
Certificates	Manage certificates
Routes	Maintain application routes: create, list, map, unmap, delete, [...]
Build Packs	Maintain application build-packs: create, list, update, rename, delete, [...]
Run-time Environments	Maintain XS run-times: create, list, display information, search, update, delete, [...]
Tasks	Maintain XS application-related tasks
User Administration	Maintain SAP HANA users: create, list, purge, delete, set/unset organizations and spaces, [...]
Administration	Maintain XS application traces and backups
Configuration	Set and maintain environment variables
Tenants	Manage and maintain tenant databases and their mapping to organizations and spaces.
Blob Store	Manage and maintain the contents of the blob store
Advanced	Perform advanced administration tasks in SAP HANA XS advanced model.
Other Commands	Display information about the SAP HANA XS version, CLI, and system
Plug-ins	Additional commands as plug-ins; install product archives and deploy multi-target applications (MTA)

Related Information

The XS Command-Line Interface [page 1057]
The XS Command-Line Interface Reference

15 SAP Web IDE for SAP HANA Reference

Reference information about SAP Web IDE for SAP HANA.

The following section describes the various development tools provided by SAP Web IDE for SAP HANA and the main tasks required for developing SAP HANA based applications.

Learn more at the SAP Web IDE for SAP HANA product page.

What's New in SAP Web IDE for SAP HANA

Learn about SAP Web IDE new features.

Product Overview [page 1060]

Understand how to use SAP Web IDE and learn about its key features and benefits for developers.

Getting Started [page 1062]

Learn how to set up and start SAP Web IDE.

Developing Multitarget Applications [page 1078]

See how SAP Web IDE supports key stages of the development life cycle.

Customizing Your Project [page 1235]

Understand how to customize your project.

Troubleshooting [page 1276]

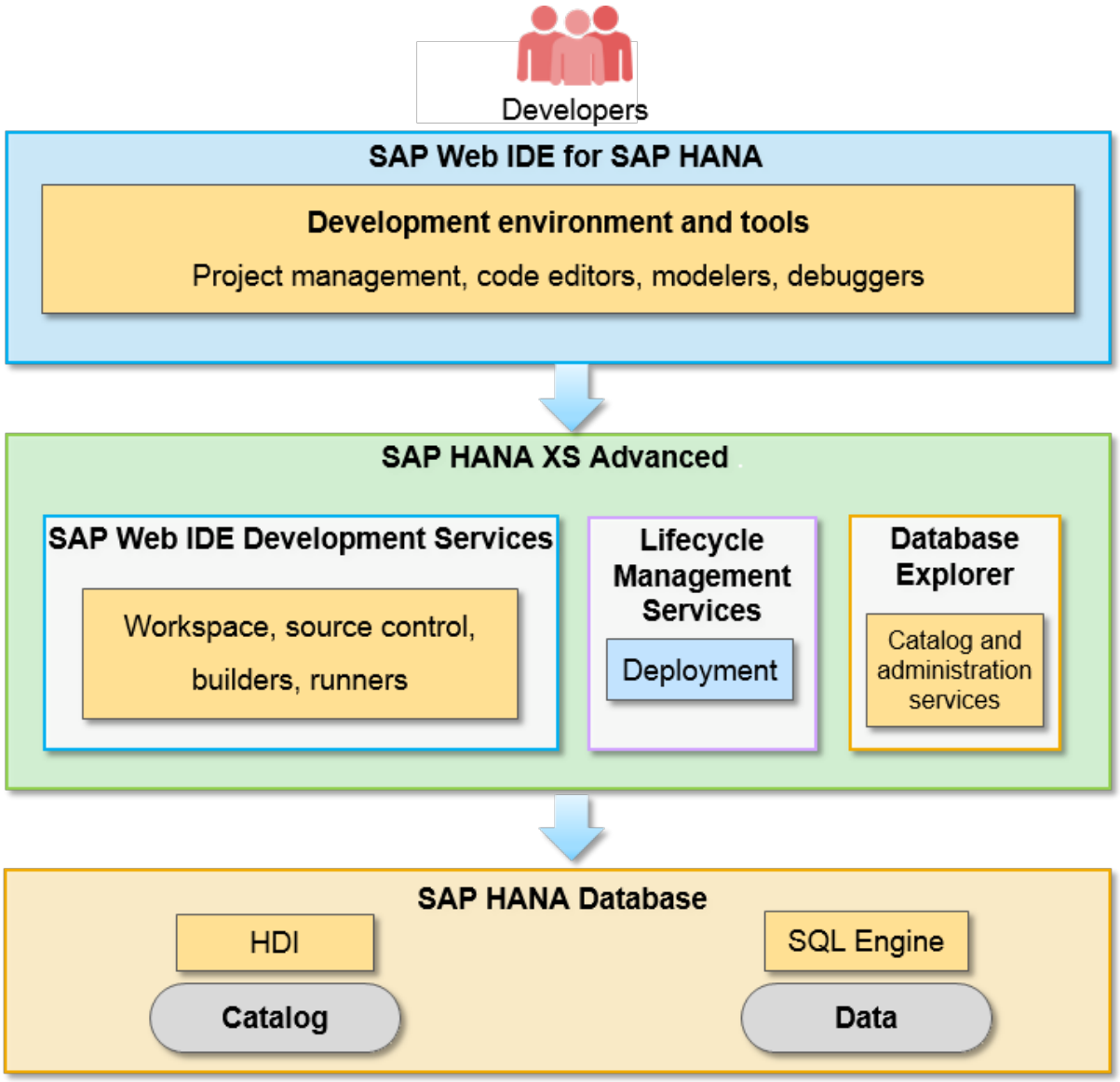
Common troubleshooting issues in SAP Web IDE.

15.1 Product Overview

SAP Web IDE for SAP HANA is a browser-based integrated development environment (IDE) for the development of SAP HANA-based applications comprised of web-based or mobile UIs, business logic, and extensive SAP HANA data models.

SAP Web IDE supports developers who use SAP HANA Extended Application Services, advanced model (XS Advanced), by providing a variety of tools, such as syntax-aware editors for code and SAP HANA artifacts, graphical editors for CDS data models and calculation views, as well as inspection, testing and debugging tools.

SAP Web IDE works in conjunction with the SAP HANA database explorer, the SAP HANA deployment infrastructure (HDI), the Application Lifecycle Management tools (ALM) and the XS Advanced runtime platform, as shown in the following diagram.



Features and Benefits

SAP Web IDE for SAP HANA and the database explorer enhance the development experience by providing the following features:

Integrated Workspace and Project Management

- The latest tool versions are always available without additional installation
- Full workspace management on the server
- Integrated Git-based version control
- A dedicated template and wizards for multi-target application projects

Development Tools

- For SAP HANA database artifact development:
  - Graphical modelers for complex artifacts, such as data models or calculation views
  - An SQL console and an MDX console in the database explorer
  - Integrated browsing of the database catalog in the database explorer
  - Syntax highlighting and content assistance for selected artifacts
  - Integrated performance analysis tools for SQL script and calculation views
- For Node.js and Java development
  - Integrated debugger
  - Integrated test executions and results with source code navigation
  - Integrated test coverage
  - Direct application start in the browser
- For UI development
  - HTML5, JavaScript, and SAPUI5-aware syntax and content assistance
  - Direct application start in the browser

Build, Run, and Deploy

- Incremental build support
- Structured console output and logging
- Automatic creation of development sandboxes on SAP HANA (HDI containers)
- Automatic provisioning of HDI container services
- Generation of deployment archives

Supported Browsers

SAP Web IDE supports the following browsers:

- Chrome (latest version)
- Firefox (latest version)
- Internet Explorer 11 on Windows
- Edge (latest version) on Windows
- Safari (latest version) on macOS

.

15.2 Getting Started

Start working with SAP Web IDE for SAP HANA by getting a user account, logging on, and getting familiar with the environment.

Prerequisites

- Your SAP HANA administrator has installed SAP Web IDE for SAP HANA (SAP Web IDE).

- Your SAP HANA administrator has granted you the permissions required to access SAP Web IDE, and performed the necessary configuration tasks, as described in Post Installation Administration Tasks.

Procedure

1. Navigate to the URL of the installed SAP Web IDE, and log on to XS Advanced with your user credentials. The SAP Web IDE window opens.
2. Get familiar with the environment, and learn how to perform the basic tasks.

> **i** Note
>
> After a period of inactivity of 80 minutes, you will be logged off automatically. To resume work, you will need to log on again.

Related Information

SAP Web IDE for SAP HANA - Installation and Upgrade Guide

15.2.1 Navigating SAP Web IDE

In SAP Web IDE, you can use the menu options and toolbar icons to perform various operations.

SAP Web IDE contains two perspectives: development and database explorer. Each perspective has its own set and layout of menus, toolbars and panes, as described below.

Development Perspective

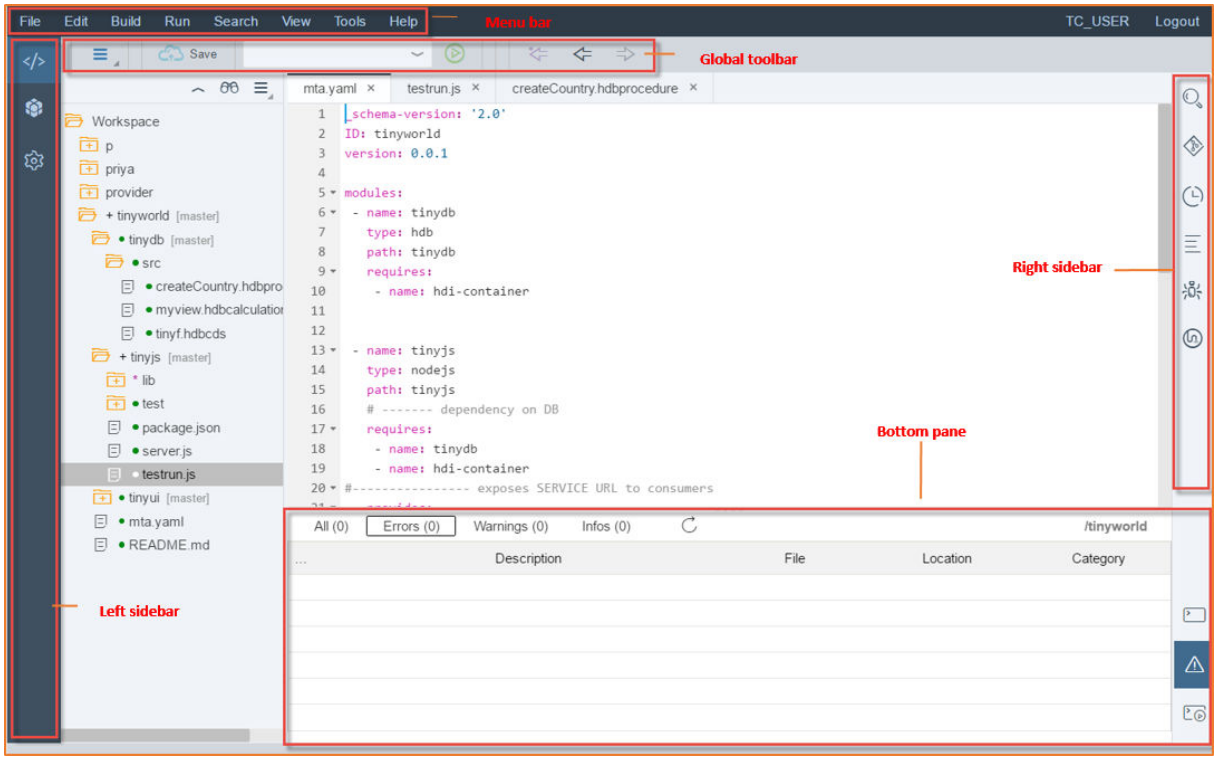
Menu bar - Provides access to all operations available in SAP Web IDE.

Global toolbar - Depending on the item that is activated in the workspace, you can choose from the icons in the global toolbar (icons of actions that are not applicable are grayed out).

Left sidebar - Use the buttons to switch between the development workspace, the database explorer, and user preferences.

Right sidebar - Use the buttons to switch between the different panes available in SAP Web IDE (for example, Git pane, Outline pane, and so on).

Bottom pane - Use the toggle buttons to display the console or the problems view.



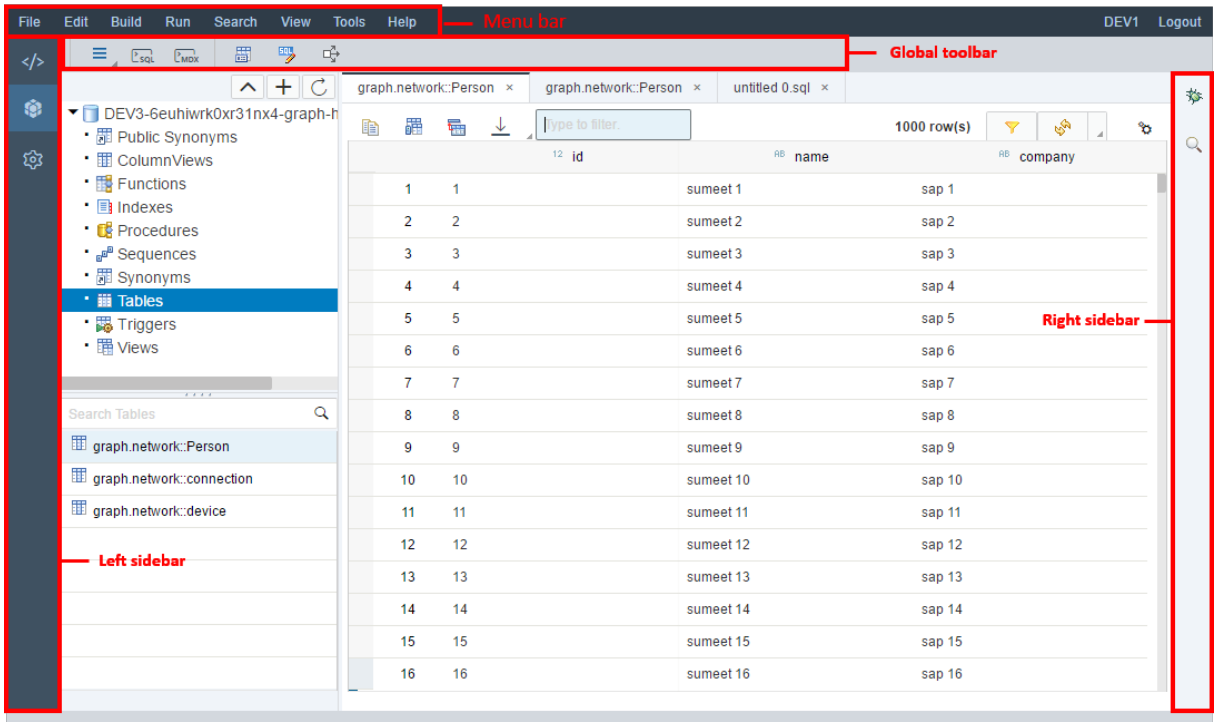
Database Explorer Perspective

Menu bar - Only the Help menu bar item is activated for the database explorer.

Global toolbar - Depending on the item that is activated in the workspace, you can choose from the icons in the global toolbar (icons of actions that are not applicable are grayed out).

Left sidebar - Use the buttons to switch between the development workspace, the database explorer, and user preferences.

Right sidebar - Use the buttons to switch between the different panes available in the database explorer.



Related Information

Working with the SAP HANA Database Explorer [page 75]

15.2.2 Setting User Preferences

You can set your preferences for working in SAP Web IDE.

Procedure

1. To open the *Preferences* perspective, in the left sidebar, choose (Preferences).
2. Select the type of preference that you want to change.

For example, you can set the following SAP Web IDE preferences:

Preferences	More Information
Code check settings to enable and disable inline code validation. The project-level appearance settings are used to validate code and flag messages.	Configure Code Checking [page 1247]
Code completion settings to enable inline code completion and comment completion.	Configure Code Completion [page 1246]
Code editor settings to select the theme to be used by the code editor and the font size. You can also choose to have the changes in all open documents saved automatically at preset intervals.	Configure the Code Editor [page 1238]
Set up Git for source control.	Set Up Git [page 1257]
Customize SAP Web IDE keyboard shortcuts.	Customize Keyboard Shortcuts [page 1069]
Delete or ignore workspace persistence.	If you start SAP Web IDE after you have been logged out, the system restores the latest status of the workspace with its preference settings and all the editors that have been opened. If you want to reload SAP Web IDE while ignoring this persistence feature, for example, because an editor cannot be loaded or is frozen, add the parameter **`settings=ignore`** to the URL and refresh your browser. The persistence information is not deleted (meaning that when you remove the parameter from the URL and refresh your browser, SAP Web IDE restores the latest status of the workspace and the editors).

15.2.3 Enable SAP Web IDE Extensions

You can enable SAP Web IDE extensions to use in application development.

Context

This is a test for dependencies

SAP Web IDE includes extensions that are not enabled by default.

> ⚠ Caution
>
> An SAP Web IDE extension extends the functionality of SAP Web IDE and provides new capabilities to your IDE. Such extensions) have full privileges to access your browser, your computer, and any data stored in

your SAP Web IDE workspace or on SAP BTP, including the ability to read and modify your private and organizational data.

Extensions that are not provided by SAP are under the responsibility of the extension author and may have different privacy policies, terms of use, or quality levels. You can enable extensions that are not provided by SAP and use them at your own risk. It is strongly recommended that you enable only extensions that you trust. At any time and without warning, SAP reserves the right to remove, disable, or uninstall extensions that are not provided by SAP from your environment.

To learn more about extension development, see the SAP Web IDE SDK.

If you would like to use extensions, perform the following steps:

Procedure

1. To open the *Preferences* perspective, in the left sidebar, choose (Preferences).
2. Under *Workspace Preferences*, choose *Extensions*.
3. In the *Extensions* pane to the right, select the toggle button for the extension you want to enable.
4. Choose *Save*.
5. Refresh your browser.

Results

You can now use the enabled extensions in your projects.

15.2.4 Keyboard Shortcuts

You can use keyboard shortcuts to perform actions in SAP Web IDE.

The following shortcuts are available in SAP Web IDE:

> **i** Note
>
> - The majority of these shortcuts are available only for English language keyboards.
> - On Mac OS keyboards, the `Alt` key is also called the `Option` key.

You can modify the SAP Web IDE predefined keyboard shortcuts as described in Customize Keyboard Shortcuts [page 1069].

Action	Microsoft Windows Keyboard Shortcut	Mac OS Keyboard Shortcut
New file	`Ctrl` + `Alt` + `N`	`Command` + `Alt` + `N`

Action	Microsoft Windows Keyboard Shortcut	Mac OS Keyboard Shortcut
New folder	Ctrl + Alt + Shift + N	Command + Alt + Shift + N
New project	Ctrl + Alt + Shift + O	Command + Alt + Shift + O
Close file	Alt + W	Alt + W
Close all files	Alt + Shift + W	Alt + Shift + W
Save file	Ctrl + S	Command + S
Save all files	Ctrl + Shift + S	Command + Shift + S
Show Code Completion Suggestions	Ctrl + Space	Ctrl + Space
Undo	Ctrl + Z	Command + Z
Redo	Ctrl + Y	Command + Y
Cut	Ctrl + X	Command + X
Copy	Ctrl + C	Command + C
Paste	Ctrl + V	Command + V
Rename file or folder	F2	F2
Delete	Del	Del
Move to the tab on the right	Alt + R	Alt +
Move to the tab on the left	Alt + Q	Alt + Q
Navigate back to the previous file	Ctrl + Alt + R	Command + Alt + R
Navigate forward	Ctrl + Alt + Y	Command + Alt + Y
Navigate to the file that was edited last	Ctrl + Shift + 9	Command + Shift + 9
Show/Hide all characters	Ctrl + I	Command + I
Toggle line comment	Ctrl + /	Command + /
Toggle line comment (German language keyboard)	Alt + 7	Alt + H
Toggle block comment	Ctrl + Shift + /	Command + Shift + /
Toggle block comment (German language keyboard)	Ctrl + Shift + 7	Command + Shift + 7
Add todo comment	Ctrl + Alt + T	Command + Alt + T
Indent line	Tab	Tab
Outdent line	Shift + Tab	Shift + Tab
Move lines up	Alt + Up Arrow	Shift + Tab

Action	Microsoft Windows Keyboard Shortcut	Mac OS Keyboard Shortcut
Move lines down	Alt + Down Arrow	Alt + Down Arrow
Copy lines up	Alt + Shift + Up Arrow	Alt + Shift + Up Arrow
Copy lines down	Alt + Shift + Down Arrow	Alt + Shift + Down Arrow
Beautify file format	Ctrl + Alt + B	Command + Alt + B
Generate JSDoc Comment	Ctrl + Alt + J	Command + Alt + J
Goto JavaScript definition	Ctrl + Alt + G	Command + Alt + G
Run	Alt + Shift + R	Alt + Shift + R
Run without frame	Ctrl + Alt + Shift + R	Command + Alt + Shift + R
Find	Ctrl + F	Command + F
Find and replace	Ctrl + H	Command + H
Find references	Ctrl + Alt + W	Ctrl + Alt + W
Advanced repository search	Ctrl + Shift + F	Command + Shift + F
Maximize/Restore Active Editor	Ctrl + M	Command + M
View console	Ctrl + Shift + M	Command + Shift + M
View Git pane	Ctrl + Shift + V	Command + Shift + V
View outline	Ctrl + Shift + U	Command + Shift + U
Open Preferences perspective	Ctrl + .	Command + .
Open Resource	Ctrl + Shift + R	Command + Shift + R
Go to line	Ctrl + L	Command + L
Refactor	Alt + J	Alt + J
Open i18n	Alt + I	Alt + I

15.2.4.1 Customize Keyboard Shortcuts

You can modify the SAP Web IDE predefined keyboard shortcuts according to your preferences by recording them.

Context

Follow the instructions below to change a keyboard shortcut.

> ⚠ Caution
>
> Using certain browser shortcuts can trigger the assigned browser action. For example, the shortcuts below are already assigned in the Chrome browser and should not be used as SAP Web IDE shortcuts.
>
> - `Ctrl` + `N`
> - `Ctrl` + `T`
> - `Ctrl` + `W`
> - `Ctrl` + `Shift` + `N`
> - `Ctrl` + `Shift` + `T`
> - `Ctrl` + `Shift` + `W`

Procedure

1. In the *Preferences* area, choose *Keyboard Shortcuts*.
2. In the *Shortcut* column, double-click a shortcut that you want to modify.
3. Record your new shortcut by pressing a key combination on your keyboard.

   > i Note
   >
   > You can revert to the SAP Web IDE default setting by clicking the *Revert* button.

4. Choose the *Save* button.

Results

The new customized keyboard shortcuts can now be used.

To use your customized shortcuts in SAP Web IDE, select the *User-Defined* option and then choose the *Save* button. You can switch between the SAP Web IDE default shortcuts and your customized shortcuts without losing your shortcut definitions. If needed, you can remove all your modified shortcuts by choosing the *Clear User-Defined* button. This action restores the SAP Web IDE default shortcuts and deletes your customized shortcuts.

Related Information

Keyboard Shortcuts [page 1067]

15.2.5 Search Options

SAP Web IDE's search options include a simple search within an open file in the code editor, advanced search and replace across multiple files in a project, and a search to find references.

15.2.5.1 Search in the Workspace

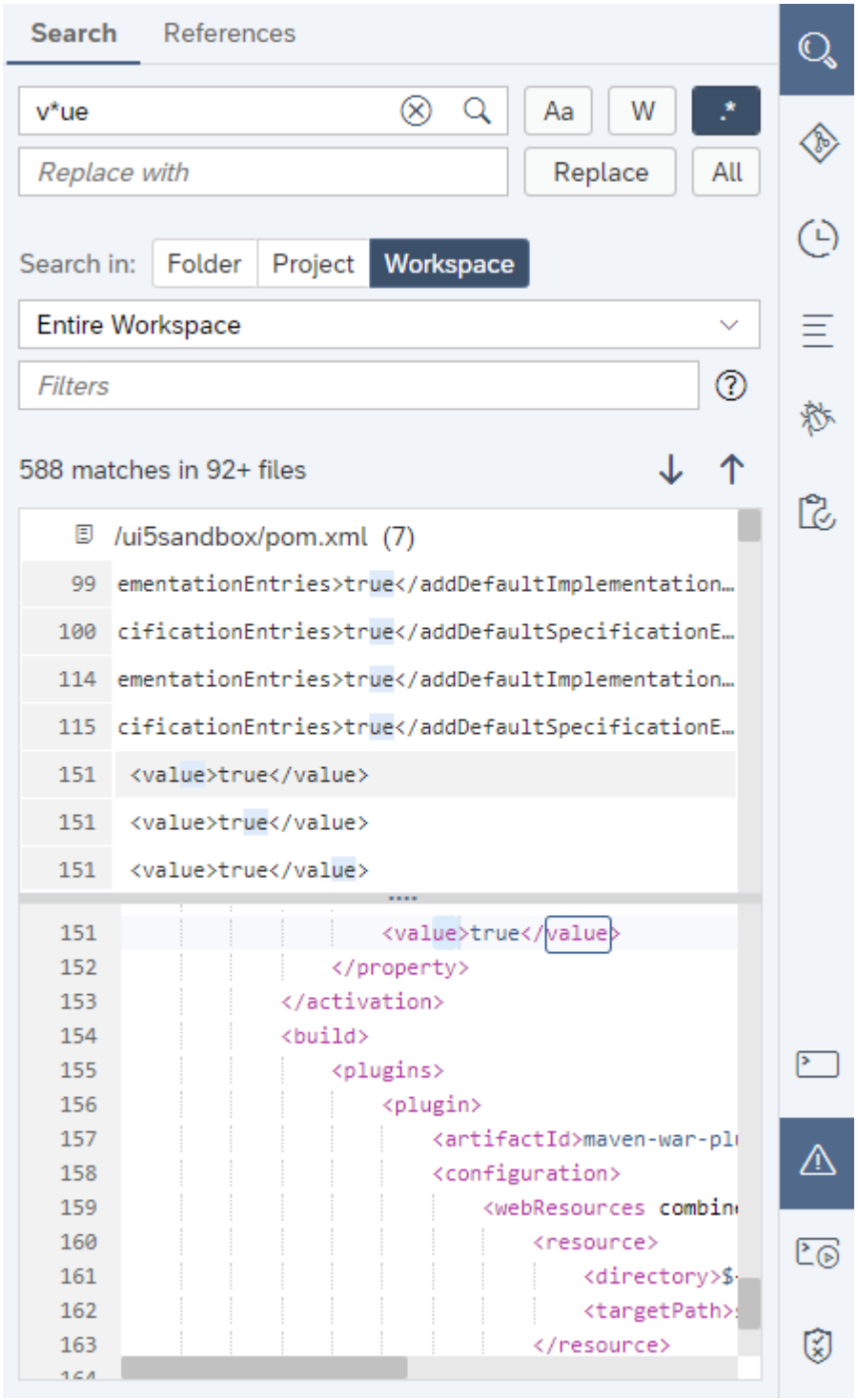
Search for a string within a folder, a project, or your workspace.

Context

To determine the success of your search, use the preview to quickly scan the results, and refine your search further as needed.

Procedure

1. Open the *Search* pane by choosing the search icon in the right sidebar.



2. Enter the string you want to find and click `Enter`.

> **i Note**
>
> Use abbreviated uppercase notation to search for words in CamelCase. For example, enter GTZ to find GetTimeZone.

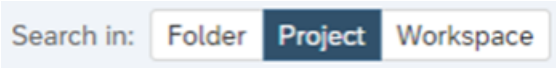
3. If needed, use the following to narrow your search:

a. Aa to perform a case-sensitive search.

b. W to search for whole words.

c. .* to search for regular expressions. To learn about which regular expressions are supported by SAP Web IDE, see here .

4. If needed, use Folder, Project, or Workspace to define the scope of your search.



a. If you click *Folder*, the dropdown list contains a list of all the folders in the selected project.

b. If you click *Project*, the dropdown list contains a list of all the projects in the workspace.

> **i Note**
>
> Selecting a project or folder in the workspace sets the search scope in the *Search* pane to that selected project or folder.

5. If needed, enter wildcards to filter your results.

- Use an asterisk (`*`) as a substitute for a part of a file or folder name.
- Use a backslash (`/`) to search only in specific folders.
- Use an exclamation mark (`!`) to exclude a file name or folder.
- Use CamelCase abbreviations to search only in files that have CamelCase names. For example, entering BCP will search in these files:
  - `…/BreadCrumbsPage.js`
  - `…/BlockConnectionProtocol.java`
  - `…/LogBigClockPivotData.xml`

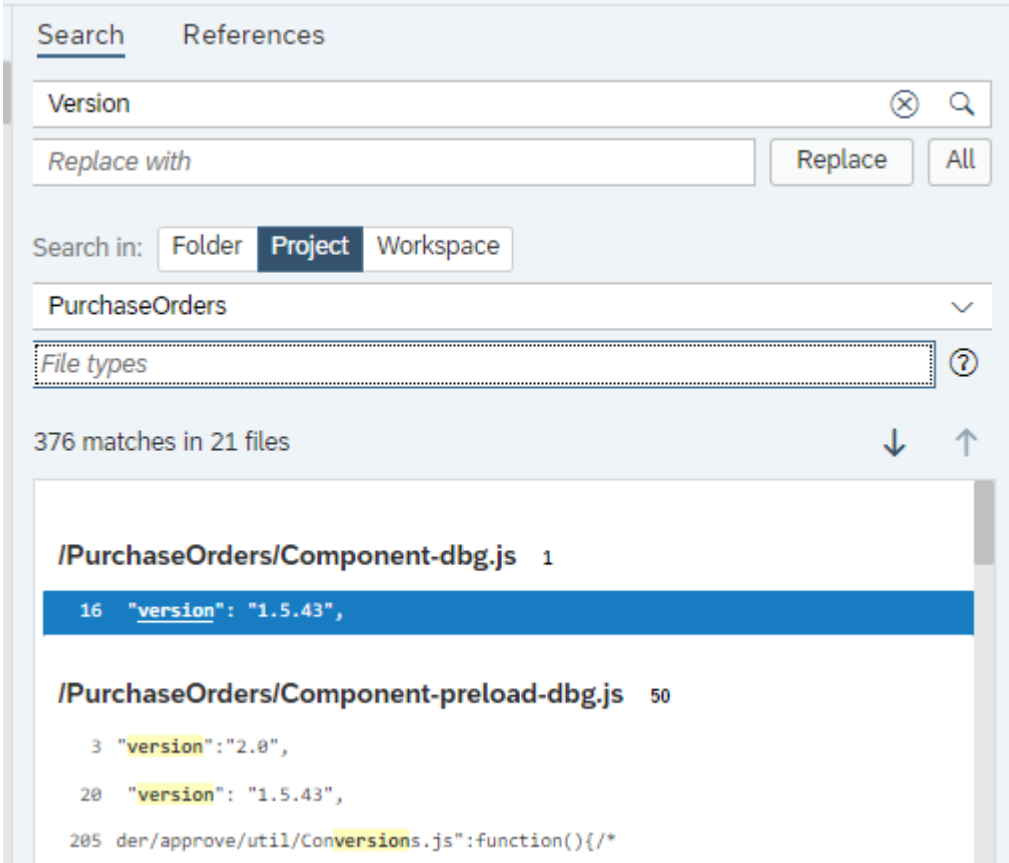
6. In the results list at the bottom of the *Search* pane, you see search results, organized according to file.
7. Click a row to see the file's preview in the bottom part of the pane.
8. When you have identified the file and row that you were looking for, double-click the specific search result to open the file with the appropriate row highlighted.

15.2.5.2 Replace Strings Across Multiple Files

Perform an advanced find and replace across multiple files with the *Search* pane.

Procedure

1. Open the *Search* pane by choosing the search icon in the right sidebar.



2. Define the scope of your search:Folder, Project, or Workspace.
   a. If you select *Folder*, the dropdown list contains a list of all the folders in the selected project.
   b. If you select *Project*, the dropdown list contains a list of all the projects in the workspace.

> **i Note**
>
> The project or folder selected in the workspace are displayed by default in the *Search* pane.

3. Enter the string you want to find and press `Enter`.
4. In the *Replace with* field, enter the replace text and perform one of the following:
   - Click *Replace* to select all the instances that should be replaced. As each instance is replaced, it disappears from the results list. Skip all results that you do not want to replace.
   - Click *All* to perform a global replace of all results in the list.

> **i Note**
>
> You cannot undo the global replace operation.

15.2.5.3 Find and Replace in an Open File

Perform a simple find and replace within a single open file from the code editor.

Context

These searches are limited to the file that is currently in view. To search and replace across multiple files, perform an advanced search. See Search in the Workspace [page 1071].

Procedure

1. Open the file that you want to perform a simple string search in.
2. Select *Search* > *Find and Replace*.
3. Do one of the following:
   - To perform a basic search only, enter the search string in the search field (the first field). Use the adjacent up and down arrows to find instances previous to or following the current cursor location. Use the icons to further limit the search for:
     - Regular expressions
     - Case-sensitive results
     - Whole words only
     - Search in selection
   - To replace text, enter both a search string and a *Replace with* string. To search and replace one by one, click *Replace*. If other instances exist, the next one will be selected. To automatically search and replace all instances, click *All*.

     > **i Note**
     >
     > You cannot undo the global replace operation.
4. If you replaced strings, save the file.

15.2.6 Working with Files and Folders

Use keyboard shortcuts and context menus to work with files and folders.

You can open multiple files in the SAP Web IDE code editor, and you can navigate between them. Before closing files, save or discard your changes. You can cut, copy, paste, and rename files and folders.

> **i Note**
>
> For keyboard shortcuts, see Keyboard Shortcuts [page 1067].

Action	Context Menu or Toolbar Icon
Open selected files in the workspace.	*Open with* > *Code Editor*
Search and open files in the workspace.	*Open Resource*
Navigate between open files in the workspace. For more navigation options in the code editor, see Working in the Code Editor [page 1239].	(Back) (Forward) (Last Edit Location)
Close one or more files.	Tab menu: • *Close* • *Close All* • *Close Other* (Closes all files except the active file.)
Import a file from the local file system or from an archive to a selected folder. **i Note** File names can contain only letters, numbers, and the period (.) and underscore (_) characters. Check the *Extract Archive* checkbox to extract files in the archive into the selected folder	*Import* > *From File System*
Export selected files or folders.	*Export*
Archive a folder and save it as a compressed (`.zip`) file under the parent folder. Compressed files larger than 20 MB cannot be archived.	*Archive*
Refresh the view of the selected files and folders in the workspace.	*Refresh*
Link the code editor to the workspace to easily see the location of any active file or show the location of the current active file without linking.	Tab menu: • *Link Workspace to Editor* • *Show in Workspace*

15.2.6.1 Working with Multiple Open Files

You can open multiple files in the code editor, each of which appears in a separate tab.

Tab Area

If more than one file is opened, each one is shown in a different editor tab in the code editor. Each tab shows the file name and its extension as well as an *x*, which allows you to close the file by clicking it. If a file is being edited, its file name is shown in bold and a small asterisk appears next to its name.

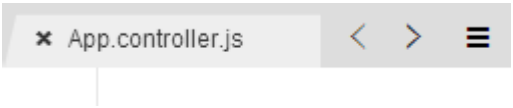
When you hover over the file name, the file path appears in a tooltip.

Maximizing the Editing Area

To enlarge the editing area, double-click any tab in the editor. All open SAP Web IDE panes disappear, and the complete current editor area is fitted to your screen. To restore, double-click the tab.

Tab Overflow

If you open more files in the editor than there is space to display their tabs in the editor area, a small menu appears on the right of the tab area:



Use the arrows to navigate through the tabs in either direction. You can also click the list icon on the right to see the names of all opened files and then choose one from the list to make it the active editor tab.

15.2.7 Resizing Panes

You can show and hide various panes, as well as change their sizes, to fit your way of working.

Context

You can make the following adjustments:

- Show and hide the workspace by choosing *View* > *Workspace*.
- Change the sizes of the workspace, editors area, and panes by dragging the splitters between them.
- Maximize an editor window (that is, hide the workspace and any panes on the right) by double-clicking the title bar of the editor window.

  > **i Note**
  >
  > Double-click again to restore the editor window to its original size. This always redisplays the workspace, and displays any previously opened pane on the right.

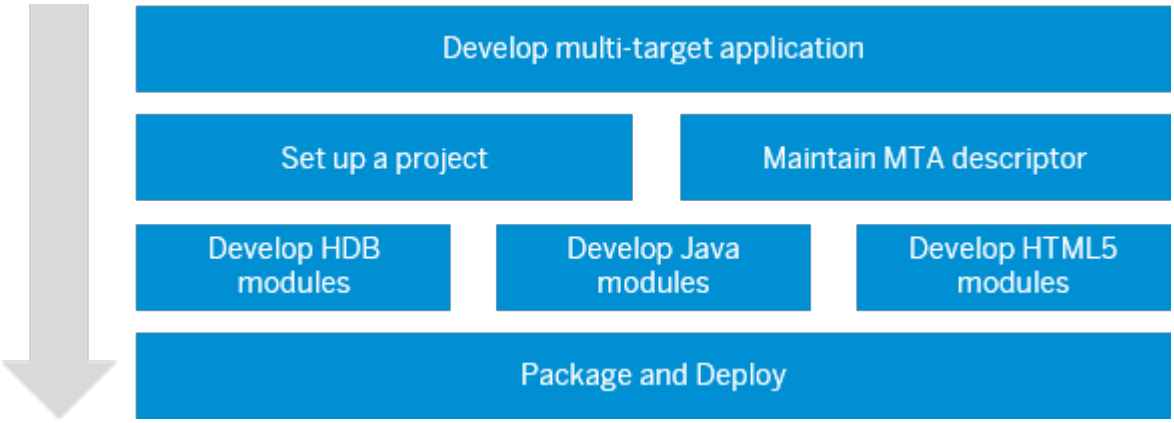
- Restore SAP Web IDE pane defaults by choosing *View* > *Reset to Default*.

Changes to the layout are saved, so the SAP Web IDE will appear the same the next time you open it.

15.3 Developing Multitarget Applications

Multitarget, or multi-tier applications are composed of multiple software modules representing the data, business logic, and UI tiers. These modules are created with different technologies and are deployed to different target platforms, yet share the same development lifecycle.

The multitarget application concept aims at orchestrating the deployment of all these modules so that all runtime dependencies are properly resolved and the application functions as expected. This is achieved by supplying to the deployment tools a set of descriptors that define the interdependencies and deployment scenarios for all modules in the application.



- Developing Multitarget Applications [page 1078]
- Setting Up Application Projects [page 1087]
- Inside an MTA Descriptor [page 1079]
- Developing SAP HANA Database (HDB) Modules [page 1090]
- Developing Java Modules [page 1140]
- Developing HTML5 Modules [page 1150]
- Packaging and Deploying Applications [page 1234]

Terms and Concepts

Term	Description
Multitarget application (MTA)	An application comprised of multiple software modules, which are created with different technologies and deployed to different target platforms, yet share the same lifecycle. In the context of this guide, an "application" is an MTA.
Target platform	A platform to which a module is deployed, such as an SAP HANA database.

Term	Description
MTA descriptor	A YAML file named `mta.yaml` that contains a list of all entities, such as modules, resources, and properties that belong to an application or are used by it at runtime, and the dependencies between them. It is automatically generated when an MTA project is created or modified, or when a module is added or removed. The developer needs to edit the descriptor manually to define resources, properties, and dependencies, as well as fill in missing information.
Module	A self-contained application of a certain type, which is developed, packaged, and deployed to a target platform as part of an MTA.
Module type	A type that defines the structure, development technology, and target platform of a module. SAP Web IDE supports the following module types, representing the three application tiers: • SAP HANA database (HDB) module - represents the data tier • Java and Node.js modules - represent the business logic tier • HTML5 module - represents the UI tier Other module types: • Streaming analytics module - if SAP HANA streaming analytics is installed on your SAP HANA system
Resource	Any resource, such as an external service, property, or environment variable, that is required by a module at runtime but not provided by the module itself.
Property	A property (key-value pair) of an application, module, or resource, that is used during deployment or at runtime.
Parameter	A reserved variable belonging to a module or resource, whose value is used during deployment or at runtime. Parameters can be read-only, write-only, or read-write. The values of writable parameters can be specified in the descriptor.
Dependency	A relationship between a module and another module, resource, or property, such as *provides* and *requires*. • *provides*: indicates the properties or parameters that are provided by a module or resource to other modules. • *requires*: indicates other modules or resources that are required by a module in order to run.
Deployment archive	An archive similar to JAR into which all the application artifacts are packaged for deployment.

15.3.1 Inside an MTA Descriptor

The multi-target application (MTA) descriptor contains the metadata of all entities comprising an application or used by it during deployment or runtime, and the dependencies between them.

The MTA descriptor (the `mta.yaml` file located in the root project folder) is automatically generated when an application project is created from scratch, and it is updated when the project properties change or when a module is added or removed. However, not all the necessary information can be generated automatically. You need to maintain the descriptor manually to define resources, properties, and dependencies, as well as fill in missing information.

Related Information

MTA Editor [page 1080]
MTA Descriptor Model [page 1080]
MTA Descriptor Example [page 1081]
MTA Descriptor Elements [page 1082]
Resource Manager [page 1085]
Configuring Resource Parameters [page 1085]
The Multi-Target Application Model

15.3.1.1 MTA Editor

The MTA descriptor is written in the YAML format, which has strict syntax requirements. You can edit the descriptor in the text-based code editor, but we recommend you use the visual MTA editor because it provides input validation.

To set the MTA editor as the default for the MTA descriptor, go to *Preferences* > *Default Editors* and set the editor for the *MTA Application Descriptor*.

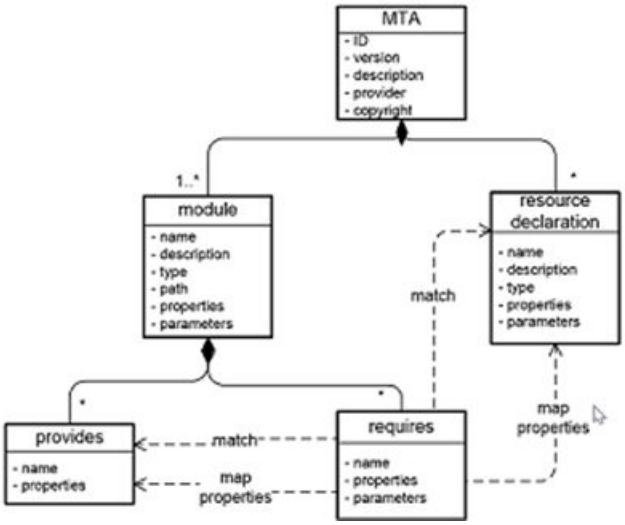
> **i** Note
>
> The MTA visual editor removes comments and formats the file. If you want to add comments, use the code editor. To open the code editor, either make the code editor the default editor or right-click the file and choose *Open With* > *Code Editor*.
>
> If you edit the file with the code editor, it is important to use spaces rather than tabs for indentation.

Related Information

MTA Descriptor Model [page 1080]
MTA Descriptor Example [page 1081]
MTA Descriptor Elements [page 1082]

15.3.1.2 MTA Descriptor Model

The following figure illustrates the MTA descriptor model.



Related Information

MTA Editor [page 1080]

MTA Descriptor Example [page 1081]

MTA Descriptor Elements [page 1082]

15.3.1.3 MTA Descriptor Example

An example of an MTA descriptor is provided here.

```
ID: com.sap.node.hello.world.db
version: 1.0.0
description: A Hello World sample application
provider: SAP Sample generator
copyright: 2016 SAP SE
modules:
   - name: node-hello-world-db
     type: hdb
     path: db
     requires:
        - name: hdi-container
     provides:
        - name: node-hello-world-db
   - name: node-hello-world
     type: html5
     path: web
     requires:
        - name: uaa                    # dependency on the UAA service
        - name: backend_api            # name of the 'provides' element in the
noje.js module
          group: destinations          # name of a target environment variable
          properties:
               name: nodejs            # key values in json format in the
'destinations' variable
               url: ~{url}
```

```
                forwardAuthToken: true
    - name: node-hello-world-backend
      type: nodejs
      path: js
      requires:
         - name: node-hello-world-db
         - name: hdi-container
         - name: uaa
      provides:
         - name: backend_api
           properties:
              url: ${default-url}
# Resources describe required services
resources:
  - name: hdi-container
    type: com.sap.xs.hdi-container
  - name: uaa
    type: com.sap.xs.uaa
```

Related Information

MTA Editor [page 1080]
MTA Descriptor Model [page 1080]
MTA Descriptor Elements [page 1082]

15.3.1.4 MTA Descriptor Elements

The following table provides an overview of the MTA descriptor elements and their possible values.

Sections	Element	Description	Possible Values
General	`ID`	The unique application ID.	Any number of unicode characters in a reverse-URL dot-notation, for example `com.sap.mta.sample`. The ID should be unique in the target runtime environment.
General	`version`	The application version.	Should follow the semantic versioning standard format. For more information, see https://semver.org/ .
General	`description`,	(Optional) A description of the application.	
General	`provider`	(Optional) The application vendor/provider name.	
General	`copyright`	(Optional) The copyright notice of the provider.	
`modules`	`name`	The module name.	Should be unique within the descriptor.

Sections	Element	Description	Possible Values
`modules`	`path`	The relative path to a module from the application root.	For example, `./backend`
`modules`	`type`	The module type, which defines the design-time tools and builders required for the module.	One of the following values: `hdb`, `nodejs`, `html5`.

`modules`	`requires`	A subsection of a module section that contains the names of resources, other modules, and/or properties provided by other modules, which are required by the current module to run.	For example:
			`requires:`
			`  - name: backend`
			`  properties:`
			`  name: external`
			`  url: ~{url}`

`modules`	`provides`	A subsection of a module section that specifies the properties provided by the current module or resource.	For example:
			`provides:`
			`  - name: price_opt`
			`  properties:`
			`  protocol: http`
			`  uri:`
			`host.domain`

`modules`, `resources`	`properties`	(Optional) A flat or hierarchical collection of properties, provided or required by a module or resource.	For example:
			`properties:`
			`  - name: node-hello-world-db`
			`  - name: hdi-container`
			`  - name: uaa`

`modules`, `resources`	`parameters`	(Optional) Reserved variables that can contain read-only, read-write, or write-only values, which can be used by deployment tools, but not at runtime.
		Parameters can be referenced with placeholders enclosed in ${ and }, which are resolved during deployment or runtime.
			For example:
			`parameters:`
			`domain: price.sap.com`

`modules`	`group`	(Optional) A named group of properties from different providers used as a single lookup object.	For example: `group: DESTINATIONS`
`modules`	`description`	(Optional) A description of the module, which does not appear in the application UI.	

> **Note**
>
> For a resource defined in the MTA descriptor of your application, SAP Web IDE may generate a service instance in XS Advanced, adding your user ID as a prefix to the service instance name. You can view and delete these service instances in Resource Manager and in SAP HANA XS Advanced cockpit. For more information, see Resource Manager [page 1085].

For details, see Protecting User Data [page 1274].

Related Information

MTA Editor [page 1080]
MTA Descriptor Model [page 1080]
MTA Descriptor Example [page 1081]

15.3.1.5 MTA Service Provisioning

A service instance required by the MTA application is created automatically for specific resource types.

For example, when you add an HDB module to your MTA project, a resource is added to the `mta.yaml` file. When you build the module, the required hdi-container instance is created automatically in the selected space.

```
ID: mta_db
_schema-version: '2.1'
version: 0.0.1
modules:
 - name: db
   type: hdb
   path: db
   requires:
    - name: hdi_db
resources:
 - name: hdi_db
   properties:
      hdi-container-name: ${service-name}
   type: com.sap.xs.hdi-container
```

Service instances are created automatically for the following resource types (defined in the `mta.yaml` file):

- com.sap.xs.hdi-container
- com.sap.xs.uaa
- org.cloudfoundry.managed-service
- com.sap.xs.dwf–edw-client

> **Note**
>
> Only the initial service instance is created automatically. The service instance is not updated automatically if the resource parameters change. To update the service instance, choose in the Resource Manager. For more information, see Resource Manager [page 1085].

15.3.1.6 Resource Manager

The *Resource Manager* tool lets you view information about all of the resources (service instances) required by the modules based on the `mta.yaml` file in the multitarget application project.

To open the tool, from the right sidebar, choose .

The table of service instances for the project is divided into design time and runtime.

You can also delete, update, and refresh service instances:

- Choose (Refresh) to rebuild the table based on the resources listed in the `mta.yaml` file and the service instances created in the runtime environment.
- Choose (Delete Service Instance) to delete a service instance from the runtime environment.
- Choose (Update service instance configuration according to the mta.yaml) to update a service instance from the runtime environment, based on the design time configuration (`mta.yaml` file).

15.3.1.7 Configuring Resource Parameters

How to configure resource parameters within and outside an MTA descriptor.

For the application deployment you sometimes need to configure additional parameters for resources. You can configure these parameters directly in the MTA descriptor, for example:

```
resources:
  - name: hdi-container
    type: com.sap.xs.hdi-container
    parameters:
      config:
        schema: ${default-container-name}
```

These configurations can be lengthy. To keep the MTA descriptor short and easy to maintain, you can define them in separate `.json` files within the project, and specify the file's path in the resource's `path` of the corresponding resource. For example, it is a good practice to define security configurations in a separate file:

```
resources:
  - name: node-uaa
```

```
    type: com.sap.xs.uaa
    parameters:
      path: ./xs-security.json
```

Here is how you can specify an external configuration file in the MTA descriptor.

```
resources:
- name: some_resource
  parameters:
      path: myconfig/configuration.json  #path to the json file within the
project
```

Let's say, the specified file contains a configuration `{ “config1”: “somevalue”, “config2”: [ 123, 456 ] }`.

During the application build, the `.json` file will not be included in the resulting MTA archive. Rather, the content of this file will be integrated into the deployment descriptor (`mtad.yaml`) in the following way:

```
resources:
- name: some_resource
   parameters:
      config:
         config1: somevalue
         config2:
            - 123
            - 456
```

15.3.2 Application Development Workflow

Development workflows and tasks that you perform to create and maintain multi-target applications in SAP Web IDE for SAP HANA.

Workflow/Task	Instructions
1. Start SAP Web IDE for SAP HANA and get familiar with the working environment.	Getting Started [page 1062]
2. Set up a project for developing a multi-target application.	Setting Up Application Projects [page 1087]
3. Implement the database tier of your application.	Developing SAP HANA Database (HDB) Modules [page 1090]
4. Implement the business logic tier of your application.	Developing Node.js Modules [page 1129] Developing Java Modules [page 1140]
5. Implement the user interface of your application.	Developing HTML5 Modules [page 1150]
6. Prepare your application for deployment.	Packaging and Deploying Applications [page 1234]

> **i Note**
>
> If you have SAP HANA streaming analytics installed on your SAP HANA system, you can create streaming analytics modules. For more information, see *Working with Projects in SAP Web IDE for SAP HANA* in the SAP HANA Streaming Analytics: Developer Guide.

15.3.3 Setting Up Application Projects

You can set up a multi-target application (MTA) project in various ways.

To set up an MTA project, you can either:

- Create a project from scratch
- Import a project from an archive
- Clone a project from a Git repository

You can change the space selection in SAP HANA XS Advanced for an existing, imported, or cloned project. See Select a Space for a Project [page 1090].

Related Information

Create a Project from Scratch [page 1087]
Import a Project from an Archive [page 1088]
Clone a Project from Git [page 1089]

15.3.3.1 Create a Project from Scratch

You create an MTA project using a dedicated template provided by SAP Web IDE.

Procedure

1. From the *Workspace* menu, choose *New* > *Project from Template*.
2. Choose the *Multi-Target Application Project* template and click *Next*.

   > **i** Note
   >
   > If you have enabled the Data Warehousing Foundation feature, you can choose the *Data Warehouse Project* template, which creates a customized MTA project.
   >
   > For more information, see SAP HANA Data Warehousing Foundation - Data Warehouse Scheduler Administration Guide (XSA).

3. Enter a project name and click *Next*.
4. If needed, modify the *Application ID* and *Version* properties of the project. Both properties appear in the MTA descriptor (`mta.yaml` file).
5. (Optional) Enter a description of the project.
6. Select an existing space in SAP HANA XS Advanced for your project from the *Space* dropdown list.

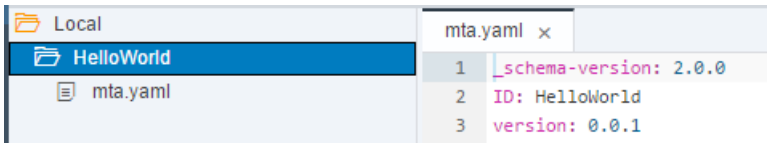
> **Note**
>
> To support the isolation of the development environment, SAP Web IDE enables developers to use dedicated spaces in XS Advanced for building and running their projects. By selecting a space in SAP HANA XS Advanced for your project, you ensure that:
>
> - Your application can share the resources with other applications or services running in this space.
> - Applications running is other spaces cannot access these resources.

The dropdown list contains the available spaces, created by your SAP HANA XS Advanced administrator, in which you have the *SpaceDeveloper* role. If the required space does not appear in the list, contact your administrator. If you do not select any space, an error will occur when you attempt to build or run your project.

You can change the space selection later, as described in the *Changing Space Selection for a Project* section.

7. Once you have entered all the mandatory fields, click *Next* or *Finish*.

   A new project with the specified name is created in your workspace. The project contains the initial MTA descriptor. For example:



Related Information

Setting Up Application Projects [page 1087]
Import a Project from an Archive [page 1088]
Clone a Project from Git [page 1089]
Select a Space for a Project [page 1090]

15.3.3.2 Import a Project from an Archive

You can import to the SAP Web IDE workspace a multi-target application project that was previously exported to an archive.

Prerequisites

- You must have a compressed `(.zip)` file of the project that you want to import.
- The size of the `(.zip)` file must not exceed 20MB.

Procedure

1. In the workspace, select the top-level `Local` folder.
2. Choose *File* > *Import* > *File or Project*.
3. Click *Browse* to locate and select your archived project file, and choose *Open*. The file name appears in the *File* field.

   The destination folder name is displayed in the *Import to* field. By default, this name is the same as for the archive file. You can change the folder name, but not its location.
4. Choose *OK*. The project is created in the specified folder.

   If an MTA descriptor (`mta.yaml` file), located in the project root folder, contains the definitions of modules, the corresponding subfolders are automatically converted into modules.

   Otherwise, you need to check whether the module subfolders are successfully converted into modules. You can do this by choosing *Project Settings* from each module subfolder context menu, and verifying that the project type is one of the supported module types. If it is not, from the subfolder context menu, choose *Convert To*, and the type of the target module.

15.3.3.3 Clone a Project from Git

You can clone an existing project from a Git repository.

Procedure

Follow the instructions in Clone Repositories [page 1258].

Related Information

Inside an MTA Descriptor [page 1079]
Customizing Your Project [page 1235]
Setting Up Application Projects [page 1087]
Create a Project from Scratch [page 1087]
Import a Project from an Archive [page 1088]

15.3.3.4 Select a Space for a Project

You can change the space selection in SAP HANA XS Advanced for an existing, imported, or cloned project.

Procedure

1. From the project context menu, choose *Project Settings* > *Space*.
2. From the *Space* dropdown list, select an appropriate space in SAP HANA XS Advanced for building and running your project.

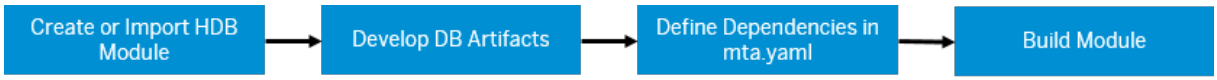
   > **i Note**
   >
   > The builder must be installed in a space that is within the same organization as the SAP space.

3. In the *Builder* section, the button label indicates the status of builder in the selected space:
   - *Install Builder* means that there is no builder in the selected space. Click the button to install the builder.
   - *Reinstall Builder* means that there is already a builder in the selected space. Click the button if you experience any build problem, or if you want to update the builder.

15.3.4 Developing SAP HANA Database (HDB) Modules

An SAP HANA database (HDB) module is a collection of related design-time database artifacts, such as data models, views, or procedures.

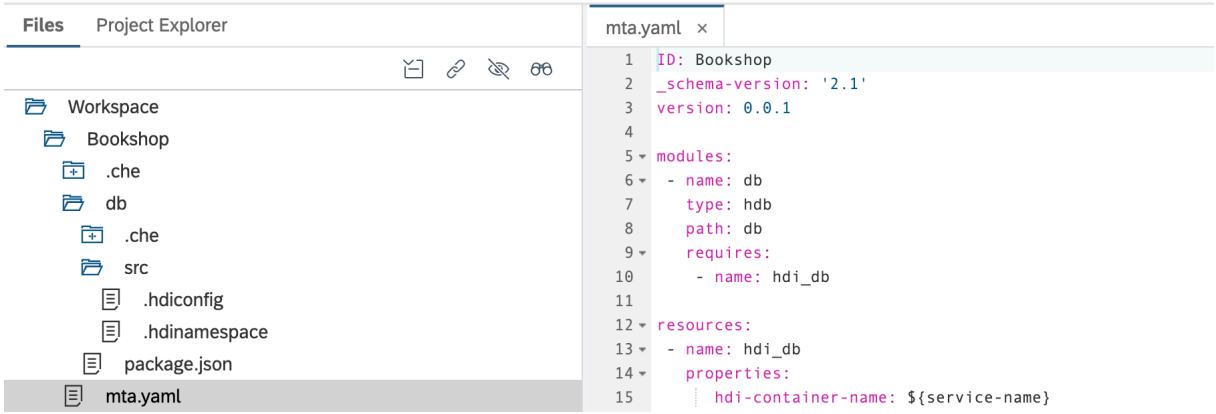
To develop and deploy these artifacts into the SAP HANA database, perform the following steps:



- Create an HDB Module [page 1091]
- Developing Database Artifacts [page 1093]
- Inside an MTA Descriptor [page 1079]
- Build an HDB Module [page 1126]

Module Folder Structure

The following figure depicts a sample HDB module folder structure alongside the corresponding entry in the `mta.yaml`.



Folder	Description
<module name>	Should not contain any files.
src (default)	The default location in which you store your design-time database artifacts. If needed, you can create additional subfolders for the same purpose.

Related Information

Defining the Data Model in XS Advanced [page 233]

15.3.4.1 Create an HDB Module

You can create new SAP HANA database (HDB) modules.

Procedure

From the project context menu, choose *New* > *SAP HANA Database Module*, and follow the wizard steps to enter the module properties.
A new HDB module with the specified name is created in your project, and a corresponding section is added to the MTA descriptor (`mta.yaml`). For example:

Related Information

Developing SAP HANA Database (HDB) Modules [page 1090]
Import an HDB Module [page 1092]

15.3.4.2 Import an HDB Module

You can import archived SAP HANA database (HDB) modules.

Prerequisites

The `.zip` module archive that you want to import, which was exported from another MTA project, is available in the file system.

Procedure

1. From the root folder of the project, choose *File* > *Import* > *File or Project*.
2. Click *Browse* to locate and select your archive, and choose *Open*. The file name appears in the *File* field.

   The destination folder is displayed in the *Import to* field. To change this folder, choose *Select Folder*, and browse to the required folder, or create a new folder.

   The specified folder, containing the artifacts extracted from the archive, is created in the project.
3. To make the imported folder a proper module in your project, you need to convert it into a module of the matching type. From the folder context menu, choose *Convert To*, and then the type of the target module.

   > **Note**
   >
   > The conversion process does not check whether the imported folder structure matches the selected module type. The process does not generate the module artifacts according to the selected type.

   The imported module becomes a part of your MTA project, and the module entry is added to the MTA descriptor.

15.3.4.3 Developing Database Artifacts

Create the database artifacts required for your module.

Procedure

From the context menu of the module's `src` subfolder, choose *New*, and then choose one of the available artifacts.

> **i Note**
>
> If you don't see the required artifact type in the menu, this means that the optional feature (extension), which supports it, is disabled. To enable the relevant feature (extension), follow the instructions in Enable SAP Web IDE Extensions [page 1066].

Artifact	Instructions
Database procedure (.hdbprocedure)	Choose *Procedure*, and enter the file name. The new artifact is added to the module, and opens in a dedicated code editor. For information about developing HDB procedures, see: Defining the Data Model in XS Advanced [page 233] .
Calculation view (.hdbcalculationview) **A design-time definition of a calculation view, which can be referenced in an OData service definition.**	Choose *Calculation View*. Enter a name, label, and select the type and category. The new artifact is added to the module, and opens in the chosen editor. For information about developing calculation views in SAP Web IDE for SAP HANA (on-premise), see Creating a Graphical Calculation View..
Flowgraph (.hdbflowgraph) **A batch or real-time data flow to transform your data.**	Use the Flowgraph Editor to model the transformation of your data. Choose *Flowgraph*, and then enter a name. In the *Flowgraph* editor that opens, you can place nodes onto the canvas, and configure the options to output the data you need. For example, you can aggregate, filter and join tables, or create a stored procedure. For more information about creating flowgraphs, see Transforming Data Using Flowgraphs [page 1123].
CDS (Core Data Services) Document (.hdbcds)	**i Note** This option is not available in SAP HANA Cloud. Choose *HDB CDS Artifact*, and enter a name. The new artifact is added to the module, and opens in the default editor defined in *Tools* > *Preferences* > *Default Editors* > *HDB CDS Source*. For information about developing CDS documents, see: • In the text editor: Create a CDS Document (XS Advanced) [page 313] • In the graphical editor: Getting Started with the CDS Graphical Editor [page 1095] and Creating Synonyms [page 1120].

Artifact	Instructions
	Native datastore object (NDSO) in SAP HANA Datawarehousing Foundation (available with DWF 2.0) is represented as an annotated CDS context. For more information about the NDSO, see Native DataStore Object [page 1123].
Analytic privilege (.hdbanalyticprivilege)	Choose *Analytic Privilege*, and enter a name and label. The new artifact is added to the module, and opens in the dedicated editor. For information about developing analytic privileges, see Setting Up the Analytic Model [page 511].
Text analysis dictionary (.hdbtextdict)	You can specify your own entity types and entity names to be used with text analysis by creating custom text analysis dictionaries. Choose *Text Analysis Dictionary* and enter a name with or without the file extension. A text editor will open for the new dictionary file, with an XML template automatically populated. For more information about developing text analysis dictionaries, see Developing Text Analysis Artifacts [page 1124].
Text analysis rule set (.hdbtextrule)	You can specify your own entity types to be used with text analysis by creating custom text analysis extraction rules. Choose *Text Analysis Rule Set* and enter a name with or without the file extension. A text editor will open for the new rule set file. For more information about developing text analysis rule sets, see Developing Text Analysis Artifacts [page 1124].
Text analysis configuration (.hdbtextconfig)	You can specify your own configuration settings to be used with text analysis. Choose *Text Analysis Configuration* and enter a name with or without the file extension. Choose one of the standard SAP HANA configurations to use as a starting point. A text editor will open for the new configuration file, with an XML template automatically populated with the specified standard configuration settings. For more information about developing text analysis configurations, see Developing Text Analysis Artifacts [page 1124].
Role (.hdbrole)	You can create a new role and grant it additional roles or object, schema, analytic and system privileges. Choose *Role* and enter a name for the new role. Optionally, you can also choose an `.hdbroleconfig` file. To edit an existing role, right-click the role in the catalog and click *Open Role Editor*. For more information about roles, see Roles (.hdbrole).

> **i Note**
>
> Currently, SAP Web IDE provides dedicated editors only for the artifacts listed above. You can develop other artifacts, supported by SAP HANA XS Advanced (XSA), using a text editor. Create a file in the module's `src` subfolder with an appropriate extension, and open it in a text editor.

You can now easily access the database artifacts of an HDB module in the database explorer. To do this, build your HDB module, and choose *Open HDI Container* from the module's context menu. The HDI (HANA Deployment Infrastructure) container with the deployed database artifacts of the module opens in the database explorer.

15.3.4.3.1 Getting Started with the CDS Graphical Editor

SAP Web IDE offers data modelers the graphical modeling capabilities that they require to model artifacts. These artifacts define the data persistence objects using the Core Data Services (CDS).

The CDS graphical editor in the SAP Web IDE helps model data persistence objects in CDS artifacts using the DDL-compliant CDS syntax. The tool recognizes the `.hdbcds` file extension required for CDS object definitions and calls the appropriate repository plug-in to parse the CDS artifact. If you right-click a file with the `.hdbcds` extension in the *Workspace* view, SAP Web IDE provides the following choice of editors in the context-sensitive menu.

- Text Editor
  View and edit DDL source code in a CDS document as text with the syntax elements highlighted for easier visual scanning.
- Graphical Editor
  View graphical representation of CDS artifacts, and also model CDS artifacts with graphical modeling tools.

This section describes how data modelers can use the graphical editor to model CDS artifacts. In the current version of the XS advanced model, you can use a graphical editor only to model the following important artifacts or elements in SAP Web IDE:

- Entities (tables)
- Contexts (Containers)
- Associations
- User-defined data types
- Views (with unions, joins, calculated columns, and more.)

Related Information

Create an Artifact with the CDS Graphical Editor [page 1096]
Create an Association with the CDS Graphical Editor [page 1102]
Create a User-Defined Structure Type with the CDS Graphical Editor [page 1106]
Create an Entity with the CDS Graphical Editor [page 1098]
Create a User-Defined Scalar Type with the CDS Graphical Editor [page 1108]
Create a View with the CDS Graphical Editor [page 1109]

15.3.4.3.1.1 Create an Artifact with the CDS Graphical Editor

A CDS artifact is a design-time source file that contains definitions of objects, which you want to create in the SAP HANA catalog. You can use graphical modeling tools to model a CDS artifact.

Context

CDS artifacts are design-time source files that contain DDL code. The code describes a persistence model according to rules defined in Core Data Services. CDS artifacts have the file suffix `.hdbcds`. Building an SAP HANA Database Module that includes this CDS artifact, creates the corresponding catalog objects in the specified schema. To create a CDS artifact, do the following:

Procedure

1. Start SAP Web IDE in a Web browser.
2. Display the application project to which you want to add a CDS artifact view.

   SAP Web IDE creates an application within a context of a project. If you do not already have a project, there are a number of ways to create one, for example: by importing it, cloning it, or creating a new one from scratch.

   a. In the SAP Web IDE, choose *File* > *New* > *Project from Template*.
   b. Choose the project template type.

      Currently, there is only one type of project template available, namely: *Multi-Target Application Project*. Select *Multi-Target Application Project* and choose *Next*.
   c. Type a name for the new MTA project (for example, **myApp**) and choose *Next* to confirm.
   d. Specify details of the new MTA project and choose *Next* to confirm.
   e. Create the new MTA project; choose *Finish*.
3. Select the module in which you want to create the CDS artifact.
4. In the *Workspace* view, browse to the *src* folder.
5. Right-click the *src* folder and choose *New* > *HDB CDS Artifact*.
6. In the *Name* field, enter the name of the CDS artifact.
7. Choose *Create*.

   The tool opens a graphical editor that you can use to define the CDS artifact. You use the same editor to model CDS entities, contexts, associations, views, user-defined data types, and more within a CDS artifact.

   > **i Note**
   >
   > You can define your preferred CDS editor (graphical or text) in *Tools* > *Preferences* > *Default Editor* > *HANA CDS Source*.
8. (Optional) Create subcontexts.

You can create nested subcontexts within a CDS artifact. Each of these subcontexts can contain any or multiple CDS artifacts such as entities, views, structure types, and more.

a. In the editor toolbar, choose (Create Context).
b. Drop the context node on the editor.
c. Provide a name to the context.
d. Double-click the context node.

   You can define new entities, associations, or another context within the subcontext. Use the breadcrumb navigation in the editor toolbar to switch between contexts.

9. (Optional) Publish CDS artifact as an OData service.

   You can publish CDS artifacts at the context level as OData v4 services. You can also publish subcontexts as OData services.

   In the editor toolbar, choose (Publish as OData).

Related Information

Create an Entity with the CDS Graphical Editor [page 1098]
Create an Association with the CDS Graphical Editor [page 1102]
Create a User-Defined Structure Type with the CDS Graphical Editor [page 1106]
Create a User-Defined Scalar Type with the CDS Graphical Editor [page 1108]
Create a View with the CDS Graphical Editor [page 1109]
Classification of Supported Data Types [page 1097]

15.3.4.3.1.1.1 Classification of Supported Data Types

You can use the CDS graphical tools to model a single CDS artifact that comprises of multiple entities, structure types, and more. When you are defining the elements in the entities or in the structure types, define data types of these elements.

The tool supports multiple data types for defining the data type of an element. For convenience, and based on its characteristics, the data types are classified as follows:

Type	Description
Primitive	Primitive types are standard Data Definition Language (DDL) data types such as String, Binary, or Integer.
Native	Native types are the native SAP HANA data types.
Structure Type	Structure types are user defined data types that comprises of a list of elements, each of which has its own data type.

Type	Description
Scalar Type	Scalar types are user define scalar data types. Unlike user-defined structure types, scalar types do not comprise of any elements in its definition.
Structure Element	Structure elements help reuse the data type of an element defined in a selected structure in another data type definition.
Entity Element	Entity elements help reuse the data types of an element defined in a selected entity in another data type definition.

15.3.4.3.1.2 Create an Entity with the CDS Graphical Editor

Use the CDS graphical editor to model entities. In the SAP HANA database, a CDS entity is a table with a set of data elements that are organized using columns and rows.

Context

You model entities within a CDS artifact using the graphical modeling capabilities of the CDS graphical editor.

Procedure

1. Start SAP Web IDE in a Web browser.
2. In the *Workspace* view, select the required CDS artifact within which you want to model the entity.
3. In the context menu, choose *Open With* > *Graphical Editor*.
   The tool opens the CDS artifact in a graphical editor where you can model the entity.
4. Create an entity.
   a. In the editor toolbar, choose (Create Entity).
   b. Drop the entity node on the editor.
   c. Provide a name for the entity.
5. Define the entity.

   You can define one or more elements for the entity.
   a. Double-click the CDS entity node.
   b. In the editor toolbar, choose + (Add) to create a new element in the entity.
   c. In the *Name* text field, you can modify the name of the element.
   d. For each element, in the *Type* dropdown list, select the required value.
   e. For each element, in the *Data Type* dropdown list, select the required value.

If you have selected the type of data type as *Structure Element* or *Entity Element*, then in the *Data Type* dropdown list, select the required structure or entity from within the same CDS artifact. Use the *Type of Element* value help list to select the required element.

f. Define the length and scale based on the selected data type.

g. Define whether the specified column is a primary key or part of the primary key for the specified entity. Select the *Key* checkbox accordingly.

> **i Note**
>
> Structured columns can be part of the key, too. In this case, all table fields resulting from the flattening of this structured field are part of the primary key.

h. Define if an entity column can (null) or cannot (not null) have the value NULL. If neither null nor not null is specified for the column, the default value null applies (except for the key column). Select the *Not Null* checkbox accordingly.

i. Define the default value for an entity column in the event that no value is provided during an INSERT operation. Enter a value in the *Default* text field.

6. (Optional) Use an expression to generate the element value at runtime.

   The SAP HANA SQL clause `generated always as <expression>` is available for use in CDS entity definitions. This clause specifies the expression that the tool must use to generate the element value at runtime.

   a. Select the required element.

   b. In the *Expression* field, choose (Element Expression Editor).

   c. In the *Expression* editor, enter the required expression.

      Use the elements and operators from the expression editor to build your expression.

   d. Choose *OK*.

   e. If you want to define the element with a generated expression, select the *Generated* checkbox.

      If you do not select the *Generated* checkbox, the tool considers the expression for the element as calculated expression. Generated elements are physically present in the database table; values are computed on `INSERT` and need not be computed on `SELECT`. Calculated elements are not actually stored in the database table; they are computed when the element is "selected". Since the value of the generated column is computed on `INSERT`, the expression used to generate the value must not contain any non deterministic functions, for example: `current_timestamp, current_user, current_schema` and more.

7. Create child element, if required.

   For any selected element in the entity, you can create one or more child elements.

   a. Select the required element.

   b. In the editor toolbar, choose (Create child).

   c. Provide a name for the child element.

   d. Define the data type and default values for the child element.

8. Import elements, if required.

   When defining the elements in an entity, you can also do so by importing elements from other catalog tables. These catalog tables can be available in the same HDI container in which you are creating the entity or a synonym that points to catalog table in another HDI container. To reuse the definition of imported elements,

a. In the editor toolbar, choose (Import Element(s)).
b. In the *Import Elements* wizard, search and select the required catalog table.

   You can use the elements and its definition from the selected catalog table to define the elements of the entity.
c. Choose *Next*.
d. Select the elements you want to import.
e. Choose *Finish*.

You can modify the definition of imported elements such as, its data type, length, scale, and more.

> **i Note**
>
> The data type of elements that you import from catalog tables, and the data type of the same elements in its design-time entity from which you generated the catalog table, need not be the same. It may vary or be the same depending on the data type defined for the elements.

9. Define additional properties.
   a. In the *Storage Type* dropdown list, select the required storage type value (row or column).

      If no storage type is specified, a "`column`" store table is generated by default.
   b. In the *Unload Priority* text field, specify the required unload priority value for the generated table.

      Unload priority specifies the priority with which a table is unloaded from memory. The priority can be set between 0 (zero) and 9 (nine), where 0 means "cannot be unloaded" and 9 means "earliest unload".
   c. Select the *Auto Merge* checkbox if you want to trigger an automatic delta merge.

      > **i Note**
      >
      > Not selecting the *Auto Merge* checkbox disables the automatic delta merge operation.
   d. If you want to specify the grouping information for the generated tables, select the *Enable Table Grouping* checkbox and specify the group name, type, and sub type.
10. (Optional) Specify an identity column.

    The SAP HANA SQL clause `generated as identity` is available for use in CDS entity definitions; if you are using an expression to generate the element value at runtime, this clause enables you to specify an identity column. An element that is defined with `generated as identity` corresponds to a field in the database table that is present in the persistence and has a value that is computed as specified in the sequence options defined in the `identity` expression, for example, `( start with 10 increment by 2 )`.
    a. In the menu bar, select the *Properties* tab.
    b. In the *Element* dropdown list, select an element from the entity that you want to use as identity column.
    c. In the *Type* dropdown list, select a value.

       You can use either *always* or *by default*. If you select *always*, then values are always generated; if you select *by default*, then values are generated by default.
    d. Provide the *Start With* and *Increment by* values.

       For example, `( start with 10 increment by 2 )`.
    e. In the *Minimum Value* and *Maximum Value* dropdown list, select the required value and provide the minimum and maximum values.

f. In the *Cache* text field, provide a value.

   The cache value is typically the difference of maximum value and minimum value.

g. In the *Cycle* dropdown list, select a value.

   This value helps the tool determine whether it is a cycling sequence or non cycling sequence.

11. Choose *Save* to save all your changes.
12. Build an SAP HANA Database Module.

    a. From the module context menu, choose *Build*.

Next Steps

Some business cases may require performing additional tasks on an entity such as defining associations, creating partitions, and more. After creating and defining an entity, you can perform the following additional tasks within the entity.

Requirement	Task to Perform
Define relationship between entities, or between structure types and entities.	Create an Association
Partition the entity using elements in the entity.	Create a Partition
Specify indexes on entities	Create Indexes
Create entities to efficiently store series data	Create Entities to Store Series Data

Related Information

Create an Association with the CDS Graphical Editor [page 1102]
Create Indexes with the CDS Graphical Editor [page 1103]
Create a Partition with the CDS Graphical Editor [page 1104]
Create Entities to Store Series Data [page 1105]

15.3.4.3.1.2.1 Create an Association with the CDS Graphical Editor

Associations define relationships between entities, or relationship between structure types and entities. You create associations in either a CDS entity definition or structure type definition, which are design-time files in SAP HANA.

Context

The CDS graphical editor tool in SAP Web IDE provides you graphical modeling capabilities to model associations between entities, or between structure types and entities. If you are creating associations between entities, then the associations are defined as part of the entity definition, which are design-time files in the repository. Similarly, if you are creating associations between structure types and entities, then the associations are maintained in the structure type definition, which are design-time files in the repository.

Procedure

1. In the *Workspace* view, select the required CDS artifact.
2. In the context menu, choose *Open With* > *Graphical Editor*.
3. Create associations between entities.

   If you are creating associations between entities, you need at least two entities in the CDS artifact.
   a. Select the required entity.
   b. Choose (Association).
   c. Drag the cursor to another entity in the CDS artifact, with which you want to create associations.
4. Create associations between structure types and entities.

   If you are creating associations between a structure type and an entity, you need at least one structure type and one entity in the CDS artifact. In associations between structure type and entity, the target is an entity. This means that, the association definition is maintained in the structure type definition.
   a. Select the required structure type.
   b. Choose .
   c. Drag the cursor to an entity in the CDS artifact, with which you want to create associations.
5. Provide association details.

   In the *Association Details* dialog box, provide further details for the association.
   a. In the *Name* text field, provide a name for the association.
   b. Select the association type.

      Select *Managed* or *Unmanaged* association type, depending on whether you want to create managed or unmanaged associations.

> **Note**
>
> For associations between structure types and entities, you can create only managed associations.

c. If you are creating managed associations, select required elements from the structure type or entity as association keys.

   For managed associations, the relation between source and target entity is defined by specifying a set of elements of the target entity that are used as a foreign key. If no foreign keys are explicitly specified, the elements of the target entity's designated primary key are used. In the *Alias* text field, provide an alias name, if required.

d. If you are creating unmanaged associations, in the expression editor, provide the required condition.

   Unmanaged associations are based on existing elements of the source and target entity; no fields are generated. In the `ON` condition, only elements of the source or the target entity can be used; it is not possible to use other associations. The `ON` condition may contain any kind of expression. Use the elements and operators from the editor, when defining the `ON` condition expression.

6. Define cardinality, if required.

   When using an association to define a relationship between entities in a CDS artifact, use the cardinality to specify the type of relation, for example, one-to-one (to-one) or one-to-many (to-n); the relationship is with respect to both the source and the target of the association.

   a. In the *Source Cardinality* and *Target Cardinality* dropdown lists, select the required cardinality values.

7. Choose *OK*.
8. Choose *Save*.

> **Note**
>
> You can also create associations using a form-based editor. Double-click the required entity or structure type node in the CDS artifact, and in the Associations tab, create, and define the association.

15.3.4.3.1.2.2 Create Indexes with the CDS Graphical Editor

You can create indexes for an entity and specify an index type. The tool supports three types of indexes, plain indexes, full text indexes, and fuzzy search indexes.

Procedure

1. In the menu bar, choose the *Indexes* tab.
2. Choose + (Add) to create a new index.
3. In the *General* section, select the required index type.

   After selecting the index type, define which of the elements in the entity should be indexed. For the plain index type, you can also select the index order and use the *Unique* checkbox to define whether the index is unique (no two rows of data in the indexed entity can have identical key values). For *Full Text Index*, select the element and define the full text parameters such as the language column, mime type column, and more.

> **Note**
>
> You cannot specify both a full-text index and a fuzzy search index for the same element.

4. Choose *Save*.

15.3.4.3.1.2.3 Create a Partition with the CDS Graphical Editor

Use elements from the entity to partition the entity. You can use hash, range, or round-robin partition types to partition the entity.

Procedure

1. In the menu bar, choose the *Partitions* tab.
2. Select the *Add Primary Partition* checkbox.
3. In the *Partition Type* dropdown list, select a partition type.

> **Note**
>
> For *Hash* or *Roundrobin* partition types, you can also define the secondary partition type. In the *Secondary Partition* section, select the required secondary partition type.

4. Choose + (Add) to add an element.
5. In the *Element* dropdown list, select an element, which you want to use to partition the data.
6. In the *Functions* dropdown list, select a value. This is applicable if any of the selected element used for partitioning the entity represents time values.
7. In the *Partitions* text field, enter the number of partitions required.

> **Note**
>
> If you want to partition the entity based on number of servers, select the *Number of Servers* checkbox

8. Choose *Save*.

15.3.4.3.1.2.4 Create Entities to Store Series Data

Use the CDS graphical editor in SAP Web IDE to create entities that can efficiently store series data.

Procedure

1. In the menu bar, choose the *Series* tab.
2. Select the *Enable Series* checkbox.
3. In the *Series Key Element(s)* dropdown list, select one or more elements.
4. If you want to create an entity to store time series data, in the *Period for Series* dropdown list, select the required period column.

   Each row of a series table has a period of validity. The period represents the period in time that the row applies to. When the series table represents instants, then there is a single period column. When the table represents intervals, there can be one or two columns.

   > **i Note**
   >
   > The period for series must be unique and should not be affected by any shift in timestamps. If you want to create an entity to store non time series data, select the *Null* checkbox.

5. Select the required *Equidistant Type*.

Type	Description
Not Equidistant	If you want to create an entity to store arbitrary time values without any regular pattern.
Equidistant	If you are creating an entity to store time values with regular pattern. For equidistant series, provide the *Increment By* value that defines the distance between adjacent elements in an equidistant series. The equidistant series tables offer improved compression compared to non equidistant tables.
Equidistant Precisewise	If you want to create an entity to store series data that retains a degree of regularity without being equidistant across the entire dataset. Equidistant piecewise data consists of regions where equal increments exist between successive points, but these increments are not always consistent for all regions in the table. Different series or different parts of a series may have different intervals. For equidistant precisewise series, you use only one period column, but you can specify one or more *Alternate Period for Series*. These columns can be used to record the time for each row, or represent the end of an interval associated with each row. Good compression can be achieved for these columns when successive values within the series differ by a constant amount.

6. (Optional) Provide minimum and maximum values.

   You can define equidistant and non equidistant series tables with a minimum and/or maximum value. These values are used to verify that loaded data corresponds to the range definition. The values in *Period for Series* column must satisfy the minimum values and the maximum value constraint.

   a. In the *Min Value* dropdown list, select the *Min Value* menu option and provide the required minimum value.
   b. In the *Max Value* dropdown list, select the *Max Value* menu option and provide the required maximum value.

> **i Note**
>
> If the *Period for Series* column does not have a lower limit and upper limit, you can use the *No Min Value* and *No Max Value* respectively.

15.3.4.3.1.3 Create a User-Defined Structure Type with the CDS Graphical Editor

A structured type is a user-defined data type comprising a list of attributes, each of which has its own data type. You create a user-defined structured type as a design-time file in the SAP HANA repository.

Context

Use the CDS graphical editor to create a user-defined structured type as a design-time file in the repository. The attributes of the structured type can be defined manually in the structured type itself and reused by another structured type (or by scalar type, or by an entity). You create a user-defined structure types within a CDS artifact.

Procedure

1. Start SAP Web IDE in a Web browser.
2. In the *Workspace* view, select the required CDS artifact within which you want to model the entity.
3. In the context menu, choose *Open With* > *Graphical Editor*.

   The tool opens the CDS artifact in a graphical editor where you can create the structure type.
4. Create a structured type.

   a. In the editor toolbar, select (Create Structure).
   b. Drop the structure node on the editor.
   c. Provide a name to the structure.
5. Define the elements.

   a. Double-click the structure type node.

b. In the editor toolbar, choose + (Add) to create a new element in the structure.
c. In the *Name* field, you can modify the name of the element.
d. For each element, in the *Type* dropdown list, select the required value.
e. For each element, in the *Data Type* dropdown list, select the required value.

   If you have selected the type of data type as *Structure Element* or *Entity Element*, then in the *Data Type* dropdown list, select the required structure or entity in the CDS artifact. Use the *Type of Element* value help list to select the required element.
f. Define the length and scale based on the selected data type.

6. Create child element, if required.

   For any element in the structure, you can also create one or more child elements.

   a. Select the required element.
   b. In the editor toolbar, choose (Create child).
   c. Define the data type for the child element.

   > **i Note**
   >
   > When you create a child element, the tool automatically changes the type of data type of its parent element to *Inline*.

7. Import elements, if required.

   When defining the elements in an entity, you can also do so by importing elements from other catalog tables. These catalog tables can be available in the same HDI container in which you are creating the entity or a synonym that points to catalog table in another HDI container. To reuse the definition of imported elements,

   a. In the editor toolbar, choose (Import Element(s)).
   b. In the *Import Elements* wizard, search and select the required catalog table.

      You can use the elements and its definition from the selected catalog table to define the elements of the structure type.
   c. Choose *Next*.
   d. Select the elements you want to import.
   e. Choose *Finish*.

   You can modify the definition of imported elements such as its data type, length, scale, and more.

8. In the editor toolbar, choose (Navigate Back).
9. Choose *Save*.

15.3.4.3.1.4 Create a User-Defined Scalar Type with the CDS Graphical Editor

Scalar types are user-defined scalar data types that does not comprise of any elements in its definition. You create a user-defined structured type as a design-time file in the SAP HANA repository.

Context

Use the CDS graphical editor to create a user-defined structured type as a design-time file in the repository. You create a user-defined scalar data types within a CDS artifact. After creating a scalar type, you can reuse it when defining data types of elements in other structure types, scalar types, or entities.

Procedure

1. Start SAP Web IDE in a Web browser.
2. In the *Workspace* view, select the required CDS artifact within which you want to model the entity.
3. In the context menu, choose *Open With* > *Graphical Editor*.

   The tool opens the CDS artifact in a graphical editor where you can create the scalar type.
4. Create a scalar type.

   a. In the editor toolbar, select (Create Scalar type).
   b. Drop the scalar type node on the editor.
   c. Provide a name to the scalar type.
5. Define the scalar type.

   a. Double-click the scalar type node.
   b. For each element, in the *Type* dropdown list, select the required value.

      The data types are classified as primitive data types, native SAP HANA data types, user-defined data types (structure types and scalar types), entity element and structure element. Based on your requirement you can select a value from any of the preceding data type classifications.
   c. In the *Data Type* dropdown list, select the required value.

      If you have selected the type of data type as *Structure Element* or *Entity Element*, then in the *Data Type* dropdown list, select the required structure or entity in the CDS artifact. Use the *Type of Element* value help list to select the required element.
   d. Define the length and scale based on the selected data type.
6. In the editor toolbar, choose (Navigate Back).

15.3.4.3.1.5 Create a View with the CDS Graphical Editor

Use the graphical modeling capabilities of the CDS graphical editor tool in SAP Web IDE to create a design-time CDS view.

Context

A view is a virtual table based on the dynamic results returned in response to an SQL statement. You use the graphical modeling tools to create a view with the CDS artifact. After creating the CDS view, use this design-time view definition to generate the corresponding catalog object when you deploy the application that contains the view-definition artifact (or just the application's database module).

Procedure

1. Start SAP Web IDE in a Web browser.
2. In the *Workspace* view, select the required CDS artifact within which you want to model a CDS view.
3. In the context menu, choose *Open With* > *Graphical Editor*.
   The tool opens the CDS artifact in a graphical editor where you can create and define the CDS view.
4. Create a CDS view.
   a. In the editor toolbar, select (Create View).
   b. Drop the view node on the editor.
   c. Provide a name to the view.
5. Add local CDS artifacts as data sources.
   You can add local CDS artifacts, active CDS artifacts, or both as a data source in a view node. Local CDS artifacts are those entities that exist within the same artifact in which you are creating the CDS view. Active CDS artifacts are those artifacts that are already built and activated.
   a. Double-click the view node.
   b. If you want to add a local CDS artifact, in the editor toolbar, select + (Add Local Objects).
   c. Select the required entities that you want to add as data sources.
   d. Choose *OK*.

   > → Tip
   >
   > The active artifacts could also be synonyms pointing to artifacts in other HDI containers.

6. Add active CDS artifacts as data sources.
   Active CDS artifacts are those artifacts that are already built and activated.
   a. Double-click the view node.
   b. If you want to add an active CDS artifact, in the editor toolbar, select (Add Active Objects).
   c. In the *Find Data Source* dialog box, search and add activated artifacts from within the same HDI container, or search and add synonyms pointing to artifacts in other HDI containers.

The active artifacts include built and activated entities (catalog tables), CDS views, calculation views, and SQL views.

> **i Note**
>
> You cannot use calculation views with input parameters as data sources in CDS views.

d. Choose *Finish*.

e. Select the required menu option.

> **i Note**
>
> If you have more than one data source in the CDS view, either create a union, or a join between the data sources.

> **→ Tip**
>
> The active artifacts could also be synonyms pointing to artifacts in other HDI containers.

7. (Optional) Using elements from associated entities.

You can model a CDS view with entities, which are defined with associations to other entities. In such cases, you can use the elements from the associated entities in the CDS view definition. For using elements from the associated entities, it is not necessary to use associated entities as data sources in the CDS view.

a. Select the required entity.

b. Select the association column in the entity.

c. In the *Select Element(s)* dialog box, select the required elements from the associated entity.

You can use the elements from the associated entities in the CDS view definition. For example, as output columns, to define SQL clauses, perform SQL `GROUP BY` or `ORDER BY` operations, and more.

d. (Optional) Define a filter expression.

In the *Filter Expression* editor, provide a filter expression rule that defines how the elements from the associated entities are used.

When following an association (for example, in a view), it is possible to apply a filter condition; the filter is merged into the `ON`-condition of the resulting `JOIN`. The following example shows how to get a list of customers and then filter the list according to the sales orders that are currently “open” for each customer. In the example, the infix filter is inserted after the association orders to get only those orders that satisfy the condition `status='open'`.

> **‘≡› Sample Code**

```
view C1 as select from Customer {
  name,
  orders[status='open'].{id as orderId, date as orderDate}
};
```

Use the elements from the associated entities and the operators provided in the dialog box to build your filter expression.

e. Choose *OK*.

8. (Optional) Using elements from structure types.

You can model a CDS view with an entity that has a structure type. In such cases, you can use elements from the structure types in the CDS view definition.

a. Select the required entity.
b. Select the structure type element in the entity.
c. In the *Select Element(s)* dialog box, select the required elements from the structure type.

   You can use the elements from the structure types in the CDS view definition. For example, as output columns, to define SQL clauses, perform SQL `GROUP BY` or `ORDER BY` operations, and more.
d. Choose *OK*.

9. Define output columns.
   a. In the CDS view editor, select the required data source.
   b. Select the columns from the data source that you want to add to the output.
   c. In the context menu, choose *Add to Output*.
   d. In the *Select Element(s)* dialog box, provide an alias name to the output column, if required.
   e. Choose *OK*.

      The tool displays the output columns of the view under the *Columns* section (in the details pane) of the editor.
10. (Optional) Group by result set.

    Use one or more columns from the data source to perform a SQL `GROUP BY` operation.
    a. Select the data source.
    b. Select the required columns.
    c. In the context menu, choose *Add to Group By* to perform a SQL `GROUP BY` operation with the selected columns.

       You can also use an output column to perform a SQL `GROUP BY` operation. In the *Columns* section, select an output column, and in the context menu, choose *Add to Group By*.
11. (Optional) Order by result set.

    Use one or more output columns from the view to perform a SQL `ORDER BY` operation.
    a. In the details pane, expand the *Columns* section.
    b. Select the required columns.
    c. In the context menu, choose *Add to Order By* to perform a SQL `ORDER BY` operation with the selected output columns.
12. (Optional) Limit and offset the result set.

    You can use the SQL `LIMIT` clause and the SQL `OFFSET` clause in a CDS view. In the details pane, specify the required limit value and offset value as integers.
    a. In the *Limit* text field, provide the required limit value.

       Use the `LIMIT` clause to restrict the number of result sets to a specified limit.
    b. In the *Offset* text field, provide an integer value required offset value.

       Use the `OFFSET` clause to specify the number of records to skip before displaying the results sets defined by the `LIMIT` SQL clause.
13. (Optional) Create Subqueries.

    You can model CDS views with nested SQL queries (subqueries) to obtain the desired output. Subqueries help model CDS views for complex business scenarios.
    a. Double-click the CDS view.
    b. In the editor toolbar, choose (Sub Query).
    c. Drop the subquery node on the editor.
    d. Provide a name to the subquery node.

e. Double click the subquery node.
f. Define the subquery.

   Defining a subquery is similar to defining a view. Within a subquery, you can only creating entities, other subqueries, creating union of entities, creating union of subqueries, and creating union of an entity and a subquery.

   > → Tip
   >
   > You can use the breadcrumb navigation in the editor toolbar to switch between subqueries and the target CDS view.

14. Choose *Save* to save all your changes.
15. Build an SAP HANA Database Module.

    The build process uses the design-time database artifacts to generate the corresponding actual objects in the database catalog. The activation process generates a corresponding catalog object for each of the artifacts defined within another artifact; the location in the catalog is determined by the type of object generated.

    a. From the module context menu, choose *Build*.

Next Steps

Modeling CDS views for complex business scenarios could include layers of calculation logic. In such cases, it may require to perform certain additional tasks to obtain the desired output.

The following table lists some important additional tasks that you can perform to enrich the CDS view.

Requirement	Task to Perform
If you want to query data from two data sources and combine records from both the data sources based on a join condition.	Create Joins
If you want to combine the results of two more data sources.	Create Unions
If you want to create new output columns and calculate its values at runtime using an expression.	Create Calculated Columns
If you want to define relationships between CDS views.	Create Associations for CDS Views with CDS Graphical Editor
If you want to create CDS views with parameters that enable you to pass additional values to modify the results of the CDS view at runtime.	Create Parameters

Related Information

Create SQL WHERE Clause [page 1113]
Create SQL HAVING Clause [page 1113]
Create Calculated Columns [page 1114]

Create Joins [page 1114]
Create Unions [page 1116]
Creating Synonyms [page 1120]
Create Associations for CDS Views with CDS Graphical Editor [page 1117]
Create Parameters [page 1118]

15.3.4.3.1.5.1 Create SQL WHERE Clause

Define SQL `WHERE` clause in a view to filter and retrieve records from the output of a data source based on a specified condition.

Procedure

1. Open the required CDS artifact.
2. Select the CDS view in which you want to define the SQL `WHERE` clause.
3. Double-click the CDS view.
4. In the *Where* section, choose + (Add Where Clause).
5. In the *Where Clause Editor*, define the SQL `WHERE` clause condition using the required elements and operators.

   You can also use parameters in the expression.
6. Choose *OK*.
7. Choose *Save*.

15.3.4.3.1.5.2 Create SQL HAVING Clause

Define SQL `HAVING` clause in a view to retrieve records from the output of a data source, only when the aggregate values satisfy a defined condition.

Procedure

1. Open the required CDS artifact.
2. Select the CDS view in which you want to define the SQL `HAVING` clause.

> **i Note**
>
> You can create SQL `HAVING` clause only if you are using one or more output columns to perform a SQL `GROUP BY` operation on the selected view.

3. Double-click the CDS view.
4. In the *Having* section, choose + (Add Having Clause).
5. In the *Having Clause Editor*, define the SQL `HAVING` clause condition using the required elements and operators.

   You can also use parameters in the `HAVING` clause condition.
6. Choose *OK*.
7. Choose *Save*.

15.3.4.3.1.5.3 Create Calculated Columns

Create new output columns for a view and calculate the column values at runtime based on the result of an expression. You can use other column values, functions, or constants when defining the expression.

Procedure

1. Open the required CDS artifact.
2. Select the CDS view in which you want to create calculated columns.
3. Double-click the CDS view.
4. In the details pane, select the *Columns* section.
5. Choose + (Add Calculated Column).
6. Provide a valid expression.

   In the *Add Calculated Column* dialog box, provide the expression that defines the calculated column.
   a. In the *Name* text field, provide a name for the calculated column.
   b. In the expression editor, provide a valid expression using the required elements and operators.

      You can also use parameters in the expression.
   c. Choose *OK*.

15.3.4.3.1.5.4 Create Joins

Joins help query data from two or more data sources. It helps to limit the number of records, or to combine records from both the data sources, so that they appear as one record in the query results.

Context

After modeling a CDS view, you can include a `JOIN` clause in the CDS view definition.

Procedure

1. Open the required CDS artifact.
2. Select the CDS view in which you want to create a join for data sources.
3. Double-click the CDS view.

   > **i Note**
   >
   > You can create a join only if the CDS view has two or more data sources.

4. Select an entity.
5. Choose (Join).
6. Drag the cursor to the required data source.
7. Define the join properties.

   In the *Join Definition* dialog box, define the join properties and the join condition.

   a. In the *Type* dropdown list, select the required join type.
   b. In the expression editor, specify a valid join condition using the necessary element and operators.

      Using proposed join condition.

      The tool analyzes the data in the participating entities and proposes a join condition. Based on your requirement, you can create your own join condition or use the join condition that the tool proposes. In the *Join Definition* dialog box, choose *Propose Condition* to use the join condition that the tool proposes.

      > **i Note**
      >
      > The tool does not propose any join condition for cross joins.

   c. Choose *OK*.

Related Information

Supported Join Types [page 1115]

15.3.4.3.1.5.4.1 Supported Join Types

When creating a join between two data sources, you specify the join type. The following table lists the supported join types in CDS graphical editor.

Join Type	Description
Inner	This join type returns all rows when there is at least one match in both the data sources.

Join Type	Description
Left Outer	This join type returns all rows from the left data source, and the matched rows from the right data source.
Right Outer	This join type returns all rows from the right data source, and the matched rows from the left data source.
Full Outer	This join type displays results from both left and right outer joins and returns all (matched or unmatched) rows from the tables on both sides of the join clause.
Cross Joins	This join type displays results of all possible combinations of rows from the two tables. Cross join is also called a cross product or Cartesian product.

15.3.4.3.1.5.5 Create Unions

Creating unions help combine the result sets of two or more data sources.

Context

After modeling a CDS view using the graphical modeling tools, you can include a `UNION` clause in the CDS view definition.

Procedure

1. Open the required CDS artifact.
2. Select the CDS view in which you want to perform the union operation.
3. Double-click the CDS view.

   > **i Note**
   >
   > You can create a union only if the CDS view has two or more data sources.

4. Select a data source.
5. Choose (Union).
6. Drag the cursor to the required data source.

   This operation creates a *ResultSet* node for each of the data sources.
7. Add more data sources to the union.

   You can perform union operation on multiple data sources (more than two data sources). If you have already created a union of two data sources, then to create a union of these two data sources with the third data source, you use the ResultSet node.

a. Select a *ResultSet* node.

   You cannot select the *DefaultResultSet* node to union records of multiple data sources.

b. Choose (Union).

c. Drag the cursor to the required data source.

8. Add columns to the union output.

   The columns you add to the *DefaultResultSet* node are the output columns of the union.

   a. Select the data source.

   b. Select the columns in the data source you want to add to the output.

   c. In the context menu, choose *Add to Output*.

> **i Note**
>
> Union operation combine result sets of two or more SELECT statements. It is necessary that each result set (SELECT statements) has the same number of columns, have similar data types, and also the columns in each result sets are in same order.

15.3.4.3.1.5.6 Create Associations for CDS Views with CDS Graphical Editor

Associations define relationships between CDS views, and you create association in the CDS view definition.

Context

In a CDS view definition, associations can be used to gather information from the specified target entities. You can use the CDS graphical editor to create associations only if the views are simple CDS views. This means that, the CDS views involved in the association must contain only a single entity as a data source.

Procedure

1. In the *Workspace* view, select the required CDS artifact.
2. In the context menu, choose *Open With* > *Graphical Editor*.
3. Create associations between views.

   a. Select the required CDS view.

   b. Choose (Association).

   c. Drag the cursor to another CDS view in the CDS artifact with which you want to create the association.

4. Provide association details.

   a. In the *Association Details* dialog box, provide further details for the association.

b. In the *Name* text field, provide a name for the association.
c. (Optional) If you want the tool to propose a condition for the association, choose *Propose Condition*.
d. In the expression editor, provide the required `ON` condition.

   In the `ON` condition, only elements of the source or the target CDS view can be used; it is not possible to use other associations. The `ON` condition may contain any kind of expression. Use the elements and operators from the dialog box, when defining the `ON` condition expression.

5. Define cardinality, if required.
   a. When using an association to define a relationship between CDS views in a CDS artifact, use the cardinality to specify the type of relation, for example, one-to-one (to-one) or one-to-many (to-n); the relationship is with respect to both the source and the target of the association.
   b. In the *Source Cardinality* and *Target Cardinality* dropdown lists, select the required cardinality values.
6. Choose *OK*.
7. Choose *Save*.

15.3.4.3.1.5.7 Create Parameters

Use the CDS graphical editor in SAP Web IDE to define CDS views with parameters. The parameters enable you to pass additional values to modify the results of the CDS view at runtime.

Context

After creating a parameter, you can use the parameter in the CDS view as an output column, in calculated column expressions, in the SQL `WHERE` clause, in the SQL `Group By` clause, joins, unions, and more.

Procedure

1. Open the required CDS artifact.
2. Select the CDS view in which you want to create parameters.
3. Double-click the CDS view.
4. In the details pane, choose + in the *Parameters* section.
5. Define the parameter.

   In the *Parameter Properties* dialog box, define the parameter.
   a. In the *Name* text field, provide a name for the parameter.
   b. In the *Category* dropdown list, select the required category.
   c. In the *Type* dropdown list, select the required data type.

      If you have selected the type of data type as *Structure Element* or *Entity Element*, in the *Type* dropdown list, select the required structure or entity from within the same CDS artifact. Use the *Type Of* value help list to select the required element.
   d. Define the *Length* and *Scale* based on the selected data type.

e. Choose *OK*.

6. If you want to add the parameter as an output column of the CDS view, right-click the parameter and choose *Add As Output Column*.

15.3.4.3.1.6 Preview Output of CDS Views and Entities

After modeling and activating a CDS view or an entity, you can preview the output data using the SAP Database Explorer.

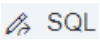
Context

You can preview output data of CDS views or entities in simple in tabular format, or you can preview output data in graphical representations, such as bar graphs, area graphs, and pie charts.

You can also export and download the output data to `.csv` files. The Database Explorer also allows you to preview the SQL query that the tool generates for an activated CDS artifact.

Procedure

1. Start SAP Web IDE in a Web browser.
2. In the *Workspace* view, select the required CDS artifact to preview the output of its views or entities.
3. In the context menu, choose *Open With* > *Graphical Editor*.
   The tool opens the CDS artifact in a graphical editor.
4. In the editor, select the required view or entity and choose (Open Data Preview).
5. Apply filters.
   a. For CDS views, if you want to apply filters on columns and view the filtered output data, choose (Add Filter).
   b. Choose *Add Filters*.
   c. Choose a column and define filter conditions.
6. Export output data, if required.
   If you want to export the raw data output to a .csv file,
   a. In the toolbar, choose (Download).
   b. In the *Delimiter* dropdown list, select the required delimiter that the tool must use to separate the values in the .csv file.
   c. Choose *Download*.
7. View SQL query for the entity or view, if required.
   a. If you want to view the SQL query that the tool generates for the activated CDS artifact, in the toolbar, choose *SQL*.

b. In you want to view and execute the SQL query in SQL editor, choose  (Edit SQL Statement in SQL Console).

8. Preview output in graphical representations.

   The tool supports bar graph, area graph, pie chart and table charts, and other graphical representations to preview the output of a CDS view or entity.

   a. In the menu bar, choose *Analysis*.
   b. Configure the axis values by dragging and dropping the required columns to the *Label Axis* and the *Value Axis*.

      The tool displays the output data in graphical representation. Select the required chart icons in the menu to view the output in different graphical representation.

   c. In the menu bar, choose (Display Settings) to toggle legends and values.

Related Information

Create a View with the CDS Graphical Editor [page 1109]
Create an Entity with the CDS Graphical Editor [page 1098]

15.3.4.3.2 Creating Synonyms

Synonym are design time files in the repository that provides data independence. After creating a synonym, you can bind this synonym with tables in different schemas.

This means that, in XS advanced, irrespective of which user or which schema holds the table, you can continue using the applications without any modifications. The complete definition of a synonym is contained in two design time files:

- `hdbsynonym` - A synonym declaration (with an optional default configuration).
- `hdbsynonymconfig` - An explicit configuration of the synonym's target.

This section describes creating the above two synonym definition files without writing any synonym definition code and the prerequisites to creating such a database synonym.

Related Information

Create a Database Synonym [page 1121]

15.3.4.3.2.1 Create a Database Synonym

Use the synonym editor in SAP Web IDE to create and define synonyms without writing any synonym-definition code.

Prerequisites

- You have defined a user provided service with an underlying database user. The user must have credentials to access all schemas required in the project. You use the following syntax to define the user provided service.

  Sample Code

  ```
  xs create-user-provided-service <SERVICE_INSTANCE> -p
  "{\"host\":\"<host_name>\",\"port\":\"<port_number>\",\"user\":\"<user_name
  >\",\"password\":\"<USER_PASSWORD>\",\"tags\":[\"hana\"] }"
  ```

- You have correctly configured all services including the user provided service in the mta.yaml file. This allows using the user provided service within the project. A sample mta.yaml configuration is shown below.

  Sample Code

  ```
  ID: DwProject
  version: 1.2.3
  modules:
   - name: DwDbModule
     type: hdb
     requires:
       - name: DwHdiService
         properties:
           TARGET_CONTAINER: ~{hdi-service-name}
       - name: CrossSchemaService
  resources:
   - name: DwHdiService
     type: com.sap.xs.hdi-container
     properties:
       hdi-service-name: ${service-name}
   - name: CrossSchemaService
     type: org.cloudfoundry.existing-service
     parameters:
       service-name : CROSS_SCHEMA_SERVICE_SYSTEM_ERP
  ```

- You have created a grantor file (.hdbgrants) that specifies the necessary privileges to access external tables.

  Sample Code

  ```
  {
    "CROSS_SCHEMA_ACCESS_SERVICE_ERP": {
      "object_owner" : {
        "schema_privileges":[
          {
            "reference":"ERP",
            "privileges_with_grant_option":["SELECT"]
          }
  ```

```
      ]
    },
     "application_user" : {
      "object_privileges":[
        {
          "schema":"ERP",
          "name": "TABLE",
          "privileges":["SELECT"]
        }

      ]
    }
  }
}
```

Procedure

1. Start SAP Web IDE in a browser.
2. Display the application project to which you want to create a database synonym.

   SAP Web IDE creates an application within a context of a project. If you do not already have a project, there are a number of ways to create one, for example: by importing it, cloning it, or creating a new one from scratch.

   a. In the SAP Web IDE, choose *File* > *New* > *Project from Template*.
   b. Choose the project template type.

      Currently, there is only one type of project template available, namely: *Multi-Target Application Project*. Select *Multi-Target Application Project* and choose *Next*.
   c. Type a name for the new MTA project (for example, **myApp** and choose *Next* to confirm).
   d. Specify details of the new MTA project and choose *Next* to confirm.
   e. Create the new MTA project; choose *Finish*.
3. Select the HDB module in which you want to create synonyms.
4. Browse to the *src* folder.
5. Create a `.hdbsynonym` file.

   a. In the context menu of the *src* folder, choose *New* > *File*.
   b. Provide a name for the synonym declaration file.

      > **i Note**
      >
      > You must add `.hdbsynonym` as a suffix to the file name.

   c. Choose *OK*.

      The tool opens a new synonym editor where you can declare all the synonyms that you require.
   d. Choose +.
   e. In the *Find Data Sources* dialog box, search the required data source for which you want to declare a synonym.

      You can declare multiple synonyms in a single file.

      > **i Note**
      >
      > You can also directly enter the name of the data source in the *Object Name* text field.

f. Choose *Save*.

6. Build an HDB module.

   The build process uses the design-time database artifacts to generate the corresponding actual objects in the database catalog.

   a. From the module context menu, choose *Build* > *Build*.

15.3.4.3.3 Native DataStore Object

A native datastore object is a specialized CDS artifact that allows merging of data sets, including delta sets, into their reportable content.

This can be achieved with a simple move or aggregation and it's also possible to mark records in the delta loads for deletion. In addition, you can identify and process the delta data set for connected NDSOs. The object also provides interoperability between native Data Warehouses and SAP BW/4HANA. The native datastore object is only available with DWF 2.0 or higher.

For more information about creating an NDSO, see the *SAP HANA Data Warehousing Foundation - Native DataStore Object Administration Guide* → *Creating Native DataStore Object*.

Related Information

https://uacp2.hana.ondemand.com/viewer/1e4f857a22aa477081d41d3b6fa48d99/2.0.0.0/en-US

15.3.4.3.4 Transforming Data Using Flowgraphs

Use the Flowgraph Editor to model the transformation of your data.

You can better virtualize data by transforming it using a flowgraph editor and various transforming nodes. Use the nodes to design a flowgraph to retrieve data from an external system, transform it, and view it.

The Flowgraph Editor helps model the transformation of your data through various nodes such as Join, Projection (Filter), Union, Aggregation, and so on.

For more information about data virtualization using the Flowgraph Editor, see *The Modeling Guide for SAP HANA Smart Data Integration and SAP HANA Smart Data Quality* → *Transforming Data*.

> **i Note**
>
> The SAP HANA smart data integration and SAP HANA smart data quality options add much more functionality to flowgraph design, such as data cleansing, geocoding, masking, date and row generation, and so on, as well as many capabilities outside of flowgraphs. Be aware that you need additional licenses for SAP HANA options and capabilities. For more information, see Important Disclaimer for Features in SAP HANA Platform, Options and Capabilities.

15.3.4.3.5 Developing Text Analysis Artifacts

Text analysis allows you to discover and classify entities and relationships in your documents and unstructured text content. You can specify your own entity types to be used with text analysis by creating custom text analysis dictionaries and extraction rules.

Related Information

SAP HANA Text Analysis Developer Guide

15.3.4.4 Connect to an External HDB Service

You can configure a connection to an external SAP HANA database (HDB) service. HDB services offer an option to provide access privileges for objects that are not contained in your current deployment target schema.

Prerequisites

Make sure you have an MTA project with a valid `mta.yaml` file.

Context

There are 2 ways to access objects in a schema that is not your current deployment target schema:

- Binding to a user-provided service
  You can either select an existing user-provided service or create a new one. User-provided services can be used to access objects in HDI container schemas and non-HDI container schemas.
- Binding to an existing HDI container service
  The existing HDI container service refers to the schema to which you want to connect. HDI container services only allow access to objects in the HDI container of the HDI container service.

The results of connecting to an external HDB service are:

- The creation of the corresponding dependency configuration in the `mta.yaml` file.
- If you choose to create a new user-provided service, the service is created in the target XSA space that was configured in the project settings.

Procedure

1. Open SAP Web IDE.
2. In the *Workspace* view, right-click the SAP HANA Database Module in which you want to configure the `mta.yaml` file.
3. Choose *New* > *SAP HANA Service Connection*.
4. Configure the `mta.yaml` file according to one of the following scenarios.

Scenario	Steps
Adding an existing non-HDI container service	1. In the *Add SAP HANA Service Connection* dialog box, from the *Type* field, select *Non-HDI Container*. 2. From the *Action* field, select *Use an existing user-provided service* and select the required service. 3. Optional - Click to test the connection to SAP HANA Database. 4. Choose *Add*. The user-provided service is added and the `mta.yaml` file is configured.
Creating a new non-HDI container service	1. In the *Add SAP HANA Service Connection* dialog box, from the *Type* field, select *Non-HDI Container*. 2. From the *Action* field, select *Create a new user-provided service* and enter a name. 3. Provide details of the SAP HANA database that the selected user-provided service refers to. 4. Optional - Click to test the connection to SAP HANA Database. 5. Choose *Add*. The user-provided service is created in the Cloud Foundry space and the `mta.yaml` file is configured.
Adding an existing HDI container service	1. In the *Add SAP HANA Service Connection* dialog box, from the *Type* field, select *HDI Container*. The list of all HDI services is displayed. 2. Select the required HDI service that you want to configure in the `mta.yaml` file. 3. Optional - Click to test the connection to SAP HANA Database. 4. Choose *Add*. The service is added and the `mta.yaml` file is configured.

15.3.4.5 Build an HDB Module

The build process deploys the design-time database artifacts and generates the corresponding actual objects in the SAP HANA database.

Procedure

From the module context menu, choose *Build*.

The steps and messages of the build process are displayed in the console that opens automatically at the bottom of the browser window. You can show and hide the console by choosing *View* > *Console* from the main menu.

> **Note**
>
> Sometimes the build fails because there is an error in the MTA descriptor (mta.yaml). The error messages displayed in the console indicate the corrections you should make in the descriptor. For more information about the descriptor syntax, see Inside an MTA Descriptor [page 1079].
>
> After making the corrections, rebuild the module.

By default, the build process generates a database schema with a unique name derived from the `schema` parameter defined in `mta.yaml`. If you want the schema name to be exactly the same as defined, set the `makeUniqueName` parameter to `false`.

For example:

> **Sample Code**

```
- name: hdi-container
  parameters:
      config:
         schema: mySpecialSchema
         makeUniqueName: false
  properties:
      hdi-container-name: ${service-name}
  type: com.sap.xs.hdi-container
```

> **Note**
>
> The `schema` and `makeUniqueName` parameters only work when deploying a fresh db module. Once the module has been deployed with any name, it cannot be changed without deleting the service (data loss) and doing a fresh deployment.

However, if you do so and use the same schema name in different projects, naming conflicts might occur in the database during builds.

> **Note**
>
> If you have an MTA extension file with the name `dev.mtaext` that extends the mta.yaml file in the MTA project root, its contents will be merged with the `mta.yaml` file when running the module.

Note

By default, the HDI deployer removes any object from the HDI container that does not exist in the database module. To prevent the HDI deployer from deleting an artifact that is not in the developer's database module, you need to set the `auto_undeploy` property to `false` in the `mta.yaml` file.

For development, set to `false` the `auto_undeploy` property in the MTA extension file (`dev.mtaext`) that will be automatically merged with the definitions provided in the `mta.yaml` file for building an HDB module and running operations. Do not change the `mta.yaml` file.

Sample Code

Editing the `dev.mtaext` file

```
ID: anyname
_schema-version: '2.1'
extends: mtaname

resources:
- name: hdi_db
  properties:
    hdi-container-name: ${service-name}
    HDI_DEPLOY_OPTIONS:
       auto_undeploy: false
   type: com.sap.xs.hdi-container
   parameters:
     service-name: my-service-name
```

For building and deploying to the target environment, set the `auto_undeploy` property to `false` in the `mta.yaml` file.

Sample Code

Editing the `mta.yaml` file

```
modules:
- name: db
   type: hdb
   path: db
   requires:
    - name: hdi_db
      properties:
        HDI_DEPLOY_OPTIONS:
            auto_undeploy: false
```

Related Information

Selective Build of HDB Modules [page 1128]
Developing SAP HANA Database (HDB) Modules [page 1090]
HDI Deploy Options

15.3.4.5.1 Selective Build of HDB Modules

You can optimize the build process of an SAP HANA Database (HDB) module by building only selected artifacts.

The full build of an HDB module containing many database artifacts can be time-consuming. Sometimes, when developing a module, you don't need to build the whole module. You can save time by building only the artifacts that you are currently working on.

In the module folder, select all the artifacts that you want to build, and choose *Build Selected Files* from the context menu.

> **i** Note
>
> To use selective build effectively, note the following:
>
> - Selective build works only if the selected files don't have dependencies on other artifacts that are not yet built or are not included in the selection.
> - Selective build includes only the files that are selected explicitly. To include hidden files in a build, choose the option *Show Hidden Files* from the *View* menu, and then select the required files.

Deployment options

You have additional options to deploy your database artifacts.

Force Deployment

By default, only changed artifacts are deployed. If you want to deploy a single artifact that hasn't been changed, right-click it and choose *Build Selected Files*.

Simulate Build

You can test the integrity of your HDB module without deploying all its artifacts by simulating a build. Open the *Problems* view, select the HDB module, and click *Analyze and Display Problems*. This executes the build simulation command and displays the errors that occurred.

Related Information

Build an HDB Module [page 1126]
Developing Database Artifacts [page 1093]

15.3.5 Developing Node.js Modules

A Node.js module is a collection of related JavaScript files and service definitions that implement the business logic of your application.

Prerequisites

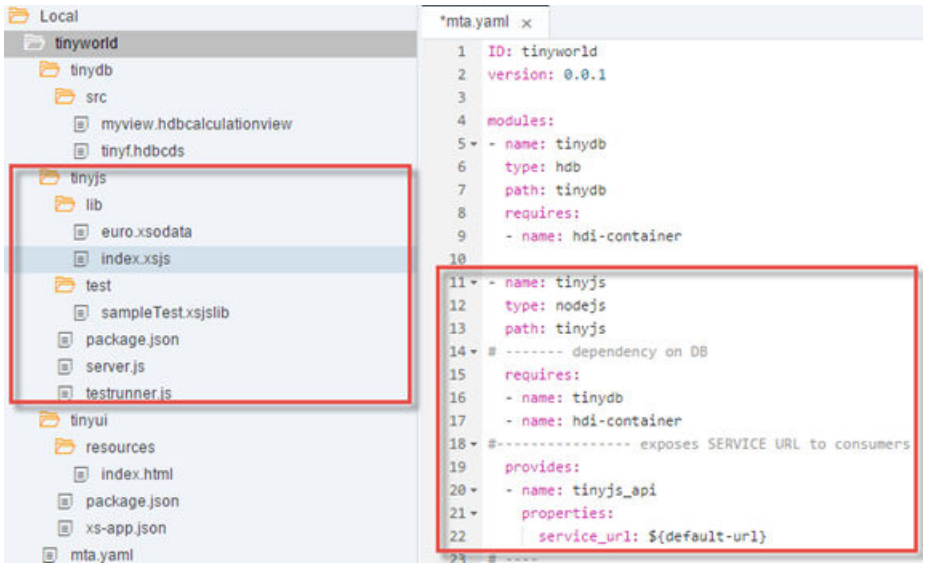
You have enabled the optional *Tools for Node.js Development* extension. For instructions, see Enable SAP Web IDE Extensions [page 1066].

To develop Node.js modules, perform the following steps:

- Create a Node.js Module [page 1130]
- Implementing Node.js files [page 1130]
- Inside an MTA Descriptor [page 1079]
- Run Node.js Modules [page 1133]
- Debug Node.js Modules [page 1137]

Module Folder Structure

The following figure depicts a sample Node.js module folder structure alongside its corresponding entry in the `mta.yaml`.



Folder	Description
<module name>	Contains the main .js file, which by default is `server.js`, and the `package.json` file (description and dependencies of Node.js modules). You can place other .js files anywhere in this folder.
lib	Created only if XSJS support is enabled for the module. Contains .xsjs (SAP HANA XS JavaScript) source files.
test	Created only if XSJS support is enabled for the module. Contains XSUnit test (.xsjslib) files.
tests	This folder is created for modules without XSJS support. It contains .js test files.

Implementing Node.js files

In the root module folder, create and implement the required .js files.

Related Information

Developing Multitarget Applications [page 1078]
https://nodejs.org/en/
Node.js Data Services in XSJS Compatibility Mode
Create a Node.js Module [page 1130]
Import a Node.js Module [page 1132]

15.3.5.1 Create a Node.js Module

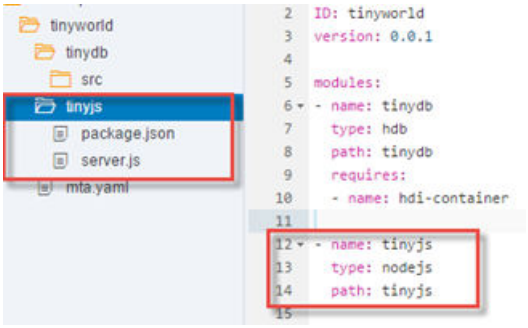
You can create new Node.js modules.

Procedure

1. From the project context menu, choose *New* > *Node.js Module* .
2. Follow the wizard steps to enter the module properties.

   If you want to run your module in the XS JavaScript (XSJS) compatibility mode, select *Enable XSJS support* .

   A new Node.js module with the specified name is created in your project, and a corresponding section is added to the MTA descriptor (`mta.yaml`). For example:



The default files for the new module contain basic Hello World code, which varies according to whether you enabled XSJS support.

Related Information

Developing Node.js Modules [page 1129]
Import a Node.js Module [page 1132]

15.3.5.1.1 Implement OData Service Definitions

Context

To expose a part of the data model, such as a table or a view, using OData, you should create a corresponding OData service definition in an `.xsodata` file. Since these files must be deployed to a JavaScript runtime, you should place them in a Node.js module.

Procedure

1. From the context menu of a Node.js module's `lib` subfolder, choose *New* > *File*.
2. In the *New File* box that opens, enter a file name with the extension `.xsodata`.
3. Double-click the file to open it in the dedicated editor. Use the Ctrl-Space key combination for hints while editing.

   For more information, see Defining OData v2 Services for XS Advanced JavaScript Applications

15.3.5.2 Import a Node.js Module

You can import archived Node.js modules.

Prerequisites

The `.zip` module archive that you want to import, which was exported from another MTA project, is available in the file system.

Procedure

1. From the root folder of the project, choose *File* > *Import* > *File or Project*.
2. Click *Browse* to locate and select your archive, and choose *Open*. The file name appears in the *File* field.

   The destination folder is displayed in the *Import to* field. To change this folder, choose *Select Folder*, and browse to the required folder, or create a new folder.

   The specified folder, containing the artifacts extracted from the archive, is created in the project.
3. To make the imported folder a proper module in your project, you need to convert it into a module of the matching type. From the folder context menu, choose *Convert To*, and then the type of the target module.

   > **i Note**
   >
   > The conversion process does not check whether the imported folder structure matches the selected module type. The process does not generate the module artifacts according to the selected type.

   The imported module becomes a part of your MTA project, and the module entry is added to the MTA descriptor.

Related Information

Developing Node.js Modules [page 1129]
Create a Node.js Module [page 1130]

15.3.5.3 Run Node.js Modules

To create a running application in XS Advanced and resolve all dependencies defined in `mta.yaml`, run each Node.js module in your project.

Prerequisites

Before running a module, make sure that:

- All the relevant dependencies of the module are defined in the MTA descriptor (mta.yaml).
- All the modules and resources on which the module is dependent are implemented and available .

> **i Note**
>
> You don't need to build your module explicitly before running, because it will be built automatically during the run process.

Procedure

From the module context menu, choose *Run*, then one of the following options:

Option	Description
Run Configurations	Create a new or modify an existing run configuration. For more information, see Create Run Configurations for Node.js Modules [page 1135].
Run as Node.js Application	Run the main .js file, which by default is `server.js`.

> **i Note**
>
> If you have an MTA extension file with the name `dev.mtaext` that extends the `mta.yaml` file in the MTA project root, its contents will be merged with the `mta.yaml` file when running the module.

Results

When the build finishes, its result is available in your workspace in the `node_modules` folder.

> **i Note**
>
> When you run a module for the first time, it is pushed to XS Advanced and bound to the required XS Advanced services that are defined in the MTA descriptor, which can take some time. After this, the run starts more quickly. If you modify only the source files, or `xs-app.json` in HTML5 modules, the module remains in XS Advanced, and the changes are injected directly into the running module.

Related Information

Show Dependency Updates in package.json [page 1134]
Use the Run Console [page 1134]
Test Node.js Modules [page 1136]
Create Run Configurations for Node.js Modules [page 1135]

15.3.5.3.1 Show Dependency Updates in package.json

You can check for dependency updates in the external dependencies defined in the `package.json` file.

Procedure

1. Select your Node.js module and from the toolbar choose *Build* > *Show Dependency Updates*.
2. Then go to the *Problems* view to see the outdated dependencies.

Related Information

Run Node.js Modules [page 1133]

15.3.5.3.2 Use the Run Console

Progress and status messages, generated during the run process, as well as the log records, are displayed in the Run console.

The console opens automatically in the main browser tab. You can show and hide the console by clicking the *(run console)* icon in the bottom pane's toolbar.

The left pane of the console displays a list of all running modules in the current MTA project along with the corresponding run configurations. Each entry is accompanied by a status icon with a tooltip. Click the run configuration name below a module name to view its most recent run log.

The controls displayed at the top of the console enable you to perform the following tasks:

- If the application URL is visible, click it to access the running application in the browser.
- To view the detailed server log of a run process, click *Logs*.
- To clear the log in the console, click *(clear the log)* .
- To stop running the module, click *(stop)* .
  This causes the module to be removed from the runtime system, so that if you run the module again, the run process starts from the beginning.

- To run the module using a different configuration, select the configuration and choose the *(run)* button in the task bar.

Related Information

Run Node.js Modules [page 1133]
Test Node.js Modules [page 1136]

15.3.5.3.3 Create Run Configurations for Node.js Modules

You can configure how to run, test, and debug Node.js modules in your project.

Procedure

1. From the context menu of a Node.js module in your project, choose *Run* > *Run Configurations*.
2. To create a new run configuration for a module, click + > *Node.js Test*.
   A new configuration with a default name appears under the selected category in the left pane.
3. Select a configuration to modify its name, and edit its properties as follows.

   General tab

Property	Description
Start with package.json script	Run one of the scripts listed in the `package.json` file of the module, for example, a test script generated in an XSJS file.
Start with application file	Run a .js file of your module.
Application File	The name of .js file to run.
Debugging Enabled	If debugging is enabled, you can select or unselect the *Break on first statement*.
Break on first statement	This is only possible if you start your application with *Debugging Enabled*. Use this option to stop the application on the very first statement, even during the bootstrapping or initializing phase.

Property	Description
Script Name	Select a script name: • *start* Start the application. • *test* Start the application with test execution. • *test-coverage* Start the application with test execution and show code coverage. This script requires more resources than others.
Arguments	(Optional) The arguments to be passed to the script.
Open a new browser window	Select whether to open the script in a new browser tab. By default, this option is unselected for .js files.
Service Path	(Optional) Enter a relative path to append to the server root. The browser is redirected to this path.

4. To save the configuration and run your module using this configuration, choose *Save and Run*. To save all configurations without running, choose *OK*. To discard all your changes, choose *Cancel*.

   These settings are persistent and are used every time that you run your application with the selected run configuration, until you edit or delete the settings.

Related Information

Run Node.js Modules [page 1133]

15.3.5.4 Test Node.js Modules

Build and test your JavaScript application with unit tests.

Procedure

1. Add unit tests in the `test` folder structure.
2. Execute your test run configuration, which could be named, for example, *Run script test*.
3. From the context menu of your module, select *Run* > *Run script test*.
4. In your run configuration, specify whether to run the test with code coverage.
5. Review the test results.

Results

Each JavaScript module must provide its own unit tests. The JavaScript module template comes with a sample unit test. Use this test to get familiar with the test functionality and the *(test results)* pane. By default, the *(test results)* pane shows *Test Results*, *Stack Trace*, and *Coverage*. If you have specified it in the run configuration, code coverage, which enables you to write targeted unit tests for uncovered code, is shown in the editor. To disable this function, edit your *(settings)* .

You can also use the *(test results)* pane to *(export)* or *(delete)* all test results for the selected module. Your workspace is connected to your *(test results)* pane. The results you see are for the currently selected module, which is also referenced in the pane as the path to your current module (`/<projectname>/<modulename>`). If you miss the latest test result, select *(refresh)*. Also check the logs to see if your test execution failed; if it has, no test result is available.

Related Information

Run Node.js Modules [page 1133]

15.3.5.5 Debug Node.js Modules

Debug your application on demand using the debugger panel.

Prerequisites

Before debugging a Node.js module in the debugger panel, make sure that the following conditions are true:

- You have run your application successfully.
- Your Node.js application is running on Node.js 8.1 or higher.
  If you use a different runtime version, you can debug your application using command-line tools.

> **i Note**
>
> To debug the initializing phase of your application, configure a run configuration and make sure to select *Break on first statement*

Procedure

1. Open the *(debugger)* panel.
2. *(attach)* your application.

3. Choose your multi target application and select a debug target.

> → Tip
>
> If you want to debug a test do the following:
>
> Create a run configuration that starts with an application file and select *rungulp.js* as the application file. Then select *Debugging Enabled* and *Break on first statement* and run your application.

Results

When the application reaches a breakpoint, you are notified that the execution is suspended. In this panel, you can perform the regular debugging tasks, such as viewing the call stack, examining the variables, stepping in and out of the functions, and so on. In the debugger console, you can also perform interactive evaluation of JavaScript statements in the context of the current stack frame.

Related Information

Conditional Breakpoints [page 1138]
JSON Data Preview [page 1139]
Pause on Exceptions [page 1140]
Debug an XS Advanced JavaScript Application Using Command-Line Tools [page 715]

15.3.5.5.1 Conditional Breakpoints

If you open the context menu on a breakpoint, select *Edit Breakpoint Condition* to enhance that breakpoint with a condition.

Breakpoint Status	Description
Enabled standard breakpoint	Shape outline: blue Shape fill: light blue You have enabled the breakpoint, either explicitly, which means one by one, or globally. To enable breakpoints, go to the *Breakpoints* section and select the corresponding checkbox or activate breakpoints globally by pressing the ⇨ *(activate breakpoints)* button. Explicitly disabled breakpoints are not activated using the global option. You need to enable them explicitly.

Breakpoint Status	Description
Enabled conditional breakpoint	Shape outline: orange Shape fill: light orange You have defined an expression for a breakpoint. If the condition is met, the application suspends when it reaches the breakpoint.
Disabled standard and conditional breakpoint	Shape outline: blue or orange Shape fill: transparent You have disabled the breakpoint, either explicitly, which means one by one, or globally. To disable breakpoints, go to the *Breakpoints* section and select the corresponding checkbox or deactivate breakpoints globally by pressing the *(deactivate breakpoints)* button. Explicitly disabled breakpoints are not activated using the global option. You need to enable them explicitly.
Checkmark	Shape outline and fill are not affected. You set the breakpoint successfully on a running debugging session. The checkmark indicates that the corresponding source file has been loaded. As a consequence, the application suspends when it reaches the breakpoint. For a conditional breakpoint, the user-defined expression has to be met. A breakpoint without a checkmark means, there is no debug session running, the source file has not been loaded, or the breakpoint could not be successfully resolved. An unresolved breakpoint may occur when the workspace resource version and the currently executed resource version don't match.

15.3.5.5.2 JSON Data Preview

When you are inspecting variables in the *(debugger)* panel you can see the current JSON for a complex variable. To see the JSON, select a variable and then *(json data preview)*. The JSON might contain a variable with the value `"<object variable not loaded>"`. In such a case, the complex variables are nested and only those that you already inspected are contained in this JSON.

Here is an example:

```
{
  "Local": {
    "this": {},
    "exports": "<object variable not loaded>",
    "require": "function require(path) {",
    "module": "<object variable not loaded>",
    "__filename": "/path/to/your/app/server.js",
    "__dirname": "/path/to/your/app",
    "http": "undefined",
    "port": "undefined"
  }
}
```

Objects are loaded when you expand them in the nested overview. In our example you already expanded `Local`. If you expand `exports` your JSON file grows.

15.3.5.5.3 Pause on Exceptions

You have started your application and attached the debugger. If the application should be paused in the event of an exception, go to the part of the *(debugger)* panel where you (de-)activate breakpoints. Select *(pause on exceptions)* and then one of the options.

If you decide to let your application pause in the event of an exception, the line where the exception occurs is highlighted.

15.3.6 Developing Java Modules

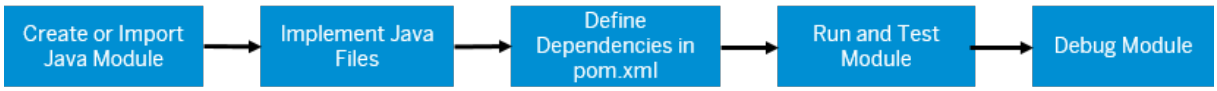
A Java module is a collection of related Java files and service definitions. Java modules implement the business logic of your application, either instead of or in addition to Node.js modules. A Java module can be either a Java Web Archive (WAR) or Java Archive (JAR) built with Apache Maven.

Prerequisites

- You have enabled the optional *Tools for Java Development* feature. For instructions, see Enable SAP Web IDE Extensions [page 1066].
- You have an internet connection.

Workflow

To develop Java modules, perform the following steps:



- Create a Java Module [page 1142]
- Implementing Java Files [page 1141]
- Define Dependencies [page 1143]
- Run a Java Module [page 1144]
- Debug Java Modules [page 1147]

Module Folder Structure

The default files of a new Java module (except Web application with OData support) already contain a basic, ready-to-run "Hello World" servlet. The module is structured following the Apache Maven standard directory layout.

Using a settings.xml File

If you decide to use a `settings.xml` file to provide project-specific Apache Maven settings, be aware that this file overrides default settings. During an update, changes to defaults do not overrule your `settings.xml`. Make sure you read the release notes to identify possible gaps between your `settings.xml` and the recommended default settings.

If needed, create the `settings.xml` next to the `pom.xml` on the same level. For more information about this optional step, see Define Dependencies [page 1143]

Implementing Java Files

In the `src/main/java` folder, create and implement the required Java files.

> → Tip
>
> While you are working on a Java file in a Java module, you can get assistance from the following features:
>
> - Code Assist
>   Provides code proposals as explained below.
>   In a Java file, press `Ctrl` + `Space` for Java keyword artifact and local objects suggestions.
> - Code Validation
>   Provides code validation as described below.
>   Problems are displayed in the file gutter. In addition, the problems view at the bottom of the screen displays complete problems in your Java module.

Related Information

Apache Maven - Introduction to the Standard Directory Layout
The SAP HANA XS Advanced Java Run Time [page 716]
Developing Multitarget Applications [page 1078]

15.3.6.1 Create a Java Module

You can create a new Java module by following the steps below.

Procedure

1. From the project context menu, choose *New* > *Java Module*, and enter a module name.
2. Define your basic module settings. Depending on your application needs, choose one of the Java module templates. To see all the available templates, choose *All Categories* from the *Category* dropdown list.

Option	Description
Simple Web Application	Creates a simple Web application with a standard Maven directory layout. Produces a `*.war` file that contains a basic ready-to-run "Hello World" servlet.
OData V2 Service	A Java Web application module that exposes an OData V2 service. The application is based on the SAP BTP SDK for service development. Produces a `*.war` file and provides an OData V2 end-point.
OData V4 Service	A Java Web application module that exposes an OData V4 service. The application is based on the SAP BTP SDK for service development. Produces a `*.war` file and provides an OData V4 end-point.
Multi-Module Web Application	Creates a Web application consisting of two Java modules, common and web. The Maven build produces `*.war` and `*.jar` files with dependencies for deployment.
Spring Boot Application	Creates a module that is build with the Spring Boot framework using Maven. Produces a `*.jar` file that brings its own runtime.

3. Define other settings. If the default values don't fit your needs, change the Maven POM settings.

Related Information

Import a Java Module [page 1143]

Maintaining OData Services in XS Advanced [page 522]

15.3.6.2 Import a Java Module

You can import an archived Java module by following the steps below.

Prerequisites

The `.zip` module archive that you want to import, which was exported from another MTA project, is available in the file system.

Procedure

1. From the root folder of the project, choose *File* > *Import* > *File or Project*.
2. Click *Browse* to locate and select your archive, and choose *Open*. The file name appears in the *File* field.

   The destination folder is displayed in the *Import to* field. To change this folder, choose *Select Folder*, and browse to the required folder, or create a new folder.

   The specified folder, containing the artifacts extracted from the archive, is created in the project.
3. To make the imported folder a proper module in your project, you need to convert it into a module of the matching type. From the folder context menu, choose *Convert To*, and then the type of the target module.

   > **i Note**
   >
   > The conversion process does not check whether the imported folder structure matches the selected module type. The process does not generate the module artifacts according to the selected type.

   The imported module becomes a part of your MTA project, and the module entry is added to the MTA descriptor.

15.3.6.3 Define Dependencies

You can define optional dependencies of a Java module after creating it.

Procedure

1. Modify `pom.xml` to add dependencies as found on the Maven Web site.

2. If your SAP Web IDE server cannot access the Internet, in particular`https://repo1.maven.org/maven2/`, you can set up your own Maven repository manager. Such custom repository setups can be configured by adding a `settings.xml` file to your Java module.
3. To identify outdated dependencies, select your Java module and go to *File* > *Show dependency updates*. Use the *Problems* view to see outdated dependencies.

Related Information

Create a Java Module [page 1142]
Run a Java Module [page 1144]
Inside an MTA Descriptor [page 1079]
http://search.maven.org/
https://maven.apache.org/settings.html
https://maven.apache.org/repository-management.html
https://repo1.maven.org/maven2/

15.3.6.4 Run a Java Module

To create a running application in the runtime system, and resolve all dependencies defined in `mta.yaml`, run each Java module in your project.

Prerequisites

Before running a module, make sure that:

- All the relevant dependencies of the module are defined in the MTA descriptor.
- All the modules and resources on which the module is dependent are implemented and available (HDB modules are built).

> **i Note**
>
> You don't need to build your module explicitly before running, because it will be built automatically during the run process. The result of that build will be available in your workspace in the `target` folder, either as a `.war` or `.jar` file, depending on the type of your Java module.

Procedure

From the module context menu, choose *Run* and select one of the following options:

Option	Description
Run Configurations	Create a new or modify an existing run configuration. In the *Browser Window* tab, you can optionally choose a service path for your module to open in a new browser window. For example, choose */hello* for the Hello World sample servlet generated by the wizard
Run as Java Application	Run the `.war` or `.jar` archive that was last built by Maven.

> **Note**
>
> If you have an MTA extension file with the name `dev.mtaext` that extends the `mta.yaml` file in the MTA project root, its contents will be merged with the `mta.yaml` file when running the module.

15.3.6.5 Test a Java Module

Build and test your Java application with unit tests.

- Add unit tests in the `test` folder structure.
- Execute your tests. In the context menu from your module select *Build and Run Tests*.
- Review the test results.

Each Java module must provide own its JUnit test. The Java module template comes with a sample JUnit test. Use this test to get familiar with the test functionality and the *(test results)* pane. By default, the *(test results)* pane shows *Test Results*, *Stack Trace*, and *Coverage* for the module that is selected in the workspace. Also by default, the code coverage is shown in the editor, which enables you to write targeted unit tests for uncovered code. To disable this function, edit your *(settings)* .

If you do not see coverage results in the test pane, open the `pom.xml` file, and make sure the following code is cotained in the `build` section.

> **Sample Code**

```
<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>0.7.9</version>
    <executions>
        <execution>
            <id>default-prepare-agent</id>
            <goals>
                <goal>prepare-agent</goal>
            </goals>
        </execution>
        <execution>
            <id>jacoco-report</id>
            <phase>test</phase>
            <goals>
                <goal>report</goal>
            </goals>
        </execution>
    </executions>
</plugin>
```

15.3.6.6 Use the Run Console

Progress and status messages, generated during the run process, as well as the log records, are displayed in the Run console.

The console opens automatically in the main browser tab. You can show and hide the console by clicking the *(run console)* icon in the bottom pane's toolbar.

The left pane of the console displays a list of all running modules in the current MTA project along with the corresponding run configurations. Each entry is accompanied by a status icon with a tooltip. Click the run configuration name below a module name to view its most recent run log.

The controls displayed at the top of the console enable you to perform the following tasks:

- If the application URL is visible, click it to access the running application in the browser.
- To view the detailed server log of a run process, click *Logs*.
- To clear the log in the console, click *(clear the log)* .
- To stop running the module, click *(stop)* .
  This causes the module to be removed from the runtime system, so that if you run the module again, the run process starts from the beginning.
- To run the module using a different configuration, select the configuration and choose the *(run)* button in the task bar.

All the log and error messages of a running Java application are immediately displayed in the Run console.

15.3.6.7 Build a Java Module

You can perform a full module build, or any build phase, or even a specific task (plugin goal) by using options from the module context menu.

Option	Description
Build	Build the module by executing the Maven `clean install` command.
Execute Maven Goal...	Execute any build lifecycle phase or even a granular task (plugin goal) by entering a Maven command without the `mvn` prefix in the *Command Line* field, and choosing *Execute*. Note that the input is not validated. If you enter an invalid command, the build will fail, and an error message `Unknown lifecycle phase`... will be displayed in the console.

15.3.6.8 Debug Java Modules

You can debug your Java modules using the tools in the Debugger pane.

If you have enabled debugging in the run configuration, and set breakpoints in your `.java` files, the (Debugger) pane will open as soon as the module is up and running. In this pane, you can perform regular debugging tasks, such as viewing the call stack, examining the variables, stepping in and out of the functions, and more.

Beginning with SAP HANA XS Advanced 2.0 SP02, you don't have to run Java modules explicitly in debug mode. Instead, you can *(attach)* to any running Java module in the (Debugger) pane. Your Java module automatically switches to debug mode.

Breakpoints

Breakpoint Status	Description
Enabled standard breakpoint	Shape outline: blue Shape fill: light blue
Disabled standard breakpoint	Shape outline: blue Shape fill: transparent Breakpoints have been explicitly disabled. Disable breakpoints in the *Breakpoints* section by selecting the corresponding checkbox or deactivate globally by pressing the *(deactivate breakpoints)* button.
Checkmark	Shape outline and fill are not affected. You set the breakpoint successfully on a running debugging session. The checkmark indicates that the corresponding source file has been loaded. As a consequence the application will suspend when it reaches the breakpoint. A breakpoint without a checkmark can mean that there is no debug session running, the source file has not been loaded, or the breakpoint could not be successfully resolved. An unresolved breakpoint may occur when the workspace resource version and the currently executed resource version don't match.

Related Information

Build a Java Module [page 1146]
Run a Java Module [page 1144]
Test a Java Module [page 1145]

15.3.6.9 Use Eclipse to Develop Java Modules

(Optional) You can develop and debug Java modules for your multi-target applications (MTA) in Eclipse, while performing all other development tasks in SAP Web IDE.

In SAP Web IDE

Task	Information
1. Create an MTA project.	Setting Up Application Projects [page 1087]
2. To generate the data objects in the database, develop and build an SAP HANA Database module.	Developing SAP HANA Database (HDB) Modules [page 1090]
3. Create a Java module.	Developing Java Modules [page 1140]
4. Submit the module to a Git repository, or export it to the file system.	Using Source Control (Git) [page 1254]
5. Switch to Eclipse IDE.	

In Eclipse IDE

Task	Information
1. Clone the Java module from the Git repository.	Choose *Window* > *Show View* > *Git Repositories* > *Clone a Git Repository*, and add the clone to the view. Alternatively, you can import the exported module from the file system.
2. Import the Java project into Eclipse.	1. Choose *Window* > *Show View* > *Project Explorer* > *Import* > *Existing Maven Projects*. 2. In *Root Directory*, enter the folder to which you have cloned the project, and click *Finish*.
3. Write Java code, then build and run the Java application.	1. From the project context menu, choose *Run As* > *Maven install*. 2. Choose *Run As* > *Run on Server*.

Task	Information
4. Deploy the application to Cloud Foundry.	1. In the *Select the Server* dropdown list, either choose an existing server, and then *Cloud* > *Cloud Foundry* >, or define the server manually. 2. In the *Add and Remove* dialog box, add the required resources to the *Configured* list, and click *Finish*. 3. In the *Application Details* dialog box, make sure that the details of your application are correct, and click *Next*. 4. In the *Launch Deployment* dialog box, enter the deployment details, and click *Next*. 5. Bind a service to the application. In the *Service selection* dialog box, select the `hdi container` of the HDB module that you previously built in SAP Web IDE. The name has the following pattern: `<project name>-hdihdb-<userid><workspace>`. Click *Next*.

5. Add environment variable PATH that points to the WAR file.	Add the following parameters to the `manifest.yml` file:
	• `path` that points to the WAR file.
	• In the `env` section, `JBP_CONFIG_RESOURCE_CONFIGURATION`
	For example:
	`...`
	`path: /Users/<UserID>/workspace/mtad4/java4/target/java4-0.0.1-SNAPSHOT.war`
	`  env:`
	`  JBP_CONFIG_RESOURCE_CONFIGURATION: '[tomcat/webapps/ROOT/META-INF/context.xml:`
	`  {"service_name_for_DefaultDB" : "<name of the hdi-container>"}]'`

6. Submit the Java project to the same Git repository.	
7. Switch to SAP Web IDE.	

In SAP Web IDE

Task	Information
1. Clone the Java module from the Git repository, and import it to your MTA project.	Create a Java Module [page 1142]
2. Develop an HTML5 module as required.	Developing HTML5 Modules [page 1150]
3. Run and test the application, then deploy it to XS Advanced.	Packaging and Deploying Applications [page 1234]

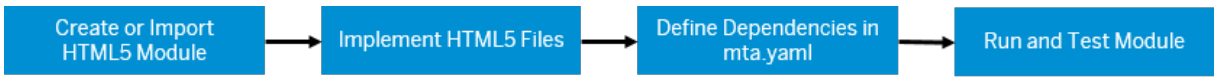
15.3.7 Developing HTML5 Modules

An HTML5 module is a collection of related HTML5 files that implement the user interface of your application.

SAP Web IDE supports the following HTML5 module types:

- Basic HTML5 module - a very basic UI app.
- SAPUI5 HTML5 module - an SAPUI5-based UI app with an optional view but without a predefined layout.
- SAP Fiori master-detail module - an SAPUI5-based UI app with the split-screen SAP Fiori layout.
- List Report module - based on the list report and object page SAP Fiori elements.

To develop HTML5 modules, perform the following steps:



- Create an HTML5 Module [page 1151]
- Implementing HTML5 Files [page 1151]
- Inside an MTA Descriptor [page 1079]
- Run and Test HTML5 Modules [page 1220]

Module Folder Structure

The following figure depicts a sample basic HTML5 module folder structure alongside the corresponding entry in the `mta.yaml`.

Folder	Description
<module name>	Contains the `xs-app.json` (application router configuration), and `package.json` (application router details and dependencies) files.
resources	Contains the HTML5 resource files, including the default *index.html* file.

> ⚠ Caution
>
> Do not change the version of `approuter` in the `package.json` of the module.

Implementing HTML5 Files

In the `\resources` subfolder, create and implement the required HTML5 files. You can design the XML views using the SAP Web IDE layout editor.

For information about developing HTML5 apps with SAPUI5, see SAPUI5: UI Development Toolkit for HTML5.

Related Information

Layout Editor [page 1154]
Developing Multitarget Applications [page 1078]

15.3.7.1 Create an HTML5 Module

You can create new HTML5 modules.

Procedure

1. From the project context menu, choose *New*.
2. Depending on your application needs, choose one of the HTML5 module templates.
3. Enter the module name, and set the relevant module properties.

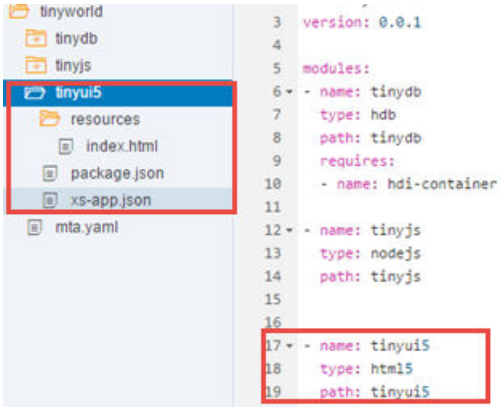
   SAPUI5 HTML5 Module

Property	Description
Namespace	Optional. Enter a value if you want to override the default, which is the module path in the project.
View Type	Select one of the available view types: None, XML, JSON, JavaScript, HTML.
View Name	Name of the view.

   SAP Fiori Master-Detail Module

Property	Description
Sources	Data source for the module. Select an existing OData service from one of the following: • *Current project*: Select an OData service from the list of available services exposed by the Java and Node.js modules in the current project. If the module is not running, run it. Note that the exposed service must allow anonymous access. • *Workspace*: select a `metadata.xml` file from one of the projects in your workspace. • *File system*: select a `metadata.xml` file from the file system.
Other properties: *Application Settings* *SAP Fiori Settings*	Click the image on the right to enlarge it, and see the detailed explanations of the properties.

4. In the *Data Connection* wizard step, define a connection to the appropriate data source. For more information, see Select a Data Source for an HTML5 Module [page 1153].
A new HTML5 module with the specified name is created in your project, and a corresponding section is added to the MTA descriptor (`mta.yaml`). For example:



> **Note**
>
> The default `index.html` file of the new module implements a basic Hello World application.

Related Information

Developing HTML5 Modules [page 1150]
Import an HTML5 Module [page 1153]

15.3.7.2 Import an HTML5 Module

You can import archived HTML5 modules.

Prerequisites

The `.zip` module archive that you want to import, which was exported from another MTA project, is available in the file system.

Procedure

1. From the root folder of the project, choose *File* > *Import* > *File or Project*.
2. Click *Browse* to locate and select your archive, and choose *Open*. The file name appears in the *File* field.

   The destination folder is displayed in the *Import to* field. To change this folder, choose *Select Folder*, and browse to the required folder, or create a new folder.

   The specified folder, containing the artifacts extracted from the archive, is created in the project.
3. To make the imported folder a proper module in your project, you need to convert it into a module of the matching type. From the folder context menu, choose *Convert To*, and then the type of the target module.

   > **i Note**
   >
   > The conversion process does not check whether the imported folder structure matches the selected module type. The process does not generate the module artifacts according to the selected type.

   The imported module becomes a part of your MTA project, and the module entry is added to the MTA descriptor.

15.3.7.3 Select a Data Source for an HTML5 Module

Select a data source to access it from your application.

The following table describes how to define connections to the available data source types.

Data Source Type	Instructions
Current Project	Browse your project to locate and select a Java or Node.js module that exposes an OData service.
File System	Browse your file system to locate and select the required service `metadata.xml` document.

Data Source Type	Instructions
Workspace	Browse your workspace to locate and select the required service `metadata.xml` document.

When you select a remote service, you are prompted to provide credentials for the requested system.

15.3.7.4 Layout Editor

Display the content of an XML view in the layout editor to see it in a way that closely corresponds to how it will appear in your finished application.

> **i** Note
>
> The layout editor is not supported in the Safari browser.

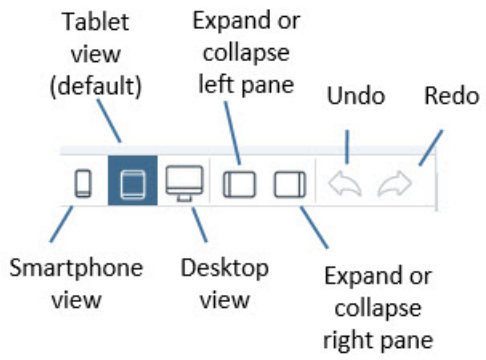
Layout Editor Landscape

The layout editor is composed of a canvas, a pane on the left that includes the *Controls* and *Outline* tabs, and a pane on the right that includes the *Events* and *Properties* panes.

Toolbar

The buttons on the layout editor toolbar allow you to:

- Change the device format of the canvas to smartphone, tablet, or desktop view.
- Change the application's SAPUI5 version.
- Expand and collapse the panes to the right and left of the canvas.
  - The pane on the left side includes the *Controls* and *Outline* tabs.
  - The pane on the right side includes the *Properties* and *Events* panes.
- Undo and redo actions.

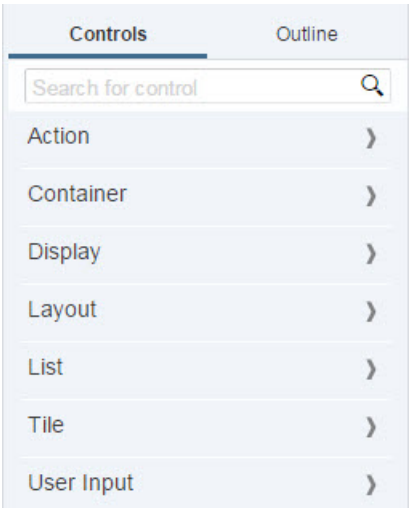


Controls Tab

You can expand or collapse each section by clicking the arrow on each section header. You can also search for controls by entering the control name in the search field at the top of the *Controls* tab. The relevant sections expand to display the controls that match the search criteria.

> **i Note**
>
> Make sure to delete the search criteria if you want to expand other sections.

You can drag and drop controls from the *Controls* tab onto the canvas. For more information, see Add Controls from the Controls Tab [page 1163].



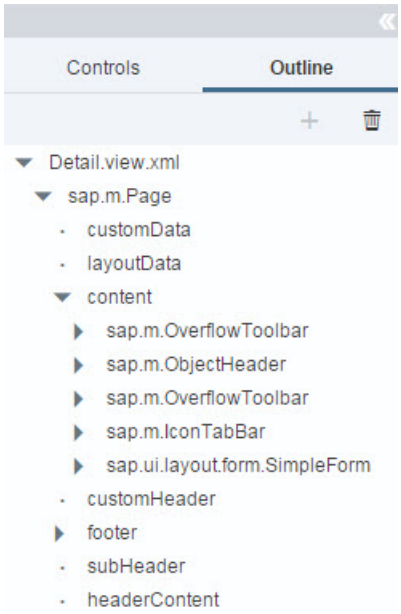
You can find the list of available controls in SAPUI5 Controls Supported in the Layout Editor [page 1171].

Outline Tab

Controls that are selected on the *Outline* tab are automatically selected on the canvas and vice versa.

You can use the *Outline* tab to see the hierarchy of controls on the canvas. In addition, you can add and remove controls from the canvas using the *Outline* tab.

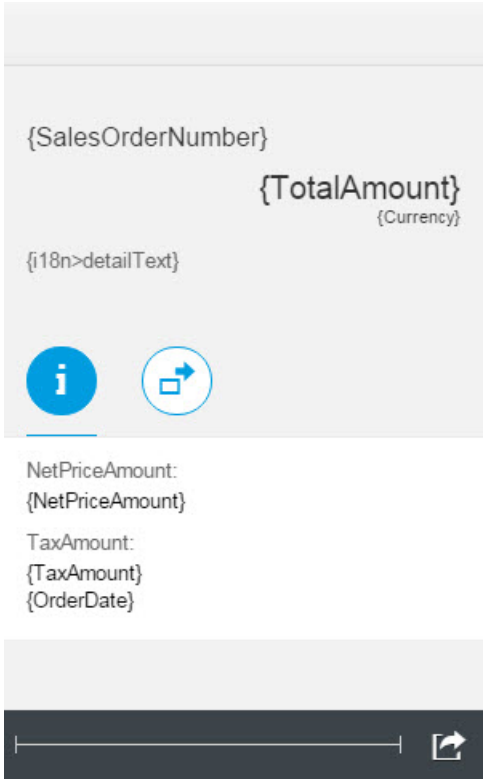
For more information, see Add Controls from the Outline Tab [page 1160].



Canvas

The canvas in the middle of the layout editor area provides a graphical display of the selected XML view.

Click a control on the canvas to select it. Click again to select its parent control. You can keep clicking until you reach the highest control in the hierarchy and then the focus will return to the original control. Click outside the canvas to undo the selection.

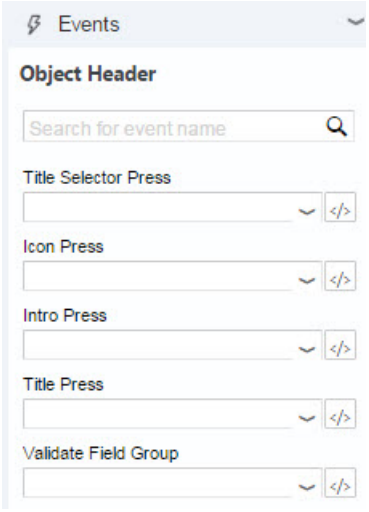


Events and Properties Pane

On the right side of the canvas is a pane that displays the following panes:

Events Pane

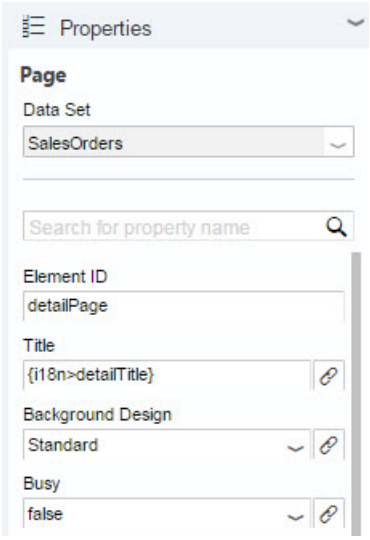
The *Events* pane allows you to select an existing event handler from the controller for an event of the selected control. The </> icon next to each event opens the code editor to display the relevant controller in the XML code.



Properties Pane

The *Properties* pane shows the properties of the control that is currently selected in the canvas and allows you to modify its property values. The most commonly used properties for each control are displayed at the top of the list. The icon next to each property opens the *Data Binding* dialog box.

For more information, see Binding Data [page 1165], Bind Data to a Simple Control [page 1168], and Bind Data to an Aggregate-Type Control [page 1168].



> **i Note**
>
> Deprecated properties or aggregations are marked with the label *deprecated* (also in the *Outline* tab). For more information, see *SAP Library for User Interface Add-On 1.0 for SAP NetWeaver* on SAP Help Portal at http://help.sap.com/nw-uiaddon. Under *Application Help*, open *SAP Library*, and search for *deprecation*.

Working with the Layout Editor [page 1159]
: An overview of the steps required to edit a project using the layout editor.

SAPUI5 Controls Supported in the Layout Editor [page 1171]
: Provides a list of SAPUI5 controls that are supported in the layout editor.

Try It: Build an Application with the Layout Editor [page 1189]
: Get an overview of the features that are available with the layout editor by following this tutorial for building an application.

Related Information

Working with the Layout Editor [page 1159]
SAPUI5 Controls Supported in the Layout Editor [page 1171]

15.3.7.4.1 Working with the Layout Editor

An overview of the steps required to edit a project using the layout editor.

Prerequisites

- In your MTA project, you have an HTML5 module with an XML view.

Procedure

1. In the `view` subfolder of the module folder, choose the XML file.
2. From the context menu of the XML view, choose *Open with* > *Layout Editor*.
3. Edit the views of your application as follows:
   - Add controls to your view using the drag and drop functionality.
   - Delete controls from your view.
   - Rearrange controls in your view using the drag and drop functionality.
   - Use the keyboard to navigate within the canvas. Double-click on a control to move the selection to its parent.
   - Bind controls in the layout editor to elements from the OData service.
   - Extract a control to a fragment using the context menu.

Add Controls from the Outline Tab [page 1160]
You can add controls to the canvas from the *Outline* tab.

Delete Controls from the Outline Tab [page 1160]
You can remove controls from the *Outline* tab.

Create a New Function [page 1161]
You can create a new function for a controller from the *Events* pane.

Create a Navigation Between Views [page 1162]
You can create a navigation between views using the layout editor.

Add Controls from the Controls Tab [page 1163]
Add controls to the canvas by using drag and drop functionality.

Keyboard Support [page 1164]
Use the keyboard to move selected controls or navigate within the view that you opened with the layout editor.

Layout Editor Binding Capabilities [page 1165]
In the layout editor, you can bind properties of controls or control aggregations to data fields, `i18n` models, and label annotations.

15.3.7.4.1.1 Add Controls from the Outline Tab

You can add controls to the canvas from the *Outline* tab.

Procedure

1. On the *Outline* tab, select a control to which you want to add another control.
2. At the top of the *Outline* tab, click the *Add* + button, then in the popup menu, select the control you want to add.

   The control is added on the *Outline* tab and appears on the canvas.

   > **i** Note
   >
   > The information bar at the top of the canvas shows you where you are about to drop the control.

Task overview: Working with the Layout Editor [page 1159]

Related Information

Delete Controls from the Outline Tab [page 1160]
Create a New Function [page 1161]
Create a Navigation Between Views [page 1162]
Add Controls from the Controls Tab [page 1163]
Keyboard Support [page 1164]
Layout Editor Binding Capabilities [page 1165]
Delete Controls from the Outline Tab [page 1160]
SAPUI5 Controls Supported in the Layout Editor [page 1171]

15.3.7.4.1.2 Delete Controls from the Outline Tab

You can remove controls from the *Outline* tab.

Procedure

1. On the *Outline* tab, select a control that you want to delete.
2. At the top of the *Outline* tab, click the *Delete* button.

The control is removed from the view.

Task overview: Working with the Layout Editor [page 1159]

Related Information

Add Controls from the Outline Tab [page 1160]
Create a New Function [page 1161]
Create a Navigation Between Views [page 1162]
Add Controls from the Controls Tab [page 1163]
Keyboard Support [page 1164]
Layout Editor Binding Capabilities [page 1165]
Add Controls from the Outline Tab [page 1160]
SAPUI5 Controls Supported in the Layout Editor [page 1171]

15.3.7.4.1.3 Create a New Function

You can create a new function for a controller from the *Events* pane.

Procedure

1. On the canvas, select a control.
2. In the *Events* pane, open the dropdown list under the event for whose controller you want to create a function.
3. In the dropdown list, select *New Function*.
4. In the *New Function* dialog box, enter a function name and click *OK*.

   > **i** Note
   >
   > The function name you enter must be a valid JavaScript function name.

   After entering the name for your new function, the code is created by the layout editor in the code editor. To go directly to that code, click the *Go to code* </> icon next to the new function name in the *Events* pane.

Task overview: Working with the Layout Editor [page 1159]

Related Information

Add Controls from the Outline Tab [page 1160]
Delete Controls from the Outline Tab [page 1160]
Create a Navigation Between Views [page 1162]
Add Controls from the Controls Tab [page 1163]
Keyboard Support [page 1164]
Layout Editor Binding Capabilities [page 1165]
Layout Editor [page 1154]

15.3.7.4.1.4 Create a Navigation Between Views

You can create a navigation between views using the layout editor.

Prerequisites

- You have enabled the *Storyboard* perspective in the *Preferences* perspective, on the *Extensions* page.
- You have created a freestyle project.

Procedure

1. In the *Development* perspective, in your Workspace, right-click the view to which you want to add a navigation, then in the context menu, choose *Open With* > *Layout Editor*.

   The view opens in the layout editor.
2. From the *Controls* pane, drag a control and drop it onto the view canvas.
3. Select the control and in the *Events* pane, in the *Press* field, click the Open Menu button and select *Navigate To*.
4. In the *Navigate To* dialog box, in the *View* dropdown list, select the view to which you want to navigate, choose *OK*, then choose *Save*.

   In the *Storyboard* perspective, the navigation now appears as an arrow between the views you configured.

Task overview: Working with the Layout Editor [page 1159]

Related Information

Add Controls from the Outline Tab [page 1160]

Delete Controls from the Outline Tab [page 1160]
Create a New Function [page 1161]
Add Controls from the Controls Tab [page 1163]
Keyboard Support [page 1164]
Layout Editor Binding Capabilities [page 1165]

15.3.7.4.1.5 Add Controls from the Controls Tab

Add controls to the canvas by using drag and drop functionality.

Select the control that you want to add to the canvas from the *Controls* tab on the left side of the layout editor area and drag it to the drop target on the canvas. A tooltip displays the drop targets as you drag the control around the canvas.

Example

If you want to add an *HBox* container with a *Button* control to your view, do the following:

1. Choose the *Controls* tab.
2. Open the *Container* section by clicking the arrow to the right of the section title.
3. Select the *HBox* control and drag it to the canvas. Drop it at the position where you want it to appear.
4. Open the *Action* section by clicking the arrow to the right of the section title.
5. Select the *Button* control and drag it to the canvas.
6. Drop the *Button* control onto the *HBox* container.

Parent topic: Working with the Layout Editor [page 1159]

Related Information

Add Controls from the Outline Tab [page 1160]
Delete Controls from the Outline Tab [page 1160]
Create a New Function [page 1161]
Create a Navigation Between Views [page 1162]
Keyboard Support [page 1164]
Layout Editor Binding Capabilities [page 1165]

15.3.7.4.1.6 Keyboard Support

Use the keyboard to move selected controls or navigate within the view that you opened with the layout editor.

Selecting Controls

If you selected a control in the canvas of the layout editor, you can move the selection using the arrow keys:

- `UP ARROW`: moves the selection to the parent of the selected control
- `DOWN ARROW`: moves the selection to the child of the selected control
- `LEFT ARROW`: moves the selection to the control that is up/to the right of the selected control (within the same aggregation)
- `RIGHT ARROW`: moves the selection to the control that is down/to the left of the selected control (within the same aggregation)
- `CTRL`+ click: moves the selection to the parent of the selected control

Moving Selected Controls

If you have selected a control in the canvas of the layout editor, you can change its position within the aggregation:

- `SHIFT`+`LEFT ARROW`: moves the control up/to the right
- `SHIFT`+`RIGHT ARROW`: moves the control down/to the left

Changing the Drop Target

When you drag and drop a control from the palette to the canvas or from one position within the canvas to another, you can use the keyboard to define the drop position of the dragged control:

- Use `SHIFT` or `ALT` to define the drop position of the dragged control within an aggregation:
  - `SHIFT`: moves the drop position up/to the right
  - `ALT`: moves the drop position down/to the left

Parent topic: Working with the Layout Editor [page 1159]

Related Information

Add Controls from the Outline Tab [page 1160]
Delete Controls from the Outline Tab [page 1160]

Create a New Function [page 1161]
Create a Navigation Between Views [page 1162]
Add Controls from the Controls Tab [page 1163]
Layout Editor Binding Capabilities [page 1165]

15.3.7.4.1.7 Layout Editor Binding Capabilities

In the layout editor, you can bind properties of controls or control aggregations to data fields, `i18n` models, and label annotations.

Parent topic: Working with the Layout Editor [page 1159]

Related Information

Add Controls from the Outline Tab [page 1160]
Delete Controls from the Outline Tab [page 1160]
Create a New Function [page 1161]
Create a Navigation Between Views [page 1162]
Add Controls from the Controls Tab [page 1163]
Keyboard Support [page 1164]

15.3.7.4.1.7.1 Binding Data

In the layout editor, you can bind properties of controls or control aggregations to an artifact in the OData service.

Prerequisites

You have defined a data set for the view that you are working on, by doing one of the following:

- If you are opening a view that has no data set defined for it, the *Data Binding* dialog box opens, where you can define a data set.
- Select the view, and define the data set from the dropdown list.

> ⚠ Caution
>
> If you change the data set that is defined for the current view, the existing data bindings might become invalid.

Overview

The following types of bindings are possible:

- Properties of controls
- Aggregations of controls

> **i Note**
>
> To bind properties of models that are not OData models, you must work from the source code files and not from the layout editor. Alternatively, if you do not want to work with the source files in the XML editor, you can enter free text for properties in the *Properties* pane.

> **i Note**
>
> If your application does not consume an OData service, you can add the *OData Service* component to it.

Define Entity Set [page 1166]
Before you can bind data to a control, you need to define an entity set.

Bind Data to a Simple Control [page 1168]
You can bind data to a simple control.

Bind Data to an Aggregate-Type Control [page 1168]
You can bind data to an aggregate-type control, which creates a template.

Bind to the i18n Model [page 1169]
You can bind a control property to the i18n model or create a new i18n entry.

Bind to a Label Annotation [page 1170]
You can bind a control property to a label annotation.

Related Information

SAPUI5 API Reference

15.3.7.4.1.7.1.1 Define Entity Set

Before you can bind data to a control, you need to define an entity set.

Prerequisites

- You have created a project in the layout editor. For more information, see Working with the Layout Editor [page 1159].
- You have dragged a control from the *Controls* pane to the canvas. For more information, see Add Controls from the Controls Tab [page 1163].

Context

Procedure

1. In the canvas, select the control for which you want to define an entity set.
2. In the *Properties* pane, click the icon.

   The *Select Entity Set* dialog box opens.
3. Select and configure one of the entity sets for your control.

   > **i** Note
   >
   > Most users should use either of the first two entity sets listed below.

   - *Use ancestor's entity set.*
   - *Define entity set and set the selected control as template.*
   - *Define dummy entity set for the selected control.*

     > **!** Restriction
     >
     > The "dummy" entity set is only for advanced users.
4. Choose OK.

Task overview: Binding Data [page 1165]

Related Information

Bind Data to a Simple Control [page 1168]
Bind Data to an Aggregate-Type Control [page 1168]
Bind to the i18n Model [page 1169]
Bind to a Label Annotation [page 1170]

15.3.7.4.1.7.1.2 Bind Data to a Simple Control

You can bind data to a simple control.

Procedure

1. On the canvas, select the desired control for which you want to define data binding.
2. In the *Properties* pane to the right of the canvas, do one of the following:
   - To the right of the property to which you want to bind data, click the *Binding* button, and then in the *Data Binding* dialog box:
     1. In the *Data Fields* list, double-click one or more data fields that you want to add to the expression. The data fields are automatically concatenated to the string in the *Expression* box.
     2. Click *OK* or first manually edit the expression string and then click *OK*.
   - In the *Properties* pane, for a field or dropdown list, manually enter the required expression within curly brackets {...} according to the data set that you selected.

Task overview: Binding Data [page 1165]

Related Information

Define Entity Set [page 1166]
Bind Data to an Aggregate-Type Control [page 1168]
Bind to the i18n Model [page 1169]
Bind to a Label Annotation [page 1170]

15.3.7.4.1.7.1.3 Bind Data to an Aggregate-Type Control

You can bind data to an aggregate-type control, which creates a template.

Prerequisites

- You have defined an entity set. For more information, see Define Entity Set.

Procedure

1. In the canvas or on the *Outline* tab to the left of the canvas, choose an aggregate-type control that you want to turn into a template, such as a *List Item* control.
2. In the properties pane to the right of the canvas, under the *Data Set* dropdown list, check the *Set as template* checkbox.

   This control becomes a template.
3. In the *Confirmation Needed* dialog box, confirm the removal of any existing controls on the same level by clicking *OK*.

Results

The template item is now marked *Template* in the *Outline* tab.

Task overview: Binding Data [page 1165]

Related Information

Define Entity Set [page 1166]
Bind Data to a Simple Control [page 1168]
Bind to the i18n Model [page 1169]
Bind to a Label Annotation [page 1170]
Aggregation Binding

15.3.7.4.1.7.1.4 Bind to the i18n Model

You can bind a control property to the i18n model or create a new i18n entry.

Procedure

1. On the canvas, select the desired control for which you want to define i18n model binding.
2. In the *Properties* pane to the right of the canvas, do one of the following:

   - To the right of the property to which you want to bind data, click the *Binding* button, and then in the dialog box that appears:
     1. From the drop down list, select *i18n*, double-click one or more entries that you want to add to the expression.

The entries are automatically concatenated to the string in the *Expression* box.

2. Click *OK* or first manually edit the expression string and then click *OK*.

- In the *Properties* pane, for a field or dropdown list, manually enter the required expression within curly brackets {...}.
- Click + to add a new i18n entry.

Task overview: Binding Data [page 1165]

Related Information

Define Entity Set [page 1166]
Bind Data to a Simple Control [page 1168]
Bind Data to an Aggregate-Type Control [page 1168]
Bind to a Label Annotation [page 1170]

15.3.7.4.1.7.1.5 Bind to a Label Annotation

You can bind a control property to a label annotation.

Context

You can bind a control property to a label annotation that resides in the OData metadata file. Other annotation files are not supported.

Procedure

1. On the canvas, select the desired control for which you want to define a label annotation binding.
2. In the *Properties* pane to the right of the canvas, do one of the following:

   - To the right of the property to which you want to bind data, click the *Binding* button, and then in the dialog box that appears:
     1. From the drop down list, select *Labels*, double-click one or more annotation that you want to add to the expression.
        The annotations are automatically concatenated to the string in the *Expression* box.
     2. Click *OK* or first manually edit the expression string and then click *OK*.
   - In the *Properties* pane, for a field or dropdown list, manually enter the required expression within curly brackets {...}.

Task overview: Binding Data [page 1165]

Related Information

Define Entity Set [page 1166]
Bind Data to a Simple Control [page 1168]
Bind Data to an Aggregate-Type Control [page 1168]
Bind to the i18n Model [page 1169]

15.3.7.4.2 SAPUI5 Controls Supported in the Layout Editor

Provides a list of SAPUI5 controls that are supported in the layout editor.

Controls Tab

The SAPUI5 controls listed below can be dragged and dropped from the *Controls* tab onto the canvas.

> **i Note**
>
> The controls on the *Controls* tab are also available from the *Outline* tab. For more information, see Add Controls from the Outline Tab [page 1160].

> **i Note**
>
> For more information about SAPUI5 controls, see UI development toolkit for HTML5 - Demo Kit.

SAPUI5 Controls Available on the Controls Tab

SAPUI5 Control Name	Description
Action List Items sap.m.ActionListItem	Button that is used to fire actions when pressed.
Action Select sap.m.ActionSelect	Provides a list of predefined items that allows end users to choose options and additionally trigger some actions.
Add Bookmark Button sap.ushell.ui.footerbar.AddBookmarkButton	Button that is displayed in the application footer. Clicking the button opens a dialog box that allows the user to save the app state, so that the app can be launched in this state directly from the launchpad.

SAPUI5 Control Name	Description
Analytic Map sap.ui.vbm.AnalyticMap	Renders a map based on a GeoJSON source.
App sap.m.App	The root element of an SAPUI5 mobile application. It inherits from the `NavContainer` control and thus provides its navigation capabilities. `App` provides certain header tags to the HTML page that are relevant for mobile apps.
Bar sap.m.Bar	Centers a control like a title while having other controls on its left and right.
Breadcrumbs sap.m.Breadcrumbs	Enables users to navigate between items by providing a list of links to previous steps in the user's navigation path. The last three steps can be accessed as links directly, while the remaining links prior to them are available in a drop-down menu.
Busy Indicator sap.ui.core.BusyIndicator	Provides methods to show or hide a waiting animation that covers the whole page and blocks user interaction.
Button sap.m.Button	Allows users to trigger actions.
Calendar sap.ui.unified.Calendar	Basic calendar that is used for `DatePickers`.
Calendar Legend sap.ui.unified.CalendarLegend	Legend for the `Calendar` control. Displays special date colors with their corresponding description.
CalendarLegendItem sap.ui.unified.CalendarLegendItem	Item to be displayed in a CalendarLegend.
Carousel sap.m.Carousel	Navigates through a list of controls by swiping right or left.
Check Box sap.m.CheckBox	Allows the user to select one or multiple items from a list.
Component Container sap.ui.core.ComponentContainer	Container that embeds a `UIComponent` in a control tree.
Control Configuration sap.ui.comp.smartfilterbar.ControlConfiguration	Can be used to add additional configuration for filter fields in the SmartFilterBar control, in order to overwrite the default settings from the OData metadata. For instance, it is possible to change the label, index or control type of a filter field.

SAPUI5 Control Name	Description
Column sap.m.Column	Allows definition of column-specific properties that are applied when rendering a `List` control.
Column List Item sap.m.ColumnListItem	Used with cell aggregation to create rows for the `sap.m.Table` control.
Combo Box sap.m.ComboBox	Combines a dropdown list with items and a text input field with a button allowing the user to either type a value directly or choose from a list of predefined items.
Custom Data sap.ui.core.CustomData	Contains a single key/value pair of custom data attached to an Element.
Custom List Item sap.m.CustomListItem	With content aggregation, can be used to customize standard list items that are not provided by SAPUI5. `ListItem` type is applied to `CustomListItem` as well. **Note** Content aggregation allows any control. Complex responsive layout controls (such as `Table` and `Form`) should not be aggregated as content.
Custom Tile sap.m.CustomTile	Displays application-specific content in the `Tile` control.
Date Picker sap.m.DatePicker	Date input control with a calendar used as a date picker.
Detail Page sap.m.semantic.DetailPage	An `sap.m.semantic.ShareMenuPage` control that supports certain semantic buttons that have default semantic-specific properties and are eligible for content aggregation.
Display List Item sap.m.DisplayListItem	Used to represent a label and a value.
DraftIndicator sap.m.DraftIndicator	A draft indicator is `sap.m.Label`.
Feed Input sap.m.FeedInput	Allows the user to enter text for a new feed entry and then post it.
Feed List Item sap.m.FeedListItem	Provides a set of properties for text, sender information, and time stamp.

SAPUI5 Control Name	Description
Filter Group Item sap.ui.comp.filterbar.FilterGroupItem	Represents a filter belonging to a group other than basic.
Filter Item sap.ui.comp.filterbar.FilterItem	Represents a filter belonging to the basic group.
Flex Item Data sap.m.FlexItemData	Holds layout data for a FlexBox / HBox / VBox.
Flex Box sap.m.FlexBox	Builds the container for a flexible box layout.
Form sap.ui.layout.form.Form	Structured into `FormContainer` controls, each of which consists of `FormElement` controls.
Form Container sap.ui.layout.form.FormContainer	Group inside a `Form`.
Form Element sap.ui.layout.form.FormElement	Row in a `FormContainer` control.
Fullscreen Page sap.m.semantic.FullscreenPage	An `sap.m.semantic.ShareMenuPage` control that supports certain semantic buttons that have default semantic-specific properties and are eligible for content aggregation.
Generic Tile sap.m.GenericTile	Displays header, subheader, and a customizable main area in a tile format.
Geo Map sap.ui.vbm.GeoMap	A map control that allows the user to position multiple visual objects on top of a map.
Grid sap.ui.layout.Grid	Layout that positions its child controls in a 12-column flow layout.
Grid Container Data sap.ui.layout.form.GridContainerData	The `GridLayout`-specific layout data for `FormContainers`.
Grid Data sap.ui.layout.GridData	Grid layout data.

SAPUI5 Control Name	Description
Grid Element Data sap.ui.layout.form.GridElement-Data	The `GridLayout`-specific layout data for `FormElement` fields.
Group sap.ui.comp.smartform.Group	Groups are used to group `group` elements.
Group Configuration sap.ui.comp.smartfilterbar.Group-Configuration	Can be used to add additional configurations for groups in the SmartFilterBar. A group in the SmartFilterBar is a group of filter fields in the advanced search.
GroupElement sap.ui.comp.smartform.GroupElement	A combination of one label and different controls associated to this label.
Group Header List Item sap.m.GroupHeaderListItem	Used to display the title of a group and act as a separator between groups in `sap.m.List` and `sap.m.Table`.
HBox sap.m.HBox	Builds the container for a horizontal flexible box layout.
Horizontal Layout sap.ui.layout.HorizontalLayout	Provides support for horizontal alignment of controls.
Icon sap.ui.core.Icon	Uses an embedded font instead of a pixel image.
Icon Tab Bar sap.m.IconTabBar	Represents a collection of tabs with associated content.
Icon Tab Filter sap.m.IconTabFilter	Represents a selectable item inside an `Icon Tab Bar` control.
Icon Tab Header sap.m.IconTabHeader	Displays a number of `Icon Tab Filter` and `Icon Tab Separator` controls.
Icon Tab Separator sap.m.IconTabSeparator	Icon used to separate two `Icon Tab Filter` controls.
Image sap.m.Image	Wrapper around the `IMG` tag.

SAPUI5 Control Name	Description
Input sap.m.Input	Allows users to input data.
Input List Item sap.m.InputListItem	List item used for a label and an input field.
Invisible Text sap.ui.core.InvisibleText	Used to bring hidden texts to the UI for screen reader support.
Item sap.ui.core.Item	Control base type.
Label sap.m.Label	Used in SAPUI5 mobile applications to provide label text for other controls.
Layout Data sap.ui.core.LayoutData	A layout data base type.
Link sap.m.Link	Used to trigger actions or to navigate to other applications or web pages.
List sap.m.List	Provides a container for all types of list items.
List Item sap.ui.core.ListItem	Used in lists or list-like controls, such as `DropdownBox`.
Master Page sap.m.semantic.MasterPage	An `sap.m.semantic.SemanticPage` control that supports certain semantic buttons that have default semantic-specific properties and are eligible for content aggregation.
Message Strip sap.m.MessageStrip	Allows the embedding of application-related messages in the application.
Nav Container sap.m.NavContainer	Handles hierarchical navigation between `Page` controls or other fullscreen controls.
Object Attribute sap.m.ObjectAttribute	Displays a text field that can be normal or active.
Object Header sap.m.ObjectHeader	Allows the user to easily identify a special object.

SAPUI5 Control Name	Description
Object Identifier sap.m.ObjectIdentifier	Display control that allows the user to easily identify a specific object.
Object List Item sap.m.ObjectListItem	Display control that provides summary information about an object as an item in a list.
Object Number sap.m.ObjectNumber	Displays number and number unit properties for an object.
Object Status sap.m.ObjectStatus	Status information that can be either text with a value state, or an icon.
Overflow Toolbar sap.m.OverflowToolbar	Container based on `sap.m.Toolbar` that provides overflow when its content does not fit in the visible area.
Overflow Toolbar Button sap.m.OverflowToolbarButton	Represents an sap.m.Button that shows its text only when in the overflow area of an sap.m.OverflowToolbar.
Page sap.m.Page	Basic container for a mobile application screen.
Paging Button sap.m.PagingButton	Allows users to navigate between items and entities.
Pane Container sap.ui.layout.PaneContainer	An abstraction of `Splitter`. Could be used as an aggregation of `ResponsiveSplitter` or other `PaneContainers`.
Panel sap.m.Panel	Container for controls that has a header and content.
Progress Indicator sap.m.ProgressIndicator	Shows the progress of a process in a graphical way.
Pull To Refresh sap.m.PullToRefresh	Triggers the refresh event.
Radio Button sap.m.RadioButton	Control similar to `CheckBox`, but it allows the user to choose only one of a predefined set of options.
Radio Button Group sap.m.RadioButtonGroup	Used as a wrapper for a group of `sap.m.RadioButton` controls, which then can be used as a single UI element.

SAPUI5 Control Name	Description
Rating Indicator sap.m.RatingIndicator	Used to rate content.
Responsive Flow Layout Data sap.ui.layout.ResponsiveFlowLayoutData	This is a `LayoutData` element that can be added to a control if this control is used within a `ResponsiveFlowLayout`.
Search Field sap.m.SearchField	Allows users to input a search string.
Segmented Button sap.m.SegmentedButton	Horizontal control made of multiple buttons, which can display a title or an image.
Segmented Button Item sap.m.SegmentedButtonItem	Used for creating buttons for the `sap.m.SegmentedButton`. It is derived from the `sap.ui.core.Item`.
Select sap.m.Select	Provides a list of items that allows the user to select an item.
Select List sap.m.SelectList	Displays a list of items that allows the user to select an item.
Semantic Page sap.f.semantic.SemanticPage	An enhanced `sap.f.DynamicPage`, that contains controls with semantic-specific meaning.
Semantic Page sap.m.semantic.SemanticPage	An enhanced `sap.m.Page` control that can contain controls with semantic meaning. Content specified in `sap.m.semantic.SemanticPage` semantic control aggregations are automatically positioned in dedicated sections of the footer or the header of the page, depending on the control's semantics.
Share Menu Page sap.m.semantic.ShareMenuPage	An `sap.m.semantic.SemanticPage` control that supports a `Share` menu in the footer.
Shell sap.m.Shell	Can be used as root element of applications. It can contain an `App` or a `SplitApp` control. The Shell provides some overarching functionality for the overall application and takes care of visual adaptation, such as a frame around the App, on desktop browser platforms.
Simple Form sap.ui.layout.form.SimpleForm	Provides an API for creating simple forms. Inside a `SimpleForm` control, a `Form` control is created along with its `FormContainers` control and `FormElements` control, but the complexity in the API is removed.
Slider sap.m.Slider	User interface control that allows the user to adjust values within a specified numerical range.

SAPUI5 Control Name	Description
Smart Chart sap.ui.comp.smartchart.SmartChart	Creates a chart based on OData metadata and the configuration specified. The `entitySet` property must be specified to use the control. This property is used to fetch fields from OData metadata, from which the chart UI will be generated. It can also be used to fetch the actual chart data. Based on the `chartType` property, this control will render the corresponding chart. **i Note** Most of the attributes are not dynamic and cannot be changed once the control has been initialized.
Smart Field sap.ui.comp.smartfield.SmartField	A wrapper for other controls. It interprets OData metadata to determine the control that has to be instantiated. The OData entity is derived from the control's binding context. The OData entity's property that is changed or displayed with the control is derived from the control's value property.
Smart Filter Bar sap.ui.comp.smartfilterbar.SmartFilterBar	The `SmartFilterBar` control uses the OData metadata of an entity in order to create a FilterBar. Whether a field is visible on the `FilterBar`, supports type-ahead and value help, for example, is automatically determined. When you use control configurations and group configurations it is possible to configure the `FilterBar` and adapt it according to your needs.
Smart Form sap.ui.comp.smartform.SmartForm	Renders a form (`sap.ui.layout.form.Form`). When used with the SmartField control the label is taken from the metadata annotation sap:label if not specified directly.
Smart Link sap.ui.comp.navpopover.SmartLink	The `SmartLink` control uses a semantic object to display `NavigationPopover` for further navigation steps. **i Note** Navigation targets are determined using `CrossApplicationNavigation` of the unified shell service.
Smart Table sap.ui.comp.smarttable.SmartTable	Creates a table based on OData metadata and the configuration specified. The entitySet attribute must be specified to use the control. This attribute is used to fetch fields from OData metadata, from which columns will be generated; it can also be used to fetch the actual table data. Based on the tableType property, this control will render a standard, analytical, tree, or responsive table.
Split Pane sap.ui.layout.SplitPane	A container of a single control in a responsive splitter. Could be used as an aggregation of a `PaneContainer`.

SAPUI5 Control Name	Description
Splitter Layout Data sap.ui.layout.SplitterLayoutData	Holds layout data for the splitter contents. Allowed size values are numeric values ending in "px" and "%" and the special case "auto". (The CSS value "auto" is used internally to recalculate the size of the content dynamically and is not directly set as style property.)
Standard List Item sap.m.StandardListItem	List item that provides the most common use cases, such as image, title, and description.
Standard Tile sap.m.StandardTile	Displayed in the `Tile` container.
Switch sap.m.Switch	User interface control on mobile devices that is used for switching between binary states.
Table sap.m.Table	Provides a set of sophisticated and convenient functions for responsive table design.
Text sap.m.Text	Used for embedding longer text paragraphs that need text wrapping into your application.
Text Area sap.m.TextArea	Allows multiline text input.
Tile Container sap.m.TileContainer	Container that arranges same-size tiles on carousel pages.
Tile Content sap.m.TileContent	This control is used within the GenericTile control.
Title sap.ui.core.Title	Used to aggregate other controls.
Time Picker sap.m.TimePicker	A single-field input control that enables the users to fill time related input fields.
Toggle Button sap.m.ToggleButton	Control that toggles between pressed and normal state.
Toolbar sap.m.Toolbar	Horizontal container that is usually used to display buttons, labels, selects, and other input controls.

SAPUI5 Control Name	Description
Toolbar Layout Data sap.m.ToolbarLayoutData	Defines layout data for the `sap.m.Toolbar` items. **Note** `ToolbarLayoutData` should not be used together with `sap.m.ToolbarSpacer`.
Toolbar Separator sap.m.ToolbarSeparator	Creates a visual separator between toolbar items.
Toolbar Spacer sap.m.ToolbarSpacer	Adds horizontal space between toolbar items.
Upload Collection sap.m.UploadCollection	Allows users to upload single or multiple files.
Upload Collection Item sap.m.UploadCollectionItem	Provides information about uploaded files.
Variant Layout Data sap.ui.core.VariantLayoutData	Allows to add multiple LayoutData to one control in case that an easy switch of layouts (e.g. in a Form) is needed.
VBox sap.m.VBox	Builds the container for a vertical flexible box layout.
Vertical Layout sap.ui.layout.VerticalLayout	Layout in which the content controls are rendered one below the other.
XML View sap.ui.core.mvc.XMLView	A View defined using (P)XML and HTML markup.

Outline Tab

The SAPUI5 controls listed below are available only from the *Outline* tab in the layout editor.

Note

For more information about SAPUI5 controls, see UI development toolkit for HTML5 - Demo Kit.

SAPUI5 Controls Available on the Outline Tab

SAPUI5 Control Name	Description
sap.m.semantic.AddAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an `sap.m.semantic.SemanticPage` control.
sap.m.semantic.CancelAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an `sap.m.semantic.SemanticPage` control.
sap.f.semantic.CloseAction	A semantic-specific button, eligible for the closeAction aggregation of the `sap.f.semantic.SemanticPage` to be placed in its title.
sap.f.semantic.CopyAction	A semantic-specific button, eligible for the copyAction aggregation of the `sap.f.semantic.SemanticPage` to be placed in its title.
sap.f.semantic.DeleteAction	A semantic-specific button, eligible for the deleteAction aggregation of the `sap.f.semantic.SemanticPage` to be placed in its title.
sap.f.semantic.DiscussInJamAction	A semantic-specific button, eligible for the discussInJamAction aggregation of the `sap.f.semantic.SemanticPage` to be placed in the share menu within its title.
sap.m.semantic.DiscussInJamAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an `sap.m.semantic.SemanticPage`.
sap.f.semantic.EditAction	A semantic-specific button, eligible for the editAction aggregation of the `sap.f.semantic.SemanticPage` to be placed in its title.
sap.m.semantic.EditAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an `sap.m.semantic.SemanticPage` control.
sap.f.semantic.ExitFullScreenAction	A semantic-specific button, eligible for the exitFullScreenAction aggregation of the `sap.f.semantic.SemanticPage` to be placed in its title.
sap.f.semantic.FavoriteAction	A semantic-specific button, eligible for the favoriteAction aggregation of the `sap.f.semantic.SemanticPage` to be placed in its title.

SAPUI5 Control Name	Description
sap.m.semantic.FavoriteAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an `sap.m.semantic.SemanticPage` control.
sap.m.semantic.FilterAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an `sap.m.semantic.SemanticPage` control.
sap.m.semantic.FilterSelect	A `sap.m.Select` control enhanced with styling according to the semantics of a common "Filter" action. A `FilterSelect` cannot be used independently but only as aggregation content of a `sap.m.semantic.SemanticPage`.
sap.f.semantic.FlagAction	A semantic-specific button, eligible for the flagAction aggregation of the `sap.f.semantic.SemanticPage` to be placed in its title.
sap.m.semantic.FlagAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an `sap.m.semantic.SemanticPage` control.
sap.f.semantic.FooterMainAction	A semantic-specific button, eligible for the footerMainAction aggregation of the `sap.f.semantic.SemanticPage` to be placed in its footer.
sap.m.semantic.ForwardAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an `sap.m.semantic.SemanticPage` control.
sap.f.semantic.FullScreenAction	A semantic-specific button, eligible for the fullScreenAction aggregation of the `sap.f.semantic.SemanticPage` to be placed in its title.
sap.m.semantic.GroupAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an `sap.m.semantic.SemanticPage` control.
sap.m.semantic.GroupSelect	A `sap.m.Select` control enhanced with styling according to the semantics of a common "Group" acton. A `GroupSelect` cannot be used independently but only as aggregation content of a `sap.m.semantic.SemanticPage`.

SAPUI5 Control Name	Description
sap.m.semantic.MainAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an `sap.m.semantic.SemanticPage` control.
sap.f.semantic.MessagesIndicator	A semantic-specific button, eligible for the messagesIndicator aggregation of the `sap.f.semantic.SemanticPage` to be placed in its footer.
sap.m.semantic.MessagesIndicator	Has default semantic-specific properties and is eligible to be included in the aggregation content of an `sap.m.semantic.SemanticPage` control.
sap.m.semantic.MultiSelectAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an `sap.m.semantic.SemanticPage` control.
sap.f.semantic.NegativeAction	A semantic-specific button, eligible for the negativeAction aggregation of the `sap.f.semantic.SemanticPage` to be placed in its footer.
sap.m.semantic.NegativeAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an `sap.m.semantic.SemanticPage` control.
sap.m.semantic.OpenInAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an `sap.m.semantic.SemanticPage` control.
sap.f.semantic.PositiveAction	A semantic-specific button, eligible for the positiveAction aggregation of the `sap.f.semantic.SemanticPage` to be placed in its footer.
sap.m.semantic.PositiveAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an `sap.m.semantic.SemanticPage` control.
sap.f.semantic.PrintAction	A semantic-specific button, eligible for the printAction aggregation of the `sap.f.semantic.SemanticPage` to be placed in the share menu within its title.
sap.m.semantic.PrintAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an `sap.m.semantic.SemanticPage` control.

SAPUI5 Control Name	Description
sap.m.semantic.SaveAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an `sap.m.semantic.SemanticPage` control.
sap.f.semantic.SendEmailAction	A semantic-specific button, eligible for the sendEmailAction aggregation of the `sap.f.semantic.SemanticPage` to be placed in the share menu within its title.
sap.m.semantic.SendEmailAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an `sap.m.semantic.SemanticPage` control.
sap.f.semantic.SendMessageAction	A semantic-specific button, eligible for the sendMessageAction aggregation of the `sap.f.semantic.SemanticPage` to be placed in the share menu within its title.
sap.m.semantic.SendMessageAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an `sap.m.semantic.SemanticPage`.
sap.f.semantic.ShareInJamAction	A semantic-specific button, eligible for the shareInJamAction aggregation of the `sap.f.semantic.SemanticPage` to be placed in the share menu within its title.
sap.m.semantic.ShareInJamAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an `sap.m.semantic.SemanticPage` control.
sap.m.semantic.SortAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an `sap.m.semantic.SemanticPage` control.
sap.f.semantic.TitleMainAction	A semantic-specific button, eligible for the titleMainAction aggregation of the `sap.f.semantic.SemanticPage` to be placed in its title.
sap.m.Title	Used for header texts and title.
sap.m.Token	Renders a token containing text and an optional *Delete* icon.
sap.m.Tokenizer	Displays multiple tokens.
sap.suite.ui.commons.BusinessCard	Allows displaying of business card information, including an image, first title (either URL link or text), second title, and multiple text lines.

SAPUI5 Control Name	Description
sap.suite.ui.commons.ComparisonChart	Displays a comparison chart.
sap.suite.ui.commons.ComparisonData	Comparison tile value holder.
sap.suite.ui.commons.CountingNavigationItem	Extends the `sap.ui.ux3.NavigationItem` control. This control displays the number of items in a corresponding content area. It also provides a rich tooltip that appears and disappears after a certain delay.
sap.suite.ui.commons.DateRangeScroller	Provides a method to scroll through a series of time periods, each of which is represented by a start date and an end date, known as the date range.
sap.suite.ui.commons.DateRangeSliderInternal	Provides the user with a `RangeSlider` control that is optimized for use with dates.
sap.suite.ui.commons.DeltaMicroChart	Displays a delta of two values as a chart.
sap.suite.ui.commons.DynamicContainer	Displays multiple `GenericTile` controls as changing slides.
sap.suite.ui.commons.FacetOverview	Used in `UnifiedThingInspector` to display a preview of facet content.
sap.suite.ui.commons.GenericTile	Tile control that displays a title, description, and customizable main area.
sap.suite.ui.commons.HarveyBallMicroChart	Chart that shows a comparative part to a total.
sap.suite.ui.commons.HarveyBallMicroChartItem	Configuration of a slice on a pie chart.
sap.suite.ui.commons.HeaderCell	Contains four cells (West, North, East, South). It can display one or more controls in different layouts. Each aggregation must contain only one instance of `HeaderCellItem`.
sap.suite.ui.commons.HeaderCellItem	Object that contains an instance of a control and information about its height. It should be used inside `sap.suite.ui.commons.HeaderCell`.
sap.suite.ui.commons.HeaderContainer	Container that provides a horizontal layout. On mobile devices, it provides a horizontal scroll. On desktops, it provides scroll left and scroll right buttons. This control supports keyboard navigation. You can use left and right arrow keys to navigate through the inner content. The `Home` key puts focus on the first control and the `End` key puts focus on the last control. Use the `Enter` key or `Spacebar` key to choose the control.

SAPUI5 Control Name	Description
sap.suite.ui.commons.JamContent	Displays SAP Jam content text, subheader, and numeric value in a tile.
sap.suite.ui.commons.KpiTile	Used in `UnifiedThingInspector` to display object-related KPIs in a factsheet.
sap.suite.ui.commons.NewsContent	Displays news content text and subheader in a tile.
sap.suite.ui.commons.NoteTaker	Allows creation and storage of notes for further reference.
sap.suite.ui.commons.NoteTakerCard	Allows storage of `NoteTaker` card header and body text.
sap.suite.ui.commons.NoteTakerFeeder	Allows entering quick notes and note cards.
sap.suite.ui.commons.NumericContent	Numeric content to be used in a tile or other place where it is needed to show numeric values with semantic colors and deviations.
sap.suite.ui.commons.TileContent	Serves as a universal container for different types of content and footer.
sap.ui.commons.ApplicationHeader	Located at the top of an application page and consists of four areas.
sap.ui.commons.Button	Allows users to trigger actions such as save or print. For the button UI, you can define text or an icon, or both.
sap.ui.commons.CheckBox	Provides a box that can be flagged and has a label. A checkbox can either stand alone, or be in a group with other checkboxes.
sap.ui.commons.ColorPicker	Allows the user to choose a color. The color can be defined using HEX, RGB, or HSV values, or a CSS colorname.
sap.ui.commons.FileUploader	Framework that generates an input field and a button with the text *Browse...*.
sap.ui.commons.MenuBar	Represents a user interface area that is the entry point for menus with their menu items.
sap.ui.commons.MenuButton	Common button control that opens a menu when chosen by the user. The control provides an API for configuring the docking position of the menu.
sap.ui.commons.Paginator	Provides navigation between pages within a list of numbered pages.
sap.ui.commons.Panel	Represents a container with scroll functionality that can be used for text and controls.

SAPUI5 Control Name	Description
sap.ui.commons.PasswordField	Text field with masked characters that borrows its properties and methods from the `TextField` control.
sap.ui.commons.ProgressIndicator	Shows the progress of a process in a graphical way.
sap.ui.commons.RadioButton	Consists of a round element and descriptive text.
sap.ui.commons.RangeSlider	Interactive control that is displayed either as a horizontal or vertical line with two pointers and units of measurement.
sap.ui.commons.RatingIndicator	Allows the user to rate a certain topic.
sap.ui.commons.SegmentedButton	Provides a group of buttons.
sap.ui.commons.Slider	Interactive control that is displayed either as a horizontal or vertical line with a pointer and units of measurement.
sap.ui.commons.Splitter	Allows splitting the screen into two areas.
sap.ui.commons.TextArea	Control for entering or displaying multiple rows of text.
sap.ui.commons.TextField	Renders an input field for text input.
sap.ui.commons.Toolbar	Horizontal row of items where in many cases the single toolbar items are buttons that contain icons.
sap.ui.commons.Tree	Simple tree for displaying an item in a hierarchical way.
sap.ui.commons.TreeNode	Tree node element.
sap.ui.layout.FixFlex	Builds the container for a layout with a fixed and a flexible part.
sap.ui.layout.form.FormLayout	Base layout used to render a `Form` control.
sap.ui.layout.form.GridLayout	Renders a `Form` control using an HTML table-based grid.
sap.ui.layout.form.ResponsiveGridLayout	Renders a `Form` control using a responsive grid.
sap.ui.layout.form.ResponsiveLayout	Renders a `Form` control with a responsive layout.
sap.ui.layout.ResponsiveSplitter	A responsive splitter which divides the application into several areas.
sap.ui.unified.Currency	Text view that displays currency values and aligns them at the separator.
sap.ui.unified.FileUploader	Framework that generates an input field and a button with the text *Browse ...*.

SAPUI5 Control Name	Description
sap.ui.unified.FileUploaderParameter	Represents a parameter for the `FileUploader`, which is rendered as a hidden input field.
sap.ui.unified.Menu	Interactive element that provides a choice of different actions to the user.
sap.ui.unified.MenuItem	Standard item used inside a menu. Represents an action that can be selected by a user in the menu or that can be used as a submenu that organizes the actions hierarchically.
sap.ui.unified.ShellOverlay	Opened in front of an `sap.ui.unified.Shell` control.
sap.ui.unified.SplitContainer	Provides a main content and a secondary content area.
sap.ui.ux3.ExactArea	Consists of two sections: a toolbar and a content area where arbitrary controls can be added.
sap.ui.ux3.FeedChunk	Unit that is embedded, standalone or multiple, into a `Feed` control.
sap.ui.ux3.Feeder	Lean common feed, or a comment feed, with a text commit function.
sap.uxap.ObjectPageHeader	Static part of an `Object` page header.
sap.uxap.ObjectPageHeaderContent	Dynamic part of an `Object` page header.

15.3.7.4.3 Try It: Build an Application with the Layout Editor

Get an overview of the features that are available with the layout editor by following this tutorial for building an application.

Related Information

Prerequisites [page 1190]
Create an OData Model File [page 1190]
Create a Project for Your New Application [page 1191]
Add Controls to Your New Application [page 1193]

15.3.7.4.3.1 Prerequisites

Prerequisite steps that you must complete before following the steps in the tutorial topic "Add Controls to Your New Application."

1. You have created an OData model file according to the instructions in the topic Create an OData Model File [page 1190].
2. You have created a new project from a template according to the instructions in the topic Create a Project for Your New Application [page 1191].

Related Information

Create an OData Model File [page 1190]
Create a Project for Your New Application [page 1191]
Add Controls to Your New Application [page 1193]

15.3.7.4.3.1.1 Create an OData Model File

This task is a prerequisite for the tutorial on building an application with the layout editor.

Procedure

1. Open a new text file.
2. Copy and paste the XML code provided below into the text file.

```
<edmx:Edmx Version="1.0" xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/
edmx" xmlns:sap="http://www.sap.com/Protocols/SAPData">
<edmx:DataServices xmlns:m="http://schemas.microsoft.com/ado/2007/08/
dataservices/metadata" m:DataServiceVersion="2.0">
<Schema xmlns="http://schemas.microsoft.com/ado/2008/09/edm"
Namespace="MySalesOrders">
<EntityType Name="SalesOrder">
 <Key>
  <PropertyRef Name="SalesOrderNumber" />
  </Key>
  <Property Name="SalesOrderNumber" Type="Edm.String" sap:label="Sales Order
Number" Nullable="false" MaxLength="10" />
  <Property Name="TotalAmount" Type="Edm.Decimal" sap:label="Total Amount"
Precision="16" Scale="3" sap:unit="Currency" MaxLength="10" />
  <Property Name="Currency" Type="Edm.String" sap:label="Currency"
MaxLength="5" sap:semantics="currency-code" />
  <Property Name="CustomerID" Type="Edm.String" sap:label="Customer ID"
MaxLength="10" />
  <Property Name="CustomerName" Type="Edm.String" sap:label="Customer Name"
MaxLength="35" />
  <Property Name="NetPriceAmount" Type="Edm.Decimal" sap:label="Net Price
Amount" Precision="16" Scale="3" sap:unit="Currency" />
  <Property Name="TaxAmount" Type="Edm.Decimal" sap:label="Tax Amount"
Precision="16" Scale="3" sap:unit="Currency" />
```

```
  <Property Name="OrderDate" Type="Edm.String" sap:label="Order Date" />
  <Property Name="RequestedDate" Type="Edm.String" sap:label="Requested
Date" />
  <Property Name="Status" Type="Edm.String" MaxLength="1"
sap:label="Status" />
  <Property Name="SalesOrganization" Type="Edm.String" MaxLength="4"
sap:label="Sales Organization" />
  <Property Name="SalesOrganizationName" Type="Edm.String" MaxLength="20"
sap:label="SalesOrganizationName" sap:creatable="false" sap:updatable="false"
sap:sortable="false" sap:filterable="false" />
  <Property Name="DistributionChannel" Type="Edm.String" MaxLength="4"
sap:label="Distribution Channel" />
  <Property Name="Division" Type="Edm.String" MaxLength="2"
sap:label="Division" />
  <Property Name="DistributionChannelName" Type="Edm.String"
MaxLength="20" sap:label="DistributionChannelName" sap:creatable="false"
sap:updatable="false" sap:sortable="false" sap:filterable="false" />
  <Property Name="DivisionName" Type="Edm.String" MaxLength="20"
sap:label="DviisionName" sap:creatable="false" sap:updatable="false"
sap:sortable="false" sap:filterable="false" />
  </EntityType>
<EntityContainer Name="MySalesOrders_Entities"
m:IsDefaultEntityContainer="false">
  <EntitySet Name="SalesOrders" EntityType="MySalesOrders.SalesOrder"
sap:searchable="true" sap:requires-filter="true" />
  </EntityContainer>
  </Schema>
  </edmx:DataServices>
  </edmx:Edmx>
```

3. Save the file to your computer with the file name `SalesOrderService_metadata.xml`.

15.3.7.4.3.1.2 Create a Project for Your New Application

This task is a prerequisite for the tutorial on building an application with the layout editor.

Context

Create a new project for a sales order tracking application using a template.

Procedure

1. Open SAP Web IDE in the Google Chrome browser.
2. Open a new project from a template by using one of the following options:
   - On the SAP Web IDE Welcome page, choose *New Project from Template*.
   - In the *File* menu, choose *New* > *Project from Template*.

   The *New Project* wizard opens.
3. In the *Template Selection* wizard step, select the *SAP Fiori Master Detail Application* tile and then choose the *Next* button.

4. In the *Basic Information* wizard step, enter a project name. Choose the *Next* button.

> **i Note**
>
> The project name must start with a letter or an underscore and may contain alphanumeric characters, periods, and underscores. It may not end with a period.

5. In the *Data Connection* wizard step, select the *File System* source. Then choose the *Browse* button and navigate to the file `SalesOrderService_metadata.xml` that you created in the topic Create an OData Model File [page 1190]. Choose the *Next* button.
6. In the *Template Customization* wizard step, enter or select the mapping data in the fields and dropdown lists according to the tables below.

Application Settings Section

Field or Dropdown List	Value
Title	**`Sales Orders`**
Project Namespace	**`<project_namespace>`** **i Note** The namespace must start with a letter or an underscore and may contain alphanumeric characters, periods, and underscores. It may not end with a period.
OData Collection	*SalesOrders*
Item Title	*SalesOrderNumber*
Numeric Attribute	*TotalAmount*
Unit of Measure	*Currency*

Detail Section

Field or Dropdown List	Value
Title	**`Sales Order`**
Detail Text	**`Sales Order Details`**
Status Attribute	Leave it blank
Attribute 1	*NetPriceAmount*
Attribute 2	*TaxAmount*
Attribute 3	*OrderDate*

7. Choose the *Next* button.
8. In the *Confirmation* wizard step, choose the *Finish* button.

Your new project is now created in your workspace.

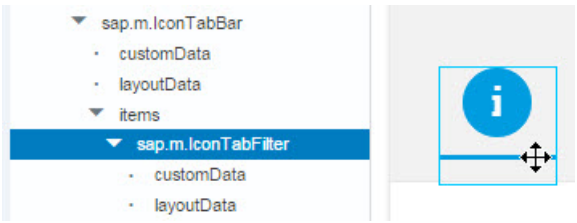
15.3.7.4.3.2 Add Controls to Your New Application

Steps for adding controls to your new sales order trackOpen Withing application.

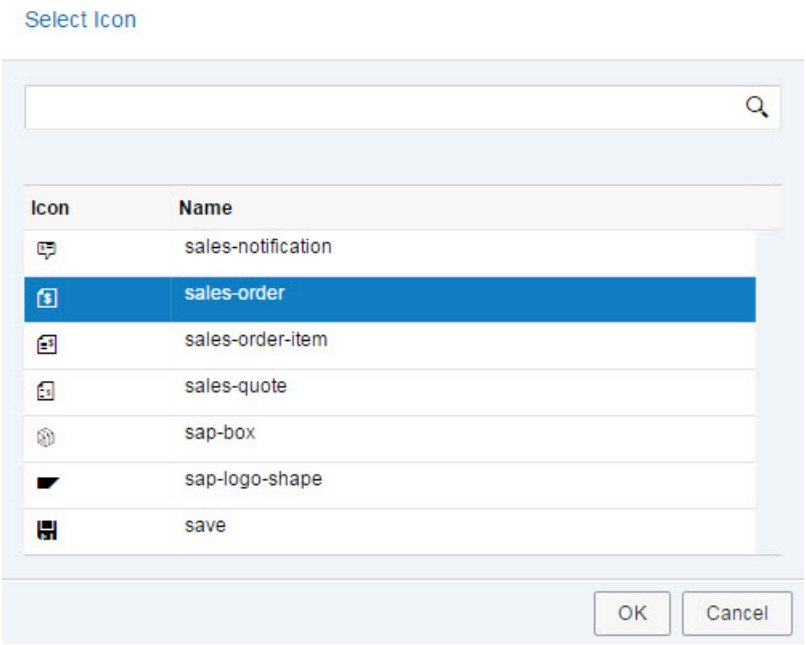
Procedure

1. In the workspace, expand the folder with name of the project that you created in the topic Create a Project for Your New Application [page 1191], then expand the `view` folder and right-click the `Detail.view.xml` view.
2. From the context menu, choose *Layout Editor*.
   The content of the XML view is displayed on the canvas in a way that corresponds to how it will appear in your finished application.
3. Change the icons of the *Icon Tab Filter* controls:

   a. On the canvas, select the first *Icon Tab Filter* control on the left side of the *Icon Tab Bar* control. In the *Outline* tab on the left side of the canvas, *sap.m.IconTabFilter* is selected.



   b. In the *Properties* pane to the right of the canvas, in the *Icon* field, choose the *Select Icon* button to open the *Select Icon* dialog box. Scroll down and select the *sales-order* icon and then choose *OK*.



c. On the canvas, select the second *Icon Tab Filter* control.

d. In the *Properties* pane, in the *Icon* field, choose the button to open the *Select Icon* dialog box. Scroll down and select the *sales-order-item* icon and then choose *OK*.

4. Add a new *Icon Tab Filter* control to the view:

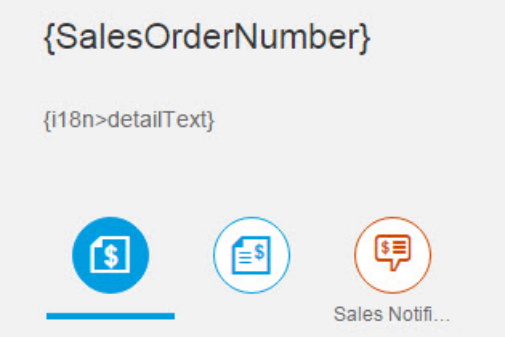
a. On the *Controls* tab to the left of the canvas, expand the *Container* section or use the search field to search for the *Icon Tab Filter* control.

b. From the *Controls* tab, drag the *Icon Tab Filter* control to the canvas and drop it on the *Icon Tab Bar* control.

c. Change its icon to the *sales-notification* icon in the same way as you changed the icons in step 3.

d. In the *Properties* pane, in the *Count* field, clear the value by deleting it.

e. In the *Properties* pane, change the value in the *Text* field to *Sales Notifications*.



5. Add a *Simple Form* control to the new sales notification *Icon Tab Filter* control:

a. On the *Controls* tab to the left of the canvas, expand the *Layout* section.

b. In the canvas, select the *Sales Notification Icon Tab Filter*.

c. From the *Controls* tab on the left, drag the *Simple Form* control to the canvas and drop it on the space below the *Icon Tab Filter* control.

6. Change the properties of the new *Simple Form* control:

a. In the canvas, select the title in the *Simple Form* control.

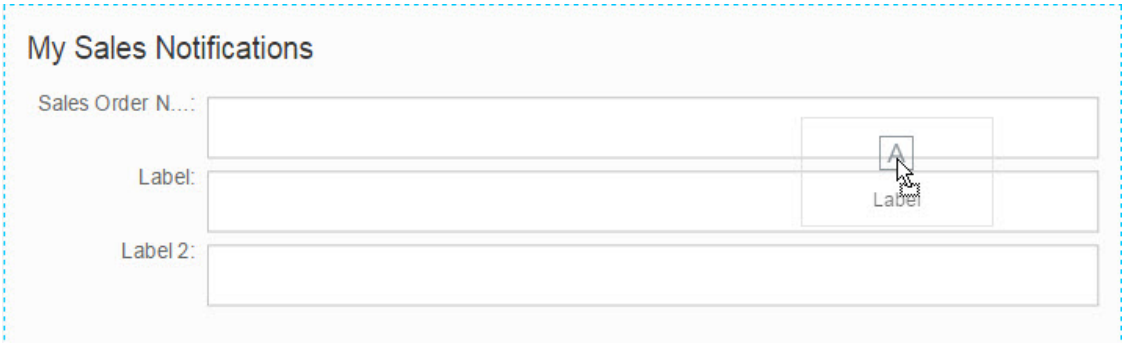
b. In the *Properties* pane, change the value of the *Text* property to **`My Sales Notifications`**.

c. In the canvas, in the *Simple Form* control, select *Label 1*.

d. In the *Properties* pane, change the value of the *Text* property to **`Sales Order Number`**.

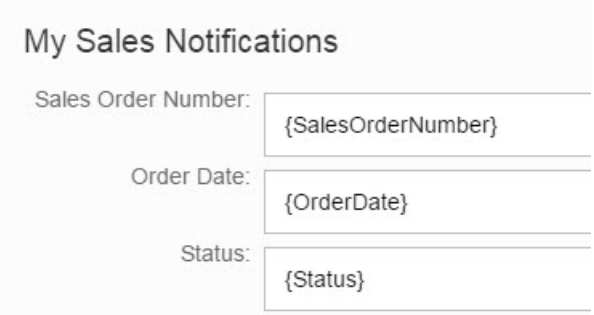
e. On the *Controls* tab to the left of the canvas, search for the *Label* control.

f. Drag a new label and drop it above the second input field.



g. In the canvas, in the *Simple Form* control, select the label that you just created and then in the properties pane to the right of the canvas, change its text to **`Order Date`**.

h. Change the text of the *Label 2* control of the third input field in the same way to **`Status`**.

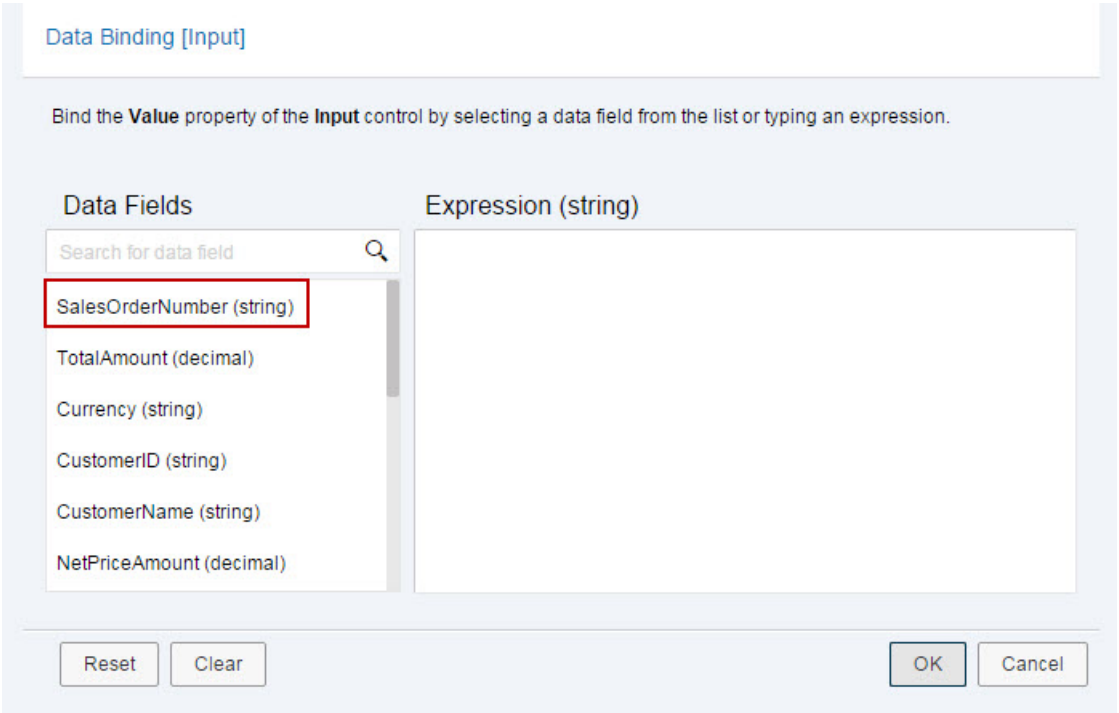


7. Bind the *Value* property of the input fields to elements from the OData service that you created:

a. In the canvas, select the *Sales Order Number* field in the *SimpleForm* control.

b. In the *Properties* pane, click the *Data Binding* button next to the *Value* property.

c. In the *Data Binding [Input]* dialog box, select *SalesOrderNumber* and choose *OK*.



d. In the canvas, select the *Order Date* field in the *Simple Form* control.

e. In the *Properties* pane, click the *Data Binding* button next to the *Value* property.

f. In the *Data Binding* dialog box, select *OrderDate* and choose *OK*.

g. Select the *Status* field and bind it to **`Status`** in the same way as the previous two input fields.

8. Save your changes.
   Your changes are saved for both the layout editor and the code editor.
9. Run your application to test the result.

15.3.7.5 Annotation Modeler

The annotation modeler provides an intuitive user interface for annotating OData services. It enables you to build and enhance the user interface of model-based applications by adding and editing UI annotations of the local annotation file.

You can also override annotations from other sources, such as metadata, by editing them in the local annotation file.

To run and use annotation modeler with your annotation file, some prerequisites need to be fulfilled. See Prerequisites for Working with Annotation Modeler [page 1200].

> **i** Note
>
> You can use annotation modeler with a *List Report Page* module to build applications based on OData services and annotations that require no JavaScript UI coding. See Developing Apps with SAP Fiori Elements.

Related Information

Architecture [page 1196]
User Interface [page 1198]
Prerequisites for Working with Annotation Modeler [page 1200]
Adding Local Annotation Files to the Project [page 1202]
Working with Annotation Modeler [page 1203]

15.3.7.5.1 Architecture

Infrastructure

The annotation modeler is embedded in the SAP Web IDE infrastructure and can be used directly in SAP Web IDE. Simply enable the annotation modeler feature (extension), see Enable SAP Web IDE Extensions [page 1066].

Supported OData Vocabularies

The annotation modeler is based on the official OASIS vocabularies (OData version 4.0) and specific SAP vocabularies that are stored as local copies in SAP Web IDE annotation modeler.

The annotation modeler recognizes annotation files containing the terms from these vocabularies and displays them in a tabular view. You can also add, edit, and delete annotations that are defined in these vocabularies and that apply to the levels listed below.

The annotation modeler supports only standard vocabularies. The following vocabularies are supported:

- **OData.org:**
  - Aggregation
  - Authorization
  - Capabilities
  - Core
  - Measures
  - Validation
- **SAP:**
  - Analytics
  - CodeList
  - Common
  - Communication
  - PersonalData
  - UI

> **i Note**
>
> The annotation modeler might not support all vocabulary terms listed in the mentioned OData vocabularies.

> **→ Tip**
>
> To find out more about how to prepare UI annotations for SAP Fiori elements, see SAPUI5 Developer Guide: Preparing UI Annotations.

Scope

The annotation modeler supports applying annotations to the following targets:

- Actions
- Action Imports
- Entity Container
- Entity Set
- Entity Type

- Entity Type Property

  > **Note**
  >
  > Exception: Entity type properties of complex type are not supported.

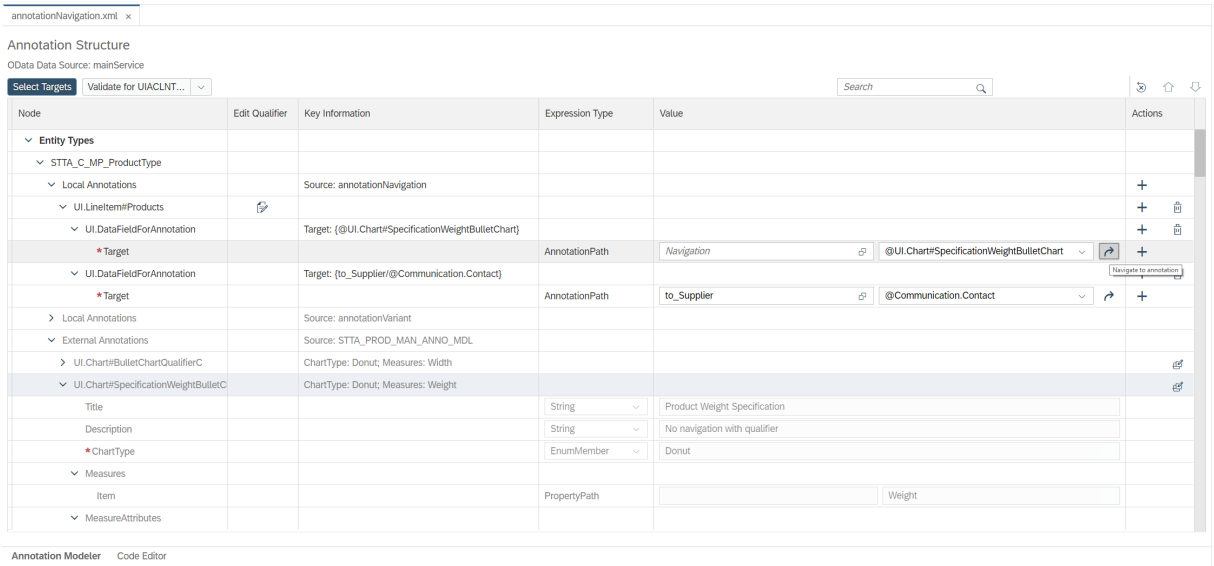
- Function
- Function Import
- Function Import Parameters

The annotation modeler also allows you to annotate the collection records and property values of the annotations.

Limitations

- The annotation modeler only supports the services of OData version 2.0.

15.3.7.5.2 User Interface



Here, you perform the following actions:

Edit annotations in your local annotation file

- Annotation modeler provides an overview of the annotations that are available in your project for the given data source of the OData type. You can see the name of this data source at the top of the page, details are displayed in toopltips.
  In the table, the annotations are displayed under the targets they apply to. Annotations that are applied to the same target are grouped in nodes according to their annotation origin. Details of the annotation origin are displayed as tooltips of the *Key Information* column.

> **i Note**
>
> The sequence of the nodes under each target depends on the sequence of their nomination in the `manifest.json` file. This sequence also defines which annotation overrides another one. See Edit External Annotations [page 1216].

- You can edit *Local Annotations* from the currently open file in annotation modeler. See Edit Annotations [page 1204]. Annotations originating from other local files or external sources relevant to the same data source of OData type are non-editable and cannot be changed directly. Annotations from other data sources are not displayed in the tree and can be viewed and edited in their respective annotation files.
- You can search through all annotations defined for the currently selected targets to locate annotations that correspond to the UI elements that you want to change. This also includes the annotations defined in sources other than the current file. See Search in Annotations [page 1212].
- You can reset annotation modeler to default view by choosing the *Reset to default view* button. This clears the search and collapses all the nodes that group annotations by origin. All your annotation changes are saved.
- You can change the sequence of records within the collections. See Change the Order of Annotations and Collection Items [page 1213].

Define properties in your local annotation file

- By default, the mandatory properties and the annotated optional properties are displayed. Mandatory properties are marked with an asterisk (*). You can add more properties from the *Actions* column to the annotation.

> **→ Tip**
>
> Annotation modeler provides vocabulary descriptions of terms and properties in tooltips.
>
> Tooltips are shown only for terms and properties that have descriptions in the OData vocabularies.

- The validation run highlights all mandatory and optional properties that do not have values added. The error tooltip states that the value for the property is missing.
- If you do not enter an ID if there are more than one facet ID fields of a given hierarchy level, the entry field is highlighted yellow and the warning tooltip states that the ID is missing.
- The internationalization details for language-dependent texts, such as labels, are also displayed with the properties. You can display and update the information using the *i18n Details* icon. See Create Internationalized Labels [page 1214].
- You can concatenate several values for all terms containing the *Value* property. See Concatenate Values [page 1215].
- You can insert parameters in any part of a URL. See Use Parameters in URLs [page 1215].

Add new annotations

You can add the terms you want to annotate to your local annotation file. See Add Annotation Terms, Items, and Records to the Local Annotation File [page 1207].

Override external annotations by cloning them to the local annotation file

External annotations and annotations from local annotation files, other than the annotation file that is currently open, are read-only and cannot be edited directly. To override an annotation, you can clone it to the local annotation file that is currently open in annotation modeler. Annotations can only be cloned to a local annotation file that uses the same OData data source as the external annotation file that includes the original

annotation. When you clone the annotation, the clone appears in the annotation file that is currently open and you can edit it there. In this case, the original annotation is marked as overridden (strikethrough) and its local clone is used to display the data on the UI. See Edit External Annotations [page 1216].

> → Tip
>
> Use the *Annotation Modeler* and *Code Editor* tabs to switch between these editors. This option is only available if you select annotation modeler as your default editor for annotation files.
>
> For more information about how to set default editors, see Setting User Preferences [page 1065].

15.3.7.5.3 Prerequisites for Working with Annotation Modeler

To adapt the UI of your application with annotation modeler, your SAP Web IDE project needs to meet the following criteria:

OData Service

- You have prepared an OData service in the same project and referenced it during the module creation.
- The metadata of the OData service must include one or multiple *edm:Schema* definitions within the *edmx:DataServices* element.
  According to the OData CSDL , your metadata file must contain a single EntityContainer.

> ⚠ Caution
>
> The namespace of the OData service should not contain / (slashes). The OData specification requires namespaces to consist of one or more SimpleIdentifiers separated by dots. Slashes are not supported.
>
> A SimpleIdentifier must start with a letter or underscore, followed by a maximum of 127 letters, underscores and digits.

> i Note
>
> For UI prototyping, you may use the metadata file from your workspace or from the file system when creating the relevant module.

Local Annotation File

- Your project contains a valid annotation XML file that includes the */edmx:DataServices/Schema* node.

  > → Tip
  >
  > If your project does not contain an `annotation.xml` file, you can create a new one or import the file from your workspace or file system. See Create New Annotation Files [page 1203] and Import Existing Annotation Files [page 1202].

Manifest.json File

- Your project contains a `manifest.json` file.

  > **i** Note
  >
  > All paths used in annotation modeler are relative to the location of the `manifest.json` file.

- The object `dataSources/mainService/localUri` in the `manifest.json` file must contain a relative path to the local copy of the service metadata. Annotation Modeler on SAP Web IDE for SAP HANA always uses the metadata stored in the local copy. As there is no automatic update of this file, you should manually update the content of the local `metadata.xml` file if needed.

  > **i** Note
  >
  `Sap.ui.generic.app.pages[0].entitySet` is an **optional** setting in the `manifest.json` file. It is used to automatically select an entity type in a combobox in the *Select Targets* dialog when you open annotation modeler. If it is not present in the `manifest.json` file, you can select the desired entity type manually.

  > **→** Tip
  >
  For an overview of how to set the attributes in the `manifest.json` file, see Descriptor for Applications, Components, and Libraries.

Component.js File

- Your project contains a `component.js` file.
  The component.js file of your project references the manifest.json file as follows:

```
sap.ui.generic.app.AppComponent.extend("LRP.Component", {
    metadata: {
        “manifest”:"json",
    }
});
```

When the above-mentioned criteria are met, annotation modeler can be used to open and edit the annotation file.

Related Information

Working with Annotation Modeler [page 1203]

15.3.7.5.4 Adding Local Annotation Files to the Project

Annotation modeler lets you add or modify UI annotations in the local annotation file. To add annotation files to your project, you can do one of the following:

- Import an existing annotation file. See Import Existing Annotation Files [page 1202].
- Create a new annotation file. See Create New Annotation Files [page 1203].
  Afterwards, you can start working with annotations in your newly created annotation file.

Related Information

Import Existing Annotation Files [page 1202]
Create New Annotation Files [page 1203]

15.3.7.5.4.1 Import Existing Annotation Files

You can import existing annotations from the server.

Procedure

1. In your SAP Web IDE workspace, navigate to the project folder to which you want to import the annotation file.
2. Right-click the annotations folder of your project folder and choose *Import* > *Annotation File* in the context menu or click the same path in the *File* menu.
3. Select the annotation source and browse to an annotation XML file containing at least one annotation. Select the matching OData service, and choose *Finish*.
4. The annotation file is imported to the folder and is available for editing if the prerequisites are met. See Prerequisites for Working with Annotation Modeler [page 1200].

> **i** Note
>
> - If needed, you can rename the annotation file.
> - You cannot import a file that contains no annotations.

15.3.7.5.4.2 Create New Annotation Files

You can create annotation files from scratch using annotation modeler.

Procedure

1. Select the folder in which you want to create the annotation file.
2. Do one of the following:
   - Right-click the annotations folder of your project folder and choose *New* > *Annotation File*.
   - Choose *File* > *New* > *Annotation File*.
3. In the dialog box, enter a file name and select an OData service.

   An empty annotation file is created, containing headers and namespaces.

15.3.7.5.5 Working with Annotation Modeler

You can adjust the data displayed on the UI of your application by editing your project's local annotation files in annotation modeler.

Context

To start working with annotation modeler, proceed as follows:

Procedure

1. Create a project in SAP Web IDE.
2. Check whether the Prerequisites for Working with Annotation Modeler [page 1200] have been met.
   - If the prerequisites have been met, you can open and edit the annotation files in annotation modeler. See Open Local Annotation Files in Annotation Modeler [page 1204].
   - You can preview the annotated data if your project supports running the application from the workspace. See Run and Test HTML5 Modules [page 1220].

Related Information

Open Local Annotation Files in Annotation Modeler [page 1204]

Edit Annotations [page 1204]
Validation of the Annotation File [page 1217]

15.3.7.5.5.1 Open Local Annotation Files in Annotation Modeler

You can open local annotation files in annotation modeler.

Annotation Modeler as the Default Editor for Annotation Files

If you set annotation modeler as the default editor for annotations, you can open local annotation files in annotation modeler by double-clicking them.

> → Tip
>
> If you select annotation modeler as your default editor for annotations, you can switch between the code editor and annotation modeler at the bottom of the page. This option is not available if the code editor is the default editor for annotations.
>
> For more information about how to set default editors, see Setting User Preferences [page 1065].

Open Annotation Modeler from the Context Menu of the Annotation File

1. Right-click the local annotation file of your project.
2. Choose *Open Annotation Modeler*.

15.3.7.5.5.2 Edit Annotations

Context

You can adjust the data displayed on the UI of your application by editing your project's local annotation files in annotation modeler.

Items from external sources and from local annotation files other than the annotation file that is currently open are read-only. For more information about editing read-only elements, see Edit External Annotations [page 1216].

> ⚠ Caution
>
> Some smart controls may have specific prerequisites for defining their properties with annotations. As annotation modeler is a generic annotation editor, it does not validate against control-related restrictions or prerequisites.
>
> Please check the documentation of the respective controls before you start annotating: UI development toolkit for HTML5 - Demo Kit.

> ℹ Note
>
> Annotation modeler does not support entity type properties of complex type.

Procedure

1. Find the term you want to edit and expand it. Make sure it is located under the *Local Annotations* node of the desired target. To display additional targets, see Select Targets [page 1206].

   By default, the mandatory and the already defined optional properties are displayed under this term.
2. In the *Actions* column, choose the *Add* icon and select the annotation properties to be defined.
3. Use the *Expression Type* and *Value* columns to enter *Property* values.
   - In the *Expression Type* column, choose from the expression types that are available in the vocabulary. For language-dependent texts that appear in the application, such as labels, choose *String (i18n)*.
   - In the *Value* column, you can either type in a new value or select a value that already exists in the project. For the values of expression type *AnnotationPath*, you can then use the *Navigate to annotation* icon to navigate to the referenced annotation. In this case, additional targets may be added to the view.

     > ℹ Note
     >
     > When annotating function imports or function import parameters, you can only type in the values of the path expression type. Annotation modeler does not provide any input help for function import or function import parameter paths.

     For expressions of type *EnumMember*, if available in the vocabulary, descriptions for the values are shown as tooltips.
4. Use the *Edit Qualifier* column to uniquely define annotation terms. The *Edit Qualifier* icon is displayed only if qualifiers are applicable for the annotation.

   > ℹ Note
   >
   > If an annotation term appears in the local annotation file more than once, you have to enter a qualifier or an ID (for facets) for all but one of the annotation terms so that they are uniquely identified.
   >
   > - Make sure that the entered **ID** is unique among all annotations.
   > - Make sure that the **qualifier** is unique for the annotation term.
5. Use the *Actions* column to add subnodes and to delete nodes.
6. Save your entries.

- When you save, annotation modeler validates the entire annotation file.
- Run your application to see the results with the preview function of SAP Web IDE.

Related Information

Add Annotation Terms, Items, and Records to the Local Annotation File [page 1207]
Change the Order of Annotations and Collection Items [page 1213]
Create Internationalized Labels [page 1214]
Concatenate Values [page 1215]
Use Parameters in URLs [page 1215]
Edit External Annotations [page 1216]

15.3.7.5.5.2.1 Select Targets

Context

To add new or update the existing annotations, you select the targets these annotations apply to.

For the supported annotation levels, see Architecture [page 1196], section *Scope*.

Procedure

1. Choose the *Select Targets* button.

   The top-level target types are displayed in alphabetical order. By default, one entity type is automatically selected.

   > **i** Note
   >
   > The list of targets has a hierarchical structure similar to that of the metadata. Not all nodes are visible at all times.
   > - To view and select the entity type properties, expand the following nodes: `Entity Types/<entity type name>/Properties`.
   > - To view and select the entity sets, expand the following nodes: `Entity Containers/<entity container name>/Entity Sets`.
   > - To view and select the function imports, expand the following nodes: `Entity Containers/<entity container name>/Function Imports`.

- To view and select the function import parameters, expand the following nodes: `Entity Containers/<entity container name>/Function Imports/<function import name>/ Parameter`.

2. You can search the tree by entering a search term.

   The tree is filtered and expanded to show every target matching the search. If you clear the search term, the original state of the tree is recovered.

   > **i** Note
   >
   > You can search again to find additional targets to select. All your previous selections remain and become visible as soon as you clear the search.

3. Select or deselect the respective checkboxes.
4. Choose *OK*.

Results

The newly selected targets are displayed in the annotation structure. Once you have defined at least one annotation for each target, they are added to the annotation file as well.

15.3.7.5.5.2.2 Add Annotation Terms, Items, and Records to the Local Annotation File

Procedure

1. Select the targets your annotation should apply to. For more information, see Select Targets [page 1206].
2. To add annotations to the selected target, proceed as follows:
   a. Choose the *Add* icon in the *Actions* column of the underlying *Local Annotations* node level.
   b. Choose the annotation you want to add. The annotation is inserted above the others and is selected for editing.
3. For annotation terms that are collections of primitive types, you can add items. For example, *UI.Contacts* can contain several items. To add and annotate an item, do the following:
   a. In the row of the annotation term representing a collection of primitive types, in the *Actions* column, choose the *Add* icon.
   b. In the *Add* to dialog box, choose *Item*. Repeat these steps for each item you want to add.
   a. Annotate the items by choosing the *Add* icon on the item node.
4. For annotation terms that are collections of complex types, you can add records. For example, you can add records of complex type *UI.DataFieldAbstract* to *UI.LineItem*. To add and annotate a record, do the following:

a. In the row of the annotation term, in the *Actions* column, choose the *Add* icon.
b. In the *Add to* dialog box, choose the record you want to add (e.g. *DataField*). Repeat these steps for each record you want to add.
c. Annotate the records by choosing the *Add* icon on the record node.

Related Information

Example: Add a Field Group (UI.ReferenceFacet) for Technical Data to an Object Page [page 1208]

15.3.7.5.5.2.2.1 Example: Add a Field Group (UI.ReferenceFacet) for Technical Data to an Object Page

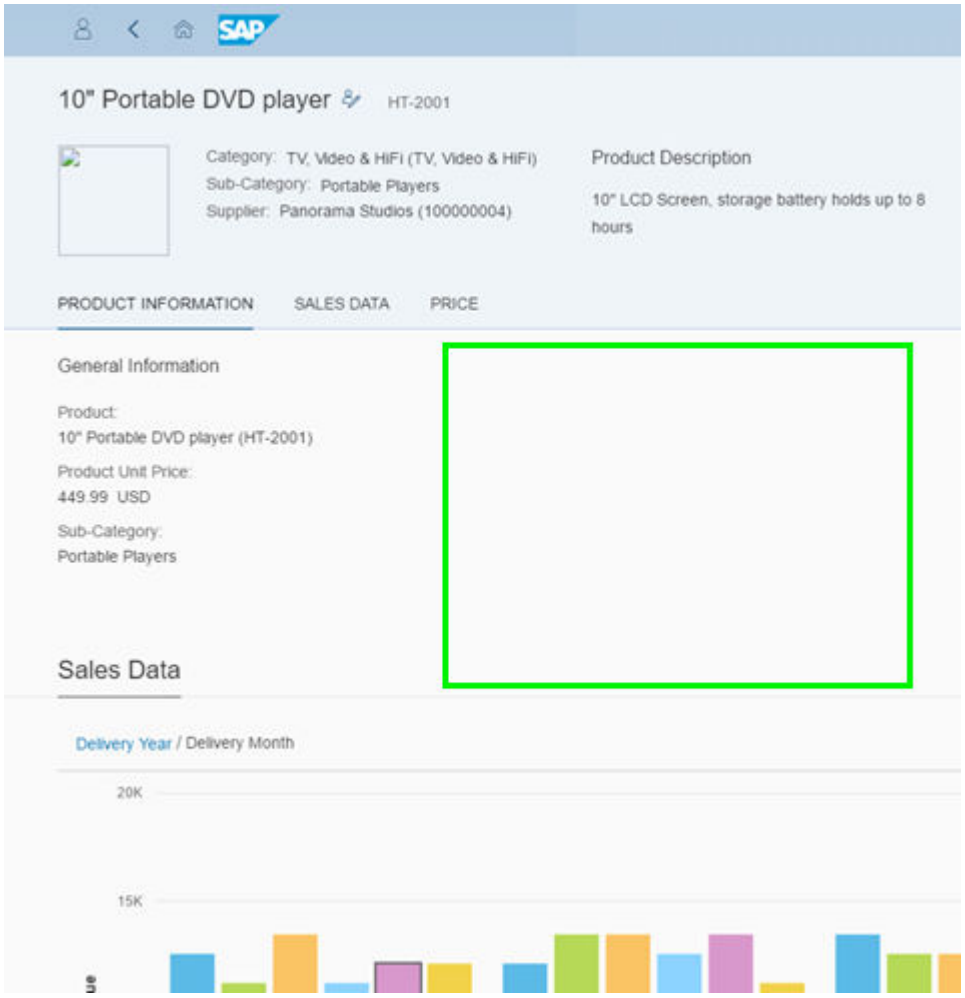
This section explains how to change the UI of your model-based application based on the sample task of adding a field group for technical data to an object page.

Context

In this sample project, we are working with a *List Report Page* project that uses an *Object Page* template.

Our goal is to add a new field group for the technical data of a product to the *General Information* section, and then edit the properties of this new field group. The new *Technical Data* field group should be displayed next to the existing *Product Information* field group.

> **i** Note
>
> In this example, the field group that includes the technical data already exists in the backend. We will create a new *UI.ReferenceFacet* and reference to the target *UI.FieldGroup*.



→ Tip

In applications using the object page template, the following applies:

- *UI.CollectionFacets* create main sections in the contact area of your application's UI (in this example: **`Product Information`**).
- *UI.ReferenceFacets* within a *UI.CollectionFacet* display the subsections that present the content of the referenced terms (in this example: the field group **`General Information`**).
  If *UI.ReferenceFacet* is not annotated as a child node of a *UI.CollectionFacet*, but appears on the same level in the annotation model. *UI.ReferenceFacet* creates its own main section that presents the content of the referenced terms.

For more information about the Object page view, see Object Page View.

Procedure

1. Open your project's annotation file with annotation modeler. See Open Local Annotation Files in Annotation Modeler [page 1204].

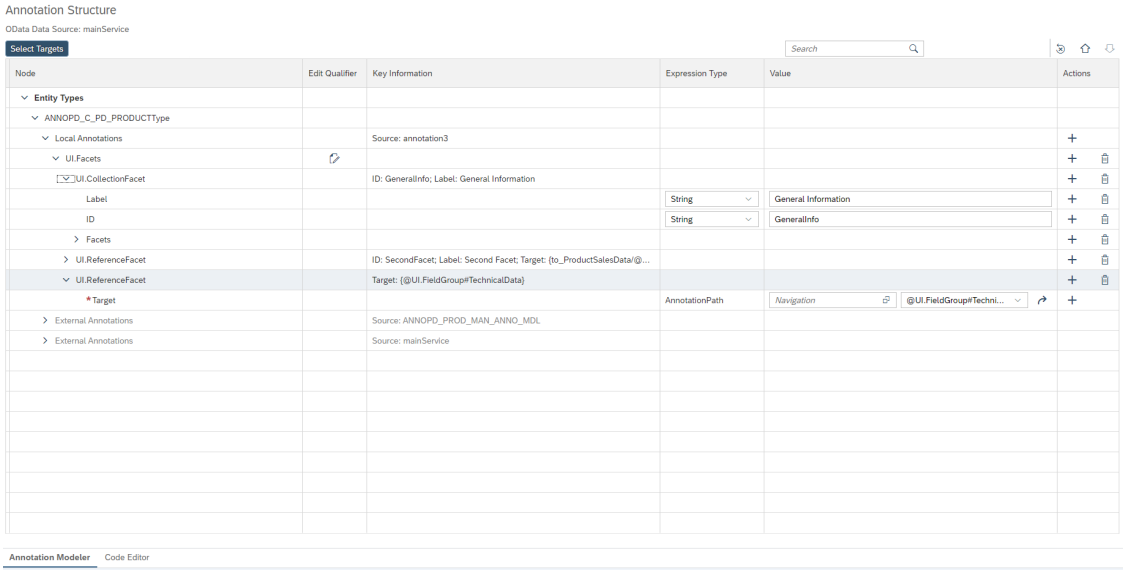
2. In annotation modeler, choose the *Select Target* dialog and select the entity type related to the object page you want to update. For more information, see Select Targets [page 1206].
3. Under the selected entity type, expand the *Local Annotations* node.
4. Expand the *UI.Facets* and the *UI.CollectionFacet* nodes to which you want to add a new *UI.ReferenceFacet*.

   To ensure that you add the new annotation at the right place, check the *Key Information* column.

   The entry in the *Label* property field of the *UI.CollectionFacet* represents the main section heading on the UI of your application. In this example, the main section header is **`Product Information`**.
5. Add a new *UI.ReferenceFacet* annotation.
   a. In the table row of the *Facets* node, choose the *Add* icon.

      A dialog box opens in which you can select all annotations and records that can be added as child nodes of the *UI.CollectionFacet*.
   b. In the dialog box, select *ReferenceFacet* and choose *OK*.

      A new *UI.ReferenceFacet* annotation is added under the *Facets* node. The mandatory properties of the new *UI.ReferenceFacet* are displayed in the table below the new *UI.ReferenceFacet*. In this example, for the *UI.ReferenceFacet*, *Target* is mandatory.



6. Before defining the property values, add the optional properties you require for your annotation.
   a. In the *UI.ReferenceFacet* row, choose the *Add* icon.

      A dialog box opens in which you select all properties that apply for the annotation.
   b. Select *Label* and *ID* and choose *OK*.

   The *Label* and *ID* rows are displayed under *UI.ReferenceFacet*.
7. Define the heading for the new field group in the *Label* row. In this example, we enter **`Technical Data`**.
   a. In the *Expression Type* column, make sure that *String (i18n)* is selected.
   b. In the *Value* column, enter *Technical Data*.
8. Define a unique value for the new *UI.ReferenceFacet* in the *ID* row. In this example, we apply the **`TechData`** ID.
   a. In the *Expression Type* column, make sure that *String* is selected.

b. In the *Value* column, enter *TechData*.

Each *UI.ReferenceFacet* requires a unique ID to ensure that it is displayed correctly on the application UI. This means that only one *UI.ReferenceFacet* in the annotation model can have an empty ID field, each additional *UI.ReferenceFacet* requires a unique value for the property *ID*.

If you create two or more *UI.ReferenceFacets* with the same ID, annotation modeler issues an error when you try to save your project or when you switch to another annotation.

9. Assign the *UI.FieldGroup* as content for *UI.ReferenceFacet*. In this example, we reference the *UI.FieldGroup* with the qualifier *TechnicalData* from the OData entity type *SEPMRA_C_PD_Product* .

   a. In the *Target* row, in the *Annotation* field, select *UI.FieldGroup#TechnicalData*.

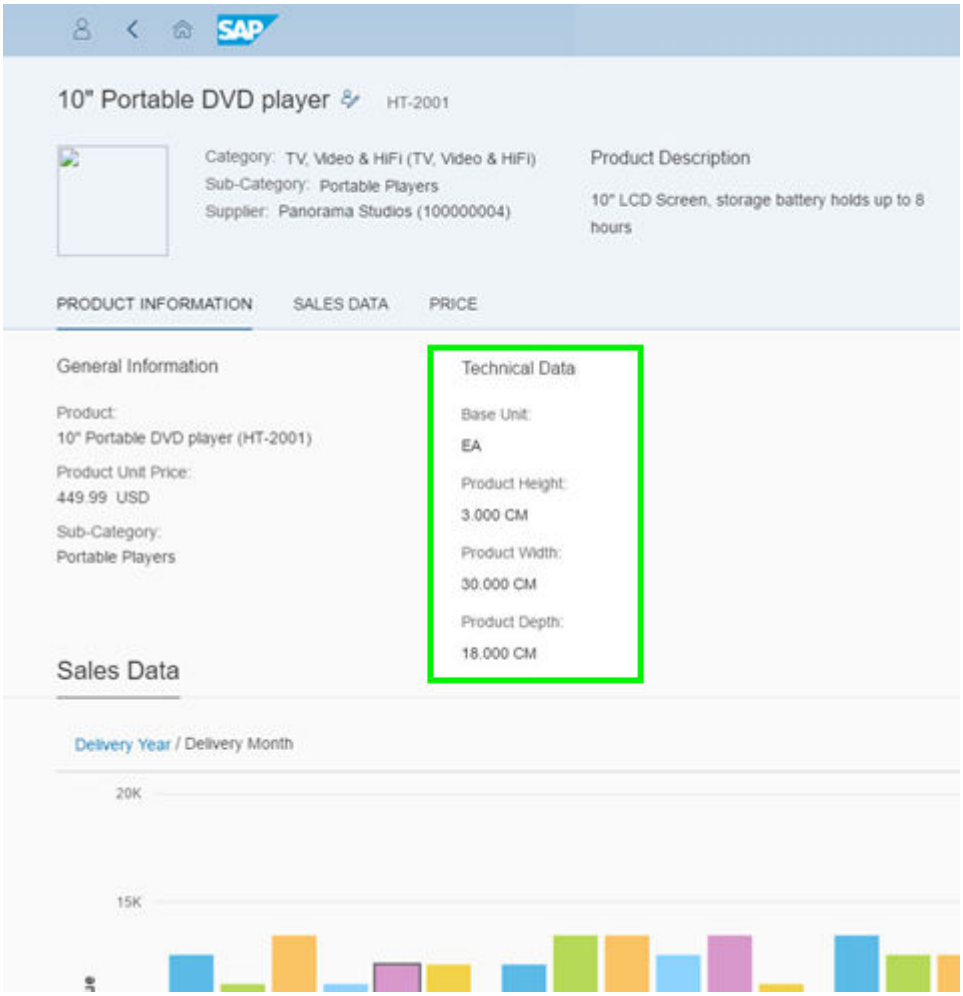
      The term following @ defines the annotation type. The term following # defines the qualifier. This means that *@UI.FieldGroup#TechnicalData* references the *UI.FieldGroup* annotation with the qualifier *TechnicalData*.

> **i Note**
>
> In the backend of this sample project, a *UI.FieldGroup* that contains all necessary technical data already exists. We reference this *UI.FieldGroup* in our new *UI.ReferenceFacet* to display the fields of this field group on the UI of the application. You can see how the referenced annotation is configured by choosing the *Navigate to annotation* icon. In this example, the *External Annotation* node expands and *@UI.FieldGroup#TechnicalData* is selected to draw your attention.

   - If you leave the *Navigation* field empty, you can reference annotations only from the currently selected OData entity type.
   - The content targets that are available in the *Annotation* dropdown menu depend on the selected *Navigation*.
     To reference an annotation from another OData entity type, select the entity type in the *Navigation* field before assigning the content target in the *Annotation* field. You can select only OData entity types that have a one-to-one relationship with the entity type that is currently selected.

10. When you have finished editing, save your project.
11. Run your application to see the changes you've made.

In this example, we have added a subsection to our main section *Product Information* that displays the technical data of the sample product:



→ Tip

To find out more about how facets annotations are used in applications based on the ObjectPage template, see Sections.

15.3.7.5.5.2.3 Search in Annotations

To quickly locate annotations that correspond to UI elements, search through the annotation modeler stack of your selected target.

Context

When extending applications, you want to locate the relevant annotations quickly.

For example, you want to edit a *Technical Data* section on the application UI.

Therefore, in annotation modeler, you can search for matching texts in the following columns:

- *Node*
- *Edit Qualifier*
- *Value*

Annotation modeler searches through the entire annotation stack for the given OData data source, that is, in both local and external annotations.

Procedure

1. Choose the *Select Target* dialog and select the targets you want to search through. For more information, see Select Targets [page 1206].
2. In the search field, enter the text you want to search for, and press ENTER.

   For example, enter **`Technical Data`**, and press ENTER.

   The number of matches is displayed next to the search field.

   In the table, the first search result is selected automatically. Further search results are highlighted.
3. Edit the node as required.

15.3.7.5.5.2.4 Change the Order of Annotations and Collection Items

Context

In annotation modeler, you can change the sequence of collection items. For example, you can change the sequence of *UI.DataFields* within a *UI.LineItem* to change the order of the columns in your application.

You can also re-order local annotations, for example, to structure your annotations as required. The changed order of the annotations is reflected in the code of the XML file and can also be seen in the code editor. The order of the items on the UI is not affected.

Procedure

1. In a local annotation node, select the row you want to move.
2. To move the row one position up or down, choose *Move up* or *Move down*.
   - With each click, the row moves one position.

- The row remains selected, so you can click several times to move the row multiple positions.

15.3.7.5.5.2.5 Create Internationalized Labels

Prerequisites

For language-dependent texts that appear in the application, such as labels, the *String (i18n)* expression type is defined in the vocabulary for applying internationalization (i18n).

Context

When defining texts as *String (i18n)* expression type, the text is automatically added to the `i18n.properties` file of your project and prepared for translation into other languages.

In addition, the internationalization icon is displayed in the *Value* column and you can edit the values for internationalization.

Procedure

1. Choose the *String (i18n)* expression type and enter a text in the *Value* column.

   The i18n key and other text properties are automatically generated and stored in the `i18n.properties` file of your project.
2. To change the automatically generated text properties (except for the text key), in the *Value* column, choose the *i18n Details* icon.

   The *i18n Details* dialog box opens.
3. Edit the i18n details as required. The available text types are read from the `i18n.properties` file that is defined in the `manifest.json` file.
4. Change the text key, if required, in the `i18n.properties` file.

Related Information

Translatable Texts

15.3.7.5.5.2.6 Concatenate Values

To create complex expressions, you can concatenate several properties of type *Value*. You can use delimiters to separate concatenated values.

Context

Concatenated paths and values are displayed in the *Key Information* column.

Procedure

1. In the *Value* row, in the *Actions* column, choose the *Add* icon.

   The *Add To Value* dialog box opens.
2. Choose *Component* and choose *OK*.

   The previous value is inserted as first component. The component is inserted as second component.
3. Edit the *Expression Type* and *Value* columns as required.
4. To concatenate more values, add another component.
5. To set delimiters, add another component and enter the delimiter string. Use the *String (i18n)* expression type for delimiters that are internationalized. Use the *String* expression type for delimiters such as space or semicolon+space.

Related Information

Binding Paths: Accessing Properties in Hierarchically Structured Models

15.3.7.5.5.2.7 Use Parameters in URLs

To make a URL more flexible, you can insert parameters in any part of a URL.

Context

You can use parameters in properties that can express URLs, for example, in *IconUrl*.

For each parameter you insert, you can then define flexible targets.

Procedure

1. In the property row, enter the URL as *String* expression type. For each parameter, use curly braces {}, for example, **`www.{SLD}test.{TLD}`**.
2. Press `Enter`.

   For each parameter, a new node is created.
3. For each parameter, in the *Value* column, enter the target of the URL parameter.

15.3.7.5.5.2.8 Edit External Annotations

Annotations defined in the metadata or in other external sources cannot be edited directly but can be overridden by cloning them to local annotations. When you clone an annotation from external sources to the local file, the clone is inserted in the currently open annotation file where you can edit it.

Prerequisites

Verify that the `manifest.json` file contains the *dataSources* object with an array of one or more *ODataAnnotation* types. See Prerequisites for Working with Annotation Modeler [page 1200].

> **i** Note
>
> Each of these objects has the *uri* and *localUri* properties. When you open the annotation file in annotation modeler, it loads all annotations that are defined in the *localUri*.

Context

Annotations can only be cloned to the currently open local annotation file that uses the same OData data source as the external annotation file that includes the original annotation.

Procedure

1. Select the targets for which you want to edit external annotations. For more information, see Select Targets [page 1206].
2. Expand the *External Annotations* node under the target and navigate to the external annotation you want to override.
3. From the *Actions* column, choose *Clone for overriding*.

Annotation modeler clones the external annotation from the remote annotation source to the local annotation file that is currently open in annotation modeler. The cloned term and its properties are selected. Now you can edit the clone in the local annotation, see Edit Annotations [page 1204].

As each of the sources now contain a different value for the same term, you have to be aware of which of the values will affect the UI:

- In the *External Annotation* section, an overridden annotation and its child nodes are displayed in gray and struck through and marked as *(overridden)*.

  > **i Note**
  >
  > Individual collection records, such as *UI.DataField* of *UI.LineItem* can only be cloned with the whole collection. In this case, choose the *Clone for overriding* icon for the parent term to clone the record together with the whole collection. As soon as the collection is cloned, all its records are marked as overridden. In your local annotation file, you can then delete the collection records that you do not need, or you can change their properties as required.

- If you change the qualifier of a cloned annotation, it is treated like a new one and is used along with the initial one. For example, if you clone the *UI.LineItem* annotation and change its qualifier, both *UI.LineItem* entities are active.
- When you delete a clone of an external annotation from the local annotations, the external annotation is no longer overridden and affects the UI.
- If your project supports previewing the changes by using the *Run* function in SAP Web IDE, you can also run the component and check which annotation affects the UI.

  > **i Note**
  >
  > Terms from different annotation files originating from the same OData data source override each other in the same sequence as specified in the `manifest.json` file. The last annotation specified in the `manifest.json` file is the active one.

15.3.7.5.5.3 Validation of the Annotation File

Annotation modeler checks your `annotation.xml` file for inconsistencies in vocabulary specifications and other issues that can cause an unstable UI.

> **i Note**
>
> - When you add new annotations to your `annotation.xml` file, annotation modeler validates their property values as soon as you leave the field after editing.
> - To validate the entire `annotation.xml` file, save your project.

Related Information

Warnings and Errors in Annotation Modeler [page 1218]
Check Error Descriptions in the Problems View [page 1218]

15.3.7.5.5.3.1 Warnings and Errors in Annotation Modeler

Annotation modeler validates the local annotation file based on the vocabulary definitions and OData standards. Error and warning messages notify you about incompatibilities during the development process and support you in resolving them.

- When you save a project, the local annotation file is validated.
- When you leave a field, the field is validated.
- If the validation process discovers an error, such as an empty value for mandatory property or wrong input format, error and warning statuses are updated: the fields are highlighted in red, and the nodes that contain them are marked with red vertical bars.
- Errors and warnings are propagated to the parent elements in the tree structure. If there are errors and warnings in the child nodes, the parent node is marked with a red vertical bar.

> → Tip
>
> To obtain an overview of all errors and warnings in your local annotation file, open the *Problems* view in SAP Web IDE.

15.3.7.5.5.3.2 Check Error Descriptions in the Problems View

In the *Problems* view, you can check your `annotations.xml` file for issues resulting from validation checks run by the annotation modeler. This includes inconsistencies regarding vocabulary specifications and other issues that can cause an unstable UI.

Procedure

1. To check the `annotations.xml` file for issues specific to the annotation modeler, open the `annotations.xml` file in the annotation modeler.

   > ℹ Note
   >
   > The *Problems* view displays issues specific to the annotation modeler only if the `annotations.xml` file is opened in the annotation modeler.

2. Open the *Problems* view. See Using the Problems View [page 1251].

   To navigate to errors in annotations of another entity type, switch to the entity type before clicking on the navigation link in the *Problems* view.

15.3.7.5.6 Troubleshooting

Error Message	Issue Description and Solution
The file `<file name>` cannot be edited, because it is not registered in the manifest.json file (app descriptor).	• **The local annotation file cannot be opened.** The annotation file cannot be loaded because it is not registered in the app descriptor (`manifest.json` file) of this project. This happens if the annotation file has not been added to the project in the standard way (using the project wizard, *New Annotation File Settings* or *Import Annotation File*). Please define the annotation file for an existing data source in the `manifest.json` file. • **The external annotation file cannot be opened.** External annotation files, such as those added ,in the project creation wizard, cannot be opened in annotation modeler because they are not supposed to be used productively. They are copies of the back-end annotations of the OData service. Changes in this file are not loaded when you run the application, because the application uses the external version of this local copy. To modify external annotations, clone them to your local annotation file to edit them. For more information, see Edit External Annotations [page 1216]. If your project does not have a local annotation file, create a new local annotation file. For more information, see Create New Annotation Files [page 1203]. If you need to use this file productively, add it to your project by choosing *Import* > *Annotation File* from the context menu.
The annotation xml file cannot be loaded because it contains xml format errors. Open the Problems view to view and fix the errors.	The xml file is invalid due to xml format errors and cannot be opened in annotation modeler. Open the *Problems* view to view and fix the errors. Then, reopen the annotation file in annotation modeler.
Invalid file: DataServices definition is missing.	Check that the annotation xml file includes an `/edmx:DataServices/Schema` node. You can create a valid file by creating an annotation file. For more information, see Create New Annotation Files [page 1203].
There is no OData data source defined in the manifest.json. Please define an OData data source.	Check the relevant entry of the dataSources setting of type OData in the `manifest.json` file (app descriptor).

15.3.7.6 Run and Test HTML5 Modules

You can run and test your HTML5 modules using the tools provided by SAP Web IDE and the browser.

Prerequisites

Before running a module, make sure that:

- All the relevant dependencies of the module are defined in the MTA descriptor (mta.yaml).
- All the modules and resources on which the module is dependent are implemented and available .

> **i Note**
>
> You don't need to build your module explicitly before running, because it will be built automatically during the run process.

Running a Module

To run an HTML5 module, from the module context menu, choose *Run*, and one of the following options:

Option	Description
Run Configurations...	Create a new or modify an existing run configuration. For more information, see Configure How to Run HTML5 Modules [page 1221].
Run as > *Web Application*	Run the module as a Web application starting with its HTML file.
Run as > *SAP Fiori Component on Sandbox*	Run the module in an SAP Fiori launchpad sandbox environment.

> **i Note**
>
> When you run a module for the first time, it is pushed to XS Advanced and bound to the required XS Advanced services that are defined in the MTA descriptor, which can take some time. After this, the run starts more quickly. If you modify only the source files, or `xs-app.json` in HTML5 modules, the module remains in XS Advanced, and the changes are injected directly into the running module.

Use the Run Console

Progress and status messages, generated during the run process, as well as the log records, are displayed in the Run console. The console opens automatically in the main browser tab. You can show and hide the console by clicking the *(run console)* icon in the bottom pane's toolbar.

The left pane of the console displays a list of all running modules in the current MTA project along with the corresponding run configurations. Each entry is accompanied by a status icon with a tooltip. Click the run configuration name below a module name to view its most recent run log.

The controls displayed at the top of the console enable you to perform the following tasks:

- If the application URL is visible, click it to access the running application in the browser.
- To view the detailed server log of a run process, click *Logs*.
- To clear the log in the console, click *(clear the log)* .
- To stop running the module, click *(stop)* .
  This causes the module to be removed from the runtime system, so that if you run the module again, the run process starts from the beginning.
- To run the module using a different configuration, select the configuration and choose the *(run)* button in the task bar.

Debugging a Module

Debug the module using the browser developer tools.

15.3.7.6.1 Configure How to Run HTML5 Modules

You can configure how to run HTML5 modules in your project.

Procedure

1. From the context menu of an HTML5 module in your project, choose *Run* > *Run Configurations* .
2. To create a new run configuration for a module, click + and the type of run configuration, *Web Application* or *SAP Fiori Component in Sandbox*.
   A new configuration with a default name appears under the selected category in the left-side panel.
3. To edit a newly created or existing run configuration, click its name in the panel, and edit its properties as required.

Field	Description
General tab	
Name	Modify the default name.
Run Application File	If the type is *Web Application*, then enter the name of the main HTML file of your module.

Field	Description
	If the type is *SAP Fiori Component in Sandbox*, then enter the SAP Fiori component to start the application, such as the *Component.js* file.
Preview Mode	Select how to open the application: • *Without Frame*: in a full window (default) • *With Frame*: in a frame with configurable viewing options
Mock Data	Select *Run with mock data* to use mock data in your application. If you use mock data for your application, make sure that: • You have configured settings for the mock server. For more information, see Configure Mock Data Usage [page 1237]. • The application that you want to run uses the SAPUI5 OData model with **JSON** format.
URL Components tab	
URL Parameters	You can define additional URL parameters as name-value pairs to pass to the application. 1. To add a new row for an additional parameter, choose *Add Parameter*. 2. In the new row, enter a parameter name and its value.
URL Hash Fragment	Enter the URL hash fragment (also known as a fragment identifier) without the hash delimiter (#) .

4. To save the configuration and run your module using this configuration, choose *Save and Run*. To save all configurations without running, choose *OK*. To discard all your changes, choose *Cancel*.

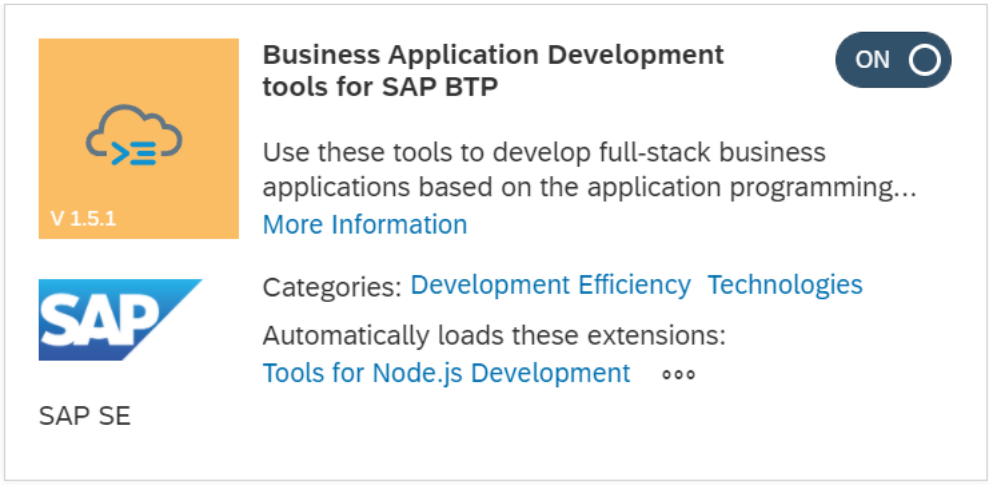
   These settings are persistent and are used every time that you run your application with the selected run configuration, until you edit or delete the settings.

15.3.8 Developing Business Applications

SAP Web IDE enables you to develop business applications using the application programming model for SAP BTP.

The Business Application Development tools for SAP BTP contain a new project template, and help you implement data models, services, and UIs to develop your own standalone business applications or extend other cloud solutions, such as SAP S/4 HANA.

To use this extension, you need to enable the tile in the *Preferences* perspective on the *Extensions* page.



For more information, see Business Applications in the SAP BTP documentation.

15.3.9 Developing SAP Fiori Launchpad Modules

SAP Fiori launchpad modules are used to create a SAP Fiori launchpad site that runs on the XS Advanced Model.

For information about the runtime capabilities of SAP Fiori launchpad, see the *User Guide* under SAP Fiori Launchpad.

Related Information

Creating a New Launchpad Site [page 1223]
Adding an App to an Existing Launchpad Site [page 1230]
Roles and Authorizations [page 1233]

15.3.9.1 Creating a New Launchpad Site

Create a Multi-Target Application (MTA) project to contain your launchpad site.

Prerequisites

In SAP Web IDE *Features* screen, make sure that the *SAP Fiori Launchpad Site* feature is enabled. If the feature is not enabled, the item *SAP Fiori Launchpad Site Modules* will not be displayed in the project context menu.

Context

The following modules are created:

- `MySite` – the Web entry point to the SAP Fiori launchpad. This module contains an `xs-app.json` file, which is configured like an approuter component. For more information, see Configure the XS Advanced Application Router.
- `MySite-Content` – the configuration files of the site, which contain a sample group and sample tiles/apps.

Procedure

1. In SAP Web IDE, create a new Multi-Target Application project.
2. Select a space for it. For more information, see Setting Up Application Projects.
3. From the context menu of the project root folder, select *New* > *SAP Fiori Launchpad Site Modules*.
4. In the wizard, enter names for the site's module, for example, **`MySite`**, and the site's content module (optional). The default name in this example is **`MySite-Content`**. Click *Finish*.

   The following modules are created: `MySite` and `MySite-Content`.
5. Configure the launchpad content: groups, catalogs, and apps.
6. Right-click the multi-target application root folder, select *Build*, to build an `.mtar` file for your project.
7. Download the `.mtar` file from `mta_archives`, by selecting *Export* from the file's context menu.
8. Use the SAP HANA XS Advanced Model command line tool to deploy this file to SAP HANA XS Advanced Model server. At the end of the deployment process, you receive the URL of the launchpad site in the command line tool.
9. Run the URL in your browser.

   You have now created a launchpad site that runs on SAP HANA XS Advanced Model with sample out-of-the-box content.

Related Information

Add an SAPUI5 App to the Launchpad Site [page 1225]
Add Content to the Launchpad Site [page 1226]
Assign Apps to Scopes [page 1227]
Apply Translations to Your Launchpad Site [page 1227]
Apply a Custom Theme to Your Launchpad Site [page 1229]

15.3.9.1.1 Add an SAPUI5 App to the Launchpad Site

Add a custom SAPUI5 app to the launchpad.

Prerequisites

You already have a SAPUI5 application that you want to add to the launchpad site.

Procedure

1. Copy your SAPUI5 application resources under the site's module ('MySite').
2. To the `xs-app.json` file, add the route to the application's static resources. For example:

   Sample Code

   ```
   {
   "source": "^/app1/sap/demo/(.*)",
   "target": "$1",
   "localDir": "app1/sap/demo/",
   "cacheControl": "public, max-age=31536000,must-revalidate"
   }
   ```

3. Under the `applications` folder, add an application configuration to your site as follows:
   a. Create a folder under the `applications` folder. For example, `MySAPUI5App`.
   b. Copy the `manifest.json` file of the application to the new folder. For example:

      Sample Code

      ```
      <site-content-deployer-folder-name>/
      | - package.json

      | - applications/

      |   \ - MySAPUI5App/
      |       | - manifest.json
      |       \ - i18n/

      \ - src/
          | - site-content.json
          \ - i18n/
      <MySite>/
      ```

   c. Add to the `manifest.json` the location of your SAPUI5 component. For example:

      Sample Code

      ```
      ...
      "sap.platform.cf": {
      "uri": "<Location of your SAPUI5 component under your MySite module>"
      }
      ```

```
...
```

Related Information

Add Content to the Launchpad Site [page 1226]

15.3.9.1.2 Add Content to the Launchpad Site

You can add a group to the launchpad, assign it to the Everyone role, and add a tile to the group.

Procedure

1. In the `site-content.json`, add to the `groups` property, a group with the same structure as the *sample* group. Under the `identification.id` property, provide the new group with an id that is different from the id of the sample group.
2. To enable all users to see this group, assign it to the *Everyone* role, by adding the group id to the `id` property. For example:

   Sample Code

   ```
   "roles": {
      "Everyone": {
         "catalogs": [],
         "groups": [{
            "id": "<id>"
         }]
      }
   },
   ```

3. To add a tile to the group, add a new tile to the array of tiles under the group's `tiles` property.

   > **Note**
   >
   > The link between the tile in the group and the app to which it navigates, is defined using the `semanticObject` and `action` properties, and the appId, which is a mandatory parameter that represents the application ID in the `manifest.json` file.

4. To see the results in runtime, build, deploy, and run your launchpad site again.

Related Information

Assign Apps to Scopes [page 1227]

15.3.9.1.3 Assign Apps to Scopes

You need to assign an app to scopes in the `site-content.json`, to determine which users see this app tile. Otherwise, all users will be able to see this tile/app.

To assign an app to a scope, add the `oAuthScopes` property to the `manifest.json` of your app. For example:

Sample Code

```
...
    "sap.platform.cf": {
      "oAuthScopes": ["$XSAPPNAME.Display", "$XSAPPNAME.Create"]
      } ...
```

Only users assigned to at least one of these scopes will see the tile/app.

Note

The scopes you assign must be the scopes in the `xs-security.json` file, which contains the configuration for the User Authentication and Authorization system (UAA). For more information, see The Application Security Descriptor [page 996].

15.3.9.1.4 Apply Translations to Your Launchpad Site

You can apply translations to your launchpad site. Translations are handled according to SAPUI5 conventions.

Context

To apply translations, you need to perform the following tasks, for both the `manifest.json` file (in the `applications` folder), and also for the group, catalog, and application entities in the `site-content.json` file.

For more information about the `manifest.json` file, see the SAPUI5 Toolkit: Descriptor for Applications, Components, and Libraries.

Procedure

1. Modify the properties of the entity to be translated, so that the value of the property is in the SAPUI5 format. For example:

   Sample Code

   ```
   {
       {"identification" :
           "title" :  "{{<key_for_title_in_property_file>}}"
   ```

```
    }
}
```

> **Note**
>
> Only properties defined by the spec as translatable are translated.

2. Modify the `i18n` property of the entity, in the `manifest.json` file, and in the `identification` sections of the group, catalog and application entities in the `site-content.json` file, to point to the master language properties file.

> **Note**
>
> The master language properties file must be located in the same folder as the `manifest.json` or `site-content.json`, or in one of their subfolders.
>
> Backward relativity is not supported. For example, this is not supported:
>
> `"i18n" : "../../otherFolder/bundle.properties"`

The following is an example of a group:

> **Sample Code**

```
{
    "_version": "1.0",
    "identification": {
        "id": "123",
        "namespace" : "",
        "title": "{{group_title}}",
        "entityType": "group",
        "i18n": "i18n/MyBundle.properties"
    },
    "payload": {…
```

3. Create property files for translation:
   a. For the master language, provide any file name with the `.properties` extension. For example, `MyBundle.properties`:

   > **Sample Code**

   ```
   <key_for_title_in_property_file> = My Group Title
   ```

   b. For other languages, provide a file name that ends with the _`<locale>` string. For example, MyBundle_fr.properties:

   > **Sample Code**

   ```
   <key_for_title_in_property_file> = My Group Title In French
   ```

15.3.9.1.5 Apply a Custom Theme to Your Launchpad Site

You can apply a custom theme to your launchpad site.

Prerequisites

- You have created a custom theme in the SAP UI theme designer tool.
- You have downloaded the custom theme `zip` file to your computer, and unzipped it.

Procedure

1. In your SAP Web IDE project, under the site module, *MySite* for example, under the `themes` folder, place the folder that contains the unzipped custom theme files.
2. In the `site-content.json` file, under the site's `payload` section, `"sap.cloud.portal"`, add the name of the custom theme that is active, and the list of themes from which the user can select a theme. For example:

   Sample Code

   ```
   "sap.cloud.portal": {
      "config": {
         "theme.id": "my_company_theme",
         "theme.active": "[\"sap_belize\",\"sap_hcb\",\"my_company_theme\"]"
      }
   }
   ```

   In addition, under "siteThemes", below the default themes, add an entry for each custom theme. For example:

   Sample Code

   ```
   "siteThemes": {
      "sap_hcb": {
         "description": "SAPHighContrastBlack",
         "name": "sap_hcb",
         "path": "sap_hcb"
      },
      "sap_belize": {
         "description": "SAPBelizel",
         "name": "sap_belize",
         "path": "sap_belize"
      },
      "my_company_theme": {
         "description": "MyCompanyTheme",
         "name": "my_company_theme",
         "path": "my_company_theme"
      }
   }
   ```

3. In the `xs-app.json` file, add the route to the `themes` folder. For example:

Sample Code

```
{
   "source": "/themes/(.*)",
   "target": "$1",
   "localDir": "themes"
}
```

15.3.9.2 Adding an App to an Existing Launchpad Site

You can add an app to an existing launchpad site.

Prerequisites

- You have a project in SAP Web IDE that contains a SAP Fiori app or a SAPUI5 app
- You have the site ID of the SAP Fiori Launchpad site to which you want to add the SAP Fiori app
- If your SAP Fiori app is configured for a User Authentication and Authorization system (UAA) that is different from the one used by the site, you need to assign your users with permissions for this UAA

Context

The workflow involves the following steps:

1. Add the `site-content.json` file to your app project.
   This file defines the group in which the app's tile will appear, and the semantic object and action properties that are used to identify the app's tile. The same semanticObject and action pair needs to be configured also in the app's `manifest.json` file.
   You can also assign the app to a catalog in this file, using the application ID.
2. In the `manifest.json` file, you also define the scopes required for viewing the app's tile. These same scopes must be defined in the `xs-security.json` of the relevant UAA as well.
3. Configure the `package.json` file to use the site-app-server which is required for running your SAP Fiori application in the SAP Fiori launchpad.
4. In the `mta.yaml` file, you add the binding to the portal service and specify the site ID of the launchpad site to which to add the app.

The following sections provide codes samples for each step.

Procedure

1. Add the `site-content.json` file to your app project. In the following sample, you see how to assign the group, MyGroup, to the predefined role Everyone, which represents all logged-in users.

   Sample Code

   ```
   {
     "roles": {
      "Everyone": {
        "_version": "1.0",
        "identification": {
               "id": "Everyone",
               "namespace": "",
               "entityType": "role"
        },
        "payload": {
               "catalogs": [],
               "groups": [{
                              "id": "MyGroupId"
                }]
        }
      }
     },
     "applications": {
     },
     "groups": {
      " MyGroupId": {
        "identification": {
         "id": " MyGroupId",
         "i18n": "i18n/i18n.properties",
         "namespace": "",
         "title": "{{ MyGroupId_GROUP_TITLE}}"
        },
        "payload": {
         "tiles": [{
           "id": "SampleTile",
           "subTitle": "{{SampleTile_SUB_TITLE}}",
           "appId": "MyUniqueAppID",
           "target": {
            "semanticObject": "myApp",
            "action": "show",
            "parameters": []
           }
         }]
        }
      }
     }
   ```

2. In the `manifest.json` file, which is the SAPUI5 application descriptor, you define the scopes that determine the users who can see this app tile.

   Sample Code

   ```
   {
     …
     "sap.app": {
       "_version": "1.3.0",
       "id": "MyUniqueAppID",
       "title": "My App",
      …
       "crossNavigation": {
         "inbounds": {
   ```

```
        "appShow": {
          "title": "{{TITLE}}",
          "icon": "sap-icon://person-placeholder",
          "semanticObject": "myApp",
          "action": "show",
}
}
}
},
...
        "sap.ui5": {
             "componentName": “<SAPUI5 component name>"
           },
...
  "sap.platform.cf": {
      "uri": "/resources/webapp",
      "oAuthScopes": ["$XSAPPNAME.Manager"]
  }
}
```

The same scopes must also be defined in the `xs-security.json` of the relevant UAA:

Sample Code

```
...
{
  "xsappname": "backend",
  "scopes": [
    {
      "name": "$XSAPPNAME.Manager",
      "description": "Manager"
    },
{
      "name": "$XSAPPNAME.User",
      "description": "User"
    }
  ],
  "role-templates": [
    {
      "name": "MyAppRoleTemplate",
      "description": "Sample role template",
      "scope-references": [
        "$XSAPPNAME.Manager"
      ]
    }
  ]
}
```

3. Configure the `package.json` file to use the `site-app-server`, which is required for running your SAP Fiori application in the SAP Fiori launchpad:

Sample Code

```
"dependencies": {
  "@sap/site-app-server": "^1.9.1"
},
"scripts": {
  "start": "node node_modules/@sap/site-app-server/index.js"
}
```

4. In the `mta.yaml` file, you add the binding to the portal service and specify the site ID of the launchpad site to which to add the app:

Sample Code

```
- name: sample-app
  type: javascript.nodejs
  path: sample-app/
  parameters:
    memory: 32M
  requires:
     - name: MySampleBackendApp
       group: destinations
       properties:
         name: MySampleBackendApp
         url: ~{backendurl}
         forwardAuthToken: true
     - name: sap-portal-services-sample-app-content
     - name: MySampleBackendAppUAA
- name: backend
        type: javascript.nodejs
        path: backend/
        requires:
          - name: independent-backend-uaa
        provides:
         - name: backend
           properties:
            backendurl: ${default-url}
        parameters:
          memory: 32M
resources:
  - name: sap-portal-services-sample-app-content
    type: com.sap.portal.site-content
    parameters:
      config:
        siteId: TargetSiteId
 - name:  independent-backend-uaa
   type: com.sap.xs.uaa-space
   parameters:
      path: security/xs-security.json
```

15.3.9.3 Roles and Authorizations

The application developer defines roles and authorizations by providing an `xs-security.json` file that contains a role template, one or more scopes, and attributes for each application.

Terminology

Term	Description
Scopes	A list of the authorization scopes define in the application's `xs-security.json` file.

Term	Description
Attributes	A list of the attributes defined in the application's `xs-security.json` file, which are applied to a selected role. Depending on the value of the attributes defined, access to resources is either granted or restricted. For example, in a sales scenario, the attribute region=emea could be used to restricts access to the sales orders for the geographical region "EMEA".
Role Templates	A list of the role templates defined in the application's `xs-security.json` file. A role template defines the type of access permitted for an application. For example, the authorization scope, and any attributes that need to be applied.
Role	An instance of a role template. You build a role based on a role template and assign the role to a role collection.
Role Collection	Roles are assigned to role collections, which, in turn, are assigned, to SAP HANA users or SAML2 groups.

Guidelines

- Roles are reflected at the application level.
- Catalogs are relevant for personalization only. They are used to add more apps to the launchpad.
- Catalogs are assigned by default to the *Everyone* role and are not be filtered by roles.
- Applications are filtered by scopes. Each application can have one or more scopes assigned. An application can declare several permissions such as read, write, or a semantic type such as payment view.
- Groups are filtered by attributes.

15.3.10 Packaging and Deploying Applications

At the last stage of multi-target application development, you need to package your application and deploy it to the Cloud Foundry environment and to the SAP HANA XS Advanced Model (XS Advanced) production system.

Prerequisites

All the modules in your application are implemented and tested, and the MTA descriptor contains all the correct definitions and dependencies.

Procedure

1. From your project context menu, choose *Build* > *Build with Cloud MTA Build Tool*. The resulting MTA archive file (`<ID>_<version>.mtar` file) is generated within the project's *mta_archives* folder.
2. Right-click the `<ID>_<version>.mtar` file, and choose *Deploy* > *To Cloud Foundry* or *Deploy* > *To XS Advanced*.
3. In the dialog box that opens, select the space in the target system where you want to deploy your application.
4. If your project contains MTA extension files, select the relevant checkboxes from the dropdown list.
5. Click *Deploy*.

15.4 Customizing Your Project

Customize the developer experience for your SAP Web IDE project.

Any project customization you make in one session persists to the next. Customizations can vary from project to project.

15.4.1 Customize JavaScript Beautifier Properties

Beautify JavaScript files to reformat the source code to make it more readable.

Context

The beautifier configuration applies to all JavaScript, `.js`, `.json`, `.xsjs`, and `.xsjslib` files in your project.

Procedure

1. From the context menu of any file in your project, select *Project Settings* > *Beautifier* > *JavaScript*.
2. Determine how you want lines to be broken:

Option	Description
Break lines on chained methods	Select this option to add a line break when you add a chained method. By default, this option is cleared.

Option	Description
New lines	Set the maximum number of new lines between tokens. Choose from: *No New Lines*, *1*, *2*, *5*, *10*, *Unlimited New Lines*. For example, if there are two `\n`s between tokens, but you set this value to *1*, the second `\n` is removed. The default is 2.
Wrapping	Determine the number of characters that triggers a line wrap. A line break is inserted before the first word that reaches the limit that you set. Choose from *Do not wrap lines*, *40*, *70*, *80*, *110*, *120*, or *140* (default).

3. Determine how you want indents to be handled:

Option	Description
Keep array indentation	Select this option to keep the indentation of arrays as is. Clear this option to remove array indentation. By default, this option is cleared.
Indent with	Select the length of each code indent: *a Tab* (default), *2 Spaces*, *3 Spaces*, *4 Spaces*, or *8 Spaces*.

4. Determine additional code formatting options:

Option	Description
Space before conditional	Select this option to insert spaces before conditional `if/then` statements. By default, this option is selected.
Unescape printable chars encoded	Select this option to decode printable characters that are encoded. By default, this option is cleared.
Braces	Determine the positioning of braces: *Braces with control statement* (default), *Braces on own line*, or *End braces on own line*.

5. Choose *OK*.
   You can now beautify any open JavaScript file according to these customized properties, by choosing *Beautify* from the context menu.

15.4.2 Customize Code Checking Rules

SAP Web IDE provides validators to check your code. You can customize code checking for each project.

You can customize and use the SAP Web IDE JavaScript validators and their rule configurations, or you can customize code checking for your project:

- Customize the SAP Web IDE JavaScript validator configuration with your own settings. For more information, see Customize JavaScript Validator Configuration [page 1250].
- Configure your own JavaScript code checking rules and use them for your project. For more information, see Create JavaScript Code Checking Rules [page 1249].
- Create your own code checking plugin, then choose the plugin in the corresponding code checking configuration pane.

15.4.3 Configure Mock Data Usage

Configure settings to run an application using a client mock server.

Context

> **i Note**
>
> If your application uses the application descriptor, the *Root URI* and the *Metadata File Path* are taken from the file and are not editable in the settings. If you need to make changes, they should be made in the application descriptor file.

Procedure

1. From the project context menu, select *Project Settings*.
2. From the *Project Settings* options, select *Mock Data*.
3. In the *Root URI* field, enter the URL of the service used in your project.
   - If you selected a service when you created your project, SAP Web IDE automatically populates this field.
   - If the service URL changes, edit this field accordingly.
   - If there is no service URL, leave this field empty.
4. In the *Metadata File Path* field, enter the path to the service `metadata xml` file to fetch mock data to use in your project.

   > **i Note**
   >
   > If you do not specify the path, SAP Web IDE looks for the mock data in the `model/metadata.xml` file. If this path changes, edit this field accordingly.
5. Select one of the following options as your *Mock Data Source*:
   - Generated data (default)
   - JSON files
6. If you have provided an additional file containing custom mock requests extending the mock server, do the following:
   a. Select the *Add custom mock requests* checkbox.
   b. In the *Extension File Path* field, enter the path to the `mockRequests.js` file.
7. *Save* your changes.

15.5 Using Code Editors

SAP Web IDE provides customizable editors in which you edit the code for your applications.

There are various editors to support different types of files, including:

- JavaScript
- XML
- xsodata
- CDS
- hdbprocedure

When you open a file, it is displayed in the appropriate editor for the file type. All these editors support code completion.

15.5.1 Configure the Code Editor

Define the appearance and behavior of the code editor, and whether to autosave all changes in SAP Web IDE.

Procedure

1. In the left sidebar, choose (Preferences), then choose *Code Editor*.
2. Customize the appearance, input behavior, or save options.

   Appearance

Option	Description
Code Editor Theme	The theme of the code editor determines the background color and color of text.
Font	The monospace font for text in the code editor
Font Size	The font size of text in the code editor.
Code Folding	Hides code from a marked begin point to a marked end point or to the end of the file if end points are not used. Click in the gutter to mark begin and end points.
Full line selection	Line selection extends to the end of the line instead of to the end of the text in the line.
Highlight selected word	Highlights all occurrences of the word at the cursor position.
Show invisible characters	Shows white-space characters such as spaces, tabs, and new lines.

   Input Behavior

Option	Description
Auto-pair characters	Auto-pairs characters, such as quotation marks, parentheses, brackets, and so on.
Use spaces for tab indentation	Uses spaces for tab indentation, also known as soft tabs.
New Line Mode	Lets you choose one of the following: • auto • windows • unix

Save Options

Option	Description
Automatically save changes in all open documents at preset intervals.	For more information, see Setting User Preferences [page 1065].
Automatically beautify the code of an active document on manual save.	For more information about beautifying JavaScript code, see Customize JavaScript Beautifier Properties [page 1235].

Test your settings in the preview area on the right side of the screen..

3. Change the *Autosave* setting, if required.
4. Click *Save*.

15.5.2 Working in the Code Editor

Use keyboard shortcuts and context menus to easily edit and navigate through your code and code comments.

Feature support depends on file types.

All edit actions are also available from the *Edit* menu.

> **i Note**
>
> ES6 is not supported in JavaScript files.

> **i Note**
>
> The following keyboard shortcuts are for Microsoft Windows. For Mac OS, see Keyboard Shortcuts [page 1067].

Basic Navigating and Editing

Action	Keyboard Shortcut	Context Menu
Move one tab to the right	`Alt` + `R`	-
Move one tab to the left	`Alt` + `Q`	-
Close all tabs to the right of the open tab		*Close Tabs to the Right*
Undo edit	`Ctrl` + `Z`	*Undo*
Redo edit	`Ctrl` + `Y`	*Redo*
Show and hide control characters, such as space, tab, newline, and paragraph	`Ctrl` + `I` (uppercase i)	-
Move a line up	`Alt` + `Up arrow`	-
Move a line down	`Alt` + `Down arrow`	-
Move to a specific line	`Ctrl` + `L`	*Go to Line*
Copy a line to the line above or below	`Alt` + `Shift` + `Up arrow` `Alt` + `Shift` + `Down arrow`	-
Expand the entire hierarchy of file elements	`Alt` + `Shift` + `F2`	Gutter context menu: *Expand All*
Collapse the entire hierarchy of file elements	`Ctrl` + `Alt` + `F2`	Gutter context menu: *Collapse All*
Search for a string	`Ctrl` + `F`	*Find*
Search and replace a string	`Ctrl` + `H`	*Find and Replace*

Commenting

Action	Keyboard Shortcut	Context Menu
Comment out a line and restore to code. Comment syntax is appended to the code fragment automatically as a line. Each comment line is prefixed with `//`. For example: `//Use the comment style for short comments.`	`Ctrl` + `/`	*Toggle Line Comment*

Comment out a block and restore to code.
 	
Comment syntax is appended to the code fragment as a block using `/*` and `*/` to wrap the comment block. For example:
 	
`/* This is a comment block. Use this block comment style when comments span multiple lines.*/`
 	
`//Use the comment style for short comments.`
	`Ctrl` + `Shift` + `/`	*Toggle Block Comment*

Flag JavaScript code with a TODO comment. A `//TODO` comment is added at the cursor location. If the line already contains a `//TODO` comment, the action is ignored.	`Ctrl` + `Alt` + `T`	*Add TODO Comment*

Beautify File Formatting

Beautify file formatting for JavaScript, JSON, XML, and CSS files using the context menu *Beautify* option or `Ctrl` + `Alt` + `B`.

- The XML beautifier formats code with line wrapping at 140 characters, and an indentation of one tab space. The beautifier is not configurable.
- The CSS beautifier formats indentation with one tab space and is not configurable.
- You can customize JavaScript beautifier settings for your project. For more information, see Customize JavaScript Beautifier Properties [page 1235].

Using Multiple Cursors

You can use multiple cursors to rename several variables at once or to insert the same text in multiple locations.

To add cursors in your file, press `Ctrl` and click at the required locations. You can then type text, which will appear in all the cursor locations.

To remove multiple cursors click anywhere in the file.

Refactoring

You can change the name of a JavaScript function or variable, by using the context menu *Refactor* or `Alt` + `J`. Enter a valid new name and click *Rename*, and all references to the function or variable are automatically updated.

Navigating from View to Controller

While editing an XML view, you can navigate to the view's controller by using the context menu *Open Controller*. The controller JavaScript file is opened in a new tab.

If you right-click the name of an event handler and select *Open Controller*, the controller is opened to the event handler function (if it is defined in the controller).

Managing i18n Strings

In the context menu of an XML or JavaScript file, select *Open i18n* to open the i18n file defined in the `manifest.json`.

You can easily create a new string without having to open the i18n file by selecting *Create i18n String*. Add the string and its key, and they are added to the i18n file.

To edit a string, right-click on the i18n binding (for example, `{i18n>fileType_title}`) and select *Edit i18n String*.

You can get code completion when you need to select a string to add to an XML or JavaScript file. Put your cursor inside quotation marks and press `Ctrl` + `Space`, and get a list of strings defined in your i18n properties files. The properties files must be defined as i18n models in your manifest.

15.5.3 Generate JSDoc Comment Snippets

You can generate a snippet for JavaScript function declaration that creates a template for documenting the function.

Context

The JSDoc comment snippet provides a template for documenting the function that you create. You can use code completion for variables within the snippet.

Procedure

1. Select a function, and press `Ctrl` + `Alt` + `J`, or choose *Generate JSDoc Comment* from the context menu.

   The snippet is displayed above the function code.
2. Enter information about the function variables in the corresponding placeholders in the comment.

15.5.4 Using Code Completion

The code completion feature assists you when you are writing your code by preventing typos and other common mistakes, and providing API reference information for SAPUI5 objects.

Context

Code completion is triggered in one of the following ways:

- Auto hint code completion for JavaScript and XML files
  Based on your cursor location, SAP Web IDE establishes the context and displays a constrained list of suggestions. Use the corresponding icon to visually identify the code type being completed (that is, XML or JavaScript). This option is configurable; see Enabling Code Completion [page 1246].
- Manual code completion
  Supported for various file types including:`i18` and `messagebundle` property files.
  - `i18` and `messagebundle`
  - JavaScript
  - XML
  - xsodata
  - CDS
  - hdbprocedure

No configuration is required.

Cross-file completion for JavaScript

The dependencies can be defined using a dependency declaration in either of the following ways: SAP Web IDE can provide suggestions for functions (methods) that are defined in other modules on which the source file depends. Suggestions include methods declared explicitly in the dependency file and generated methods for metadata properties, associations, aggregations, and events of SAPUI5 controls.

- Use `SAP Web IDE can provide suggestions for functions (methods) that are defined in jQuery.sap.require(<moduleId1>,<moduleId2>,…)`. The dependency (target) file should contain the module declaration `jQuery.sap.declare(<moduleId>)`.
- Use `sap.ui.define([<dependencyFile1>,<dependencyFile2>,…], function(d1,d2,…){ });` where `dependencyFile` is the relative file path in the current project or the logical path to the library module. The dependency (target) files should contain a module declaration using `sap.ui.define` or Asynchronous Module Definition (AMD).

> **Note**
>
> Suggestions can be provided only if the dependency file is visible in the current project

Code completion for SAPUI5

The code editor provides code completion suggestions for:

- Properties in SAPUI5 methods. For example, suggestions for `Opa5.createPageObjects`, include all the properties for `baseClass`, `namespace`, `actions`, and `assertions`.
- SAPUI5 static and instance methods when you use dependency declarations with Asynchronous Module Definition (AMD).
- SAPUI5 module dependencies in AMD. Proposed suggestions are for the element type of module according to the element name entered.

API reference information for SAPUI5 objects is shown in a tooltip when you hover over a suggestion. If more information is available, a *More Details* button is provided that links to the full SAPUI5 documentation for the current element.

Code completion for SAPUI5 is dependent on the SAPUI5 version selected in the project settings.

Procedure

To use auto hint code completion:

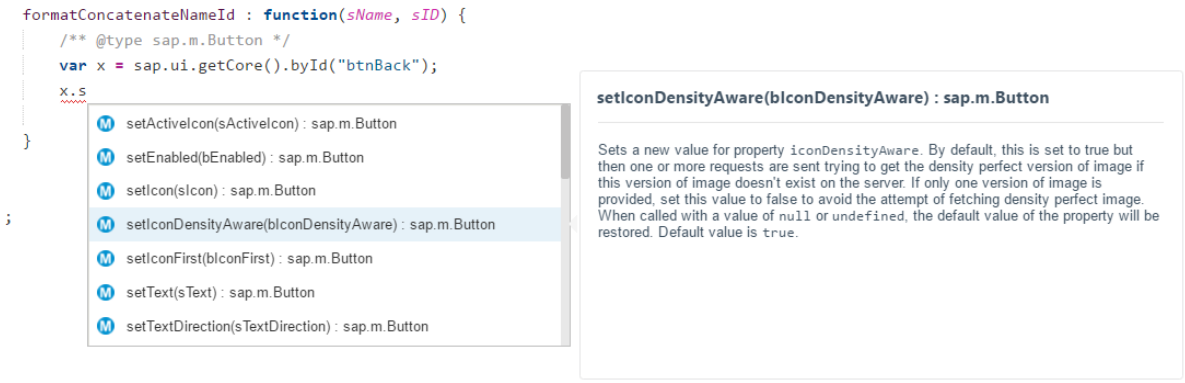
1. Create or open an XML or JavaScript file.
2. Place your cursor in the context for which you require assistance, and press `Ctrl` + `Space`.

   A list of suggestions is displayed. Deprecated APIs are indicated with strikethrough text. Hover over a suggestion and the tooltip also indicates whether an API is deprecated or experimental.
3. Scroll through the list, select the appropriate fragment, and press `Enter`.
   The code fragment is added at the cursor location.

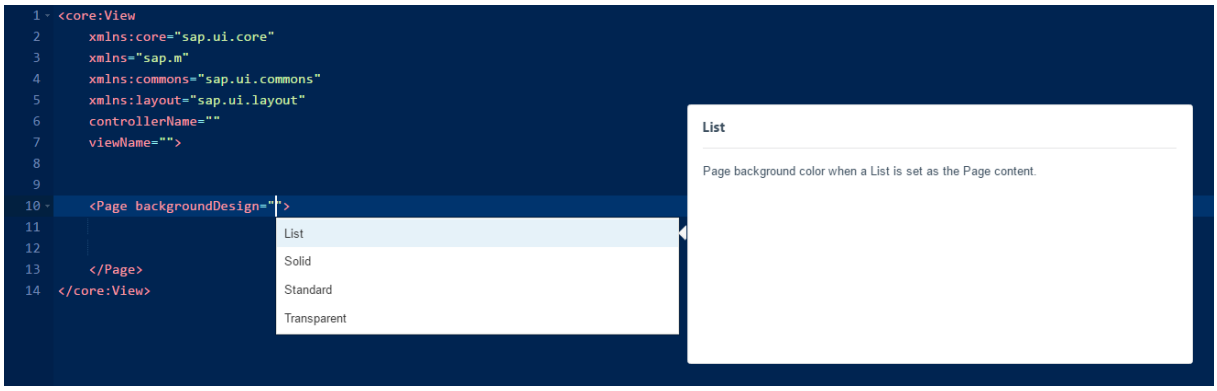
Example

This example of embedded type code completion shows how you can attach a type to your variable definition to enable presentation of appropriate code completion suggestions for the variable.

Define the variable type in a JSdoc comment before the variable definition, and then press `Ctrl` + `Space` for code assistance. When you hover over a suggestion (e.g., `setIconDensityAware`), API reference information is shown in a tooltip.



In the next example, to determine which values you can use for `<Page backgroundDesign="">`, move the cursor between the quotation marks as shown below, and launch code completion. The options for the attribute are listed. Hover over one of the options to get a tooltip with API reference information about that option.



15.5.4.1 Configure Code Completion

You can enable code completion as you type (auto hint) for JavaScript and XML files.

Procedure

1. To open the *Preferences* perspective, in the left sidebar, choose (Preferences).
2. Click *Code Completion* and select the *Enable* checkbox.
3. Click *Save*.

15.5.5 Checking Code

SAP Web IDE performs code checking, also known as validation, and displays errors as annotations.

By default, when you open a JavaScript, JSON, or XML file, code checking is triggered and the detected code issues are displayed as annotations within the editor. Understand how to configure and use code checking in your project.

Configure Code Checking [page 1247]
Configure when to trigger code checking, also known as code validation, and the level of messages to display.

Code Checking Annotations [page 1248]
If a syntax error is found during editing, the relevant lines are annotated with flags indicating the error severity. Understand the severity level of these flags so that when you open a file, you know how to interpret these annotations.

JavaScript Validation [page 1248]
Review the default JavaScript validator configuration. Customizations always override these defaults.

YAML Validation [page 1251]
A built-in client-side YAML validator is integrated with the code editor.

Using the Problems View [page 1251]
View information about problems in the projects in your workspace.

15.5.5.1 Configure Code Checking

Configure when to trigger code checking, also known as code validation, and the level of messages to display.

Context

By default, code checking is enabled when you make changes to your code, and all messages are displayed. You can change these defaults.

> **i** Note
>
> You can customize code checking rules for each project. For more information about customizing and using code checking, see Checking Code [page 1246].

Procedure

1. To open the *Preferences* perspective, in the left sidebar, choose (Preferences).
2. Select *Code Check*.
3. In the *Run Code Check* section, select when to display code checking annotations:
   - Choose *On Save* to display annotations only when you save your file.
   - Choose *On Change* to display annotations every time you make a change to your code.
4. In the *Code Check Level* section, select which messages to display:
   - *All* displays all errors, warnings, and information messages
   - *Error* displays error messages only
   - *Error and Warning* displays error and warning messages
   - *Disable* suppresses message display
5. Choose *Save*.

Task overview: Checking Code [page 1246]

Related Information

Code Checking Annotations [page 1248]
JavaScript Validation [page 1248]
YAML Validation [page 1251]
Using the Problems View [page 1251]

15.5.5.2 Code Checking Annotations

If a syntax error is found during editing, the relevant lines are annotated with flags indicating the error severity. Understand the severity level of these flags so that when you open a file, you know how to interpret these annotations.

- All syntax errors are annotated on code line and tab levels and each annotation is colored according to its severity.

Color	Severity
Red	Error
Yellow	Warning
Blue	Information

- When you hover over an annotation, a tooltip displays one or more detected issues and possible resolutions for the annotated line.
  The detected issues of an annotated line are categorized (for easy identification), identified, and described, so that you can determine how best to resolve an issue.

  Categories Used to classify the issue. For example, possible error, best practice, stylistic issue, and others.

  Rule IDs Used to define the logic for detection or list known issue exceptions. For example, *semi* is a rule ID for ESLint.

  Messages Detail the issue or suggest a possible resolution.

  You can resolve the issue and continue development with iterative fixes.

Parent topic: Checking Code [page 1246]

Related Information

Configure Code Checking [page 1247]
JavaScript Validation [page 1248]
YAML Validation [page 1251]
Using the Problems View [page 1251]

15.5.5.3 JavaScript Validation

Review the default JavaScript validator configuration. Customizations always override these defaults.

The SAP Web IDE default JavaScript validator uses ESLint code checking.

ESLint Rule Execution Defaults	See these resoures for additional support: • For ESLint configuration details, see Configuring ESLint . • For ESLint rule information, see Rules .
ESLint Rule Metadata Defaults	Attributes can have multiple supported values. Use the following: • `severity` attribute to define whether an issue renders as `error`, `warning` (default), or `information`. • `category` attribute for a better semantic classification: `possible error`, `best practice`, `stylistic issue`, and others. • `help` attribute (optional) to override the default help links listed.

Parent topic: Checking Code [page 1246]

Related Information

Configure Code Checking [page 1247]
Code Checking Annotations [page 1248]
YAML Validation [page 1251]
Using the Problems View [page 1251]

15.5.5.3.1 Create JavaScript Code Checking Rules

You can replace the SAP Web IDE default rule configurations in the JavaScript validator by configuring your own code checking rule configurations.

Context

The SAP Web IDE default JavaScript validator uses ESLint code checking. You can override the default rule configuration by creating a folder containing your custom rules.

Procedure

1. Choose *File* > *New* > *Folder*.
2. Name the new folder and add your customized code checking rules to this folder.

> **i Note**
>
> The rule files in the folder must be JavaScript files.

3. From the context menu of any file in your project, choose *Project Settings*.
4. From the *Project Settings* options, select *Code Checking* > *JavaScript*.
5. Next to the *Custom Rules Folder* field, choose *Browse*. A popup window displays the projects in the workspace.
6. Open your project and select the rules folder that you created.
7. Choose *OK*.

   Your custom rules appear in the *Rules* table instead of the default rules. All the rules are disabled by default.
8. To implement your custom code checking rules, in the *Rules* table, enable each one.
9. Choose *Save* to save the new rule configuration for the project.

15.5.5.3.2 Customize JavaScript Validator Configuration

You can customize the configuration of SAP Web IDE JavaScript validators.

Context

You can customize the configuration and rules of SAP Web IDE JavaScript validators directly in the SAP Web IDE UI.

If you change the default settings and turn on or off specific rules, this information is stored in a `.eslintrc` file in the root folder of the project. If you have already configured ESLint rules in another development environment, you can import your `.eslintrc` file into the root folder, instead of using the UI. Additional SAP Web IDE-specific information about each rule – such as category, severity and help URL – is stored in an automatically generated `.eslintrc.ext` file, also in the root folder.

Procedure

1. From the context menu of any file in your project, choose *Project Settings*.
2. From the *Project Settings* options, choose *Code Checking* > *JavaScript*.

   The default JavaScript validator opens displaying the SAP Web IDE default rules.
3. In the *Validator Configuration* field, you can define the validator configuration for globals and environments for the selected validator. The configuration should conform to `.json` file structure.
4. In the *Rules* table, configure the rules for the selected validator as follows:
   a. Enable each rule that you want to use to check your code by selecting the checkbox by the rule name.
   b. Configure the error level of the rules by setting the severity and category.
   c. Use the help link for each rule to access detailed rule information about how you can fix the detected issue.

5. Choose *Save* and refresh your browser page.

   The enabled rules will be implemented when you write your code.

   > **i** Note
   >
   > You can restore the default validator configuration and rules by clicking the *Reset* button next to the *Validator* field.

15.5.5.4 YAML Validation

A built-in client-side YAML validator is integrated with the code editor.

YAML files, such as MTA descriptors (`mta.yaml`), are automatically validated as you edit. Errors and warnings are indicated in the file tab and next to the line numbers with red and yellow markers respectively. When you hover over a marker, an error or warning message is displayed.

> **i** Note
>
> Only one error is indicated at a time. Once you fix it, the next one appears.

The client-side validator only checks the file syntax according to the YAML 1.2 specification. It does not check the MTA schema.

The syntax and schema of the `mta.yaml` files are validated during the build process on the server side according to the YAML 1.1 specification. The server-side validation messages appear in the Build console and logs.

Parent topic: Checking Code [page 1246]

Related Information

Configure Code Checking [page 1247]
Code Checking Annotations [page 1248]
JavaScript Validation [page 1248]
Using the Problems View [page 1251]

15.5.5.5 Using the Problems View

View information about problems in the projects in your workspace.

The *Problems* view displays information about problems in the modules in your projects.

The problem analysis is triggered by one of the following actions:

- Automatically whenever you select a project or a module when the *Problems* view is open.
- Manually by clicking the *Analyze and display problems* icon in the *Problems* view.
- When you build an HDB module or an entire project.

The *Problems* view displays the following information:

- The total error count and the error count for each severity for all the analyzed files. The error count does not change when you filter the display or change the severity selection.
- The severity of each problem. You can filter the list according to severity.
- The problem description with a link to more information.
- A link to the file that contains the problem. Click the link to open the file in the workspace at the location of the problem.
- The full path to the folder that contains the file with the problem.
- The problem category.

You can use the AutoFix feature to correct errors and warnings in your code in Java or JavaScript files in the *Problems* view:

1. In the *Problems* view, right-click an error or warning for a fixable problem.
2. Choose the indicated correction in the context menu.
   The code is automatically corrected.

For HTML5 and Node.js modules, the *Problems* view is dynamically updated when you edit, add, and delete files or delete a project within the scope of the analysis.

For JavaScript, you can exclude specific files or folders from the analysis:

1. Create a text file with the suffix `eslintignore` and put it under the root project folder.
2. In the file, enter the names of all the files and folders to ignore. Each entry should be on a separate line.
   The next time that you trigger an analysis, these files and folders will be excluded from the analysis.

Parent topic: Checking Code [page 1246]

Related Information

Configure Code Checking [page 1247]
Code Checking Annotations [page 1248]
JavaScript Validation [page 1248]
YAML Validation [page 1251]

15.5.6 Locate Objects in Code

The code editor allows you to locate objects or definitions of objects in code.

Context

Methods can be defined in other modules on which the source file depends. You can use *Goto Definition* to navigate to methods declared explicitly in the dependency file and generated methods for metadata properties, associations, aggregations, and events of SAPUI5 controls.

The dependencies can be defined using a dependency declaration in either of the following ways:

- Use `jQuery.sap.require(<moduleId1>,<moduleId2>,…)`. The dependency (target) file should contain the module declaration `jQuery.sap.declare(<moduleId>)`.
- Use `sap.ui.define([<dependencyFile1>,<dependencyFile2>,…], function(d1,d2,…){ });` where `dependencyFile` is the relative file path in the current project or the logical path to the library module. The dependency (target) files should contain a module declaration using `sap.ui.define` or Asynchronous Module Definition (AMD).

Suggestions can be provided only if the dependency file is visible in the current project.

You can:

- Use goto services to locate the definition action of a user-defined object (for example, a variable, function, object, or property) in JavaScript files for in a project. Goto searches the active file as well as all files in the same project.

  > **i Note**
  >
  > Goto is not supported for JavaScript native keywords (for example, `var`, `window`, `JSON`, or SAPUI5 library objects (for example, `sap`, `sap.ui`, `sap.ui.core`), as they are not user-defined objects. If you attempt to use goto with a restricted object, a message appears in the console pane to indicate that the definition was not found.

- Highlight all instances of the selected object in the active file.

Procedure

1. To locate the definition using goto services:
   a. Validate whether the object is user-defined by placing the cursor over the user-defined object and pressing `CTRL` + `ALT`.
   If the object has a user-defined definition, it appears underlined in blue. Otherwise, the console reports that the definition cannot be found.
   b. To go to the location, press `CTRL` + `ALT` and click the underlined object.
   If the definition is in the current file, the definition is highlighted. Otherwise, the correct file is first opened and the definition is highlighted.

> **Note**
>
> You can also conflate both of these steps into a single action by selecting the object and pressing CTRL + ALT + G, or selecting the object and choosing *Edit* > *JavaScript* > *Goto Definition*.

2. To locate all instances of an object in an open file, double-click the object.

   All instances of the object are highlighted in a blue box.

   For example, if you select the first instance of the string `content`, all remaining instances are identified in a blue box.

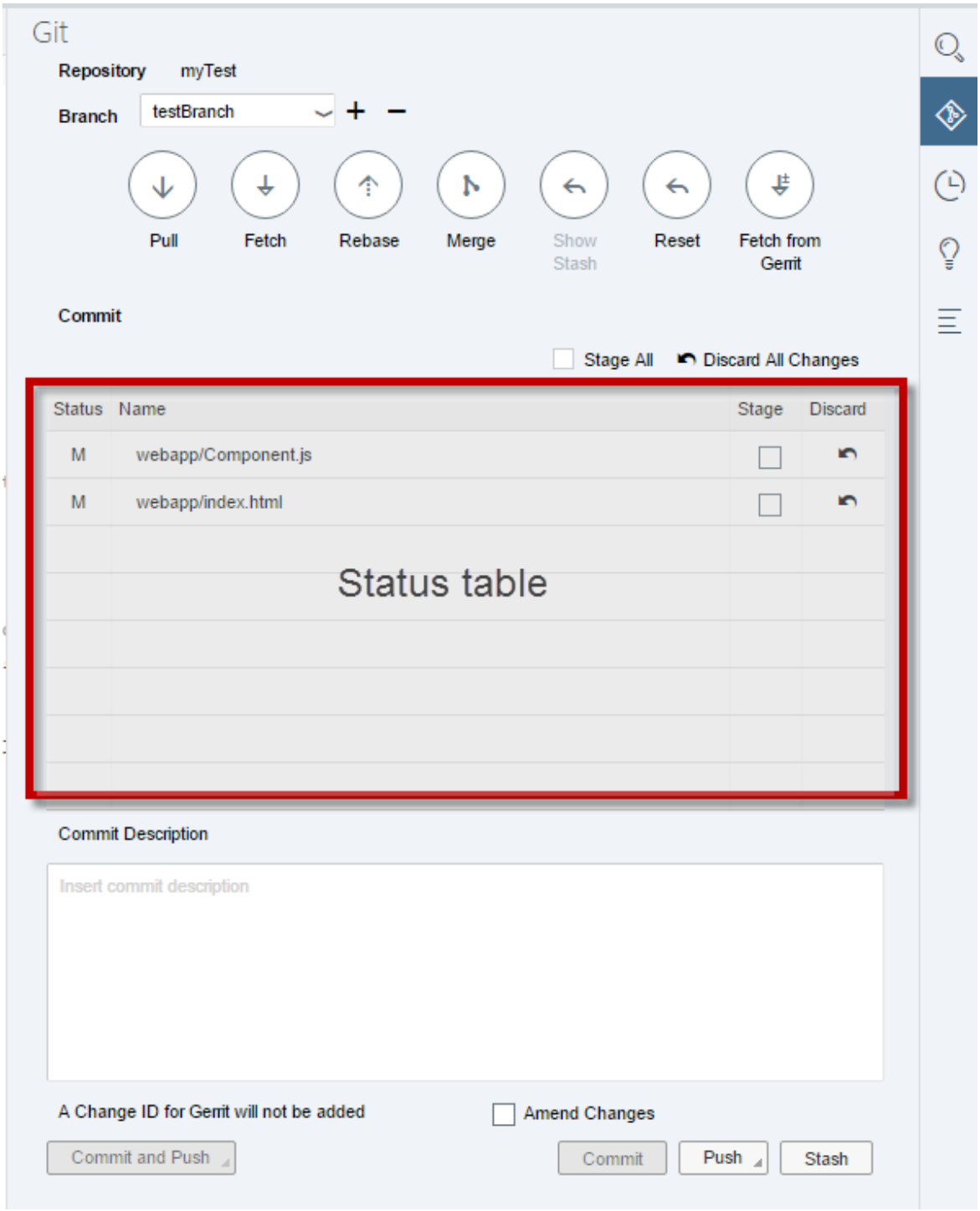
15.6 Using Source Control (Git)

SAP Web IDE enables you to connect and interact with Git source control system, letting you connect and interact with remote Git repositories.

Git Tools

SAP Web IDE provides a graphical user interface for executing Git commands and managing your source control and versioning. The following are the main tools for working with Git:

- **Git Menu:** Access the menu from *File* > *Git*. The menu includes the ability to clone a repository, as well as other Git commands for working with cloned repository.
- **Git Pane:** The Git pane provides a graphic user interface for executing Git commands on a specific Git repository, as well as a status table that lists all the uncommitted changes you've made to your project.



To open the *Git* pane:

1. From the workspace, select a Git repository
2. From the right sidebar, choose (Git pane).

- **Git History Pane:** Lets you view the commits for different branches.
  To open the *Git History* pane:
  1. From the workspace, select a Git repository

2. From the right sidebar, choose (Git History pane).

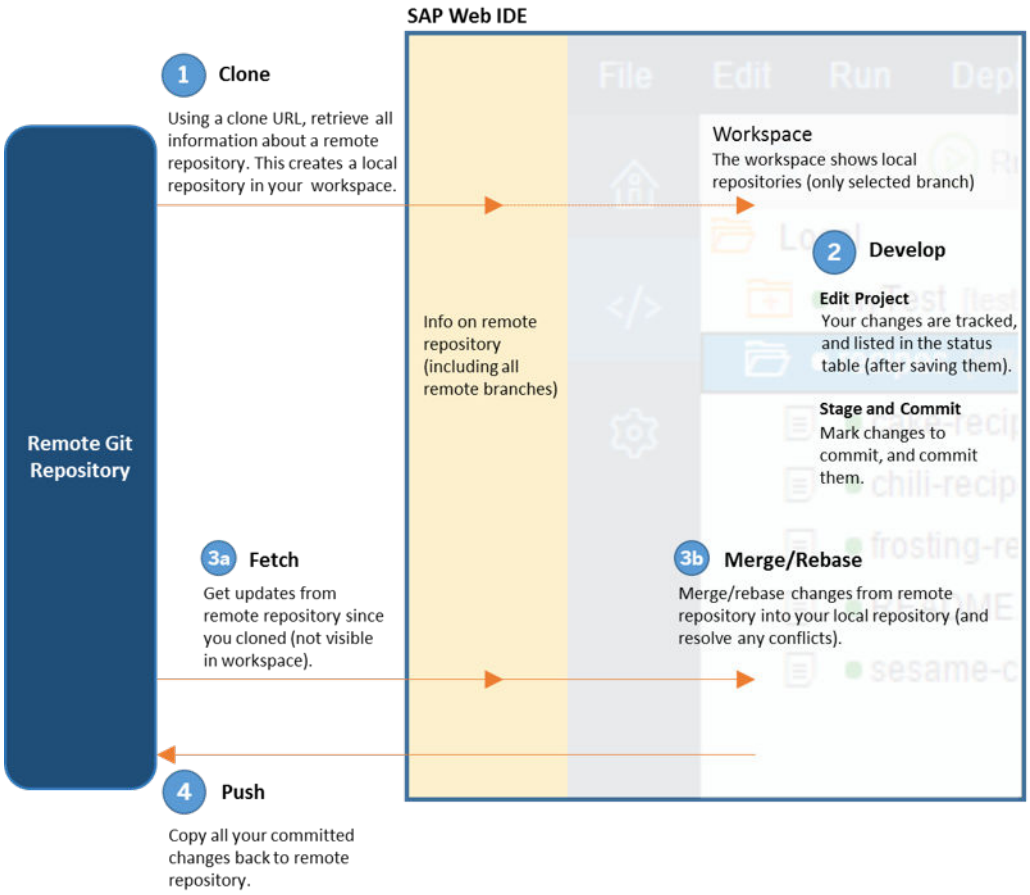
Workflow

Using Git with SAP Web IDE is easy. The basic workflow is as follows:

1. **Clone:** Clone a repository from a remote Git source control system. All the information about the repository is copied, and a local master branch is created and is visible in your workspace. If the remote repository has several branches, you can create additional local branches based on those remote branches.

   You can also create a new repository by creating a new project in your SAP Web IDE, and choosing *Git* > *Initialize Local Repository*. You can then connect the local repository to a remote repository by choosing *Git* > *Set Remote*

   .
2. **Develop:** Once you have the code, you can develop – add files, delete files, modify files. Your changes are visible in the status table of the *Git* pane. When you are ready, you can stage your changes and commit them.
3. **Fetch and Merge/Rebase:** (Optional) Before sending back your changes to the remote repository, you can fetch all the changes made by others. Then you can merge or rebase the changes into your changes to make sure there are no conflicts. If there are conflicts, you can adjust your code.
4. **Push:** Add your changes to the remote repository.



Related Information

Git Troubleshooting [page 1277]

15.6.1 Set Up Git

To use source control in your SAP Web IDE project, your user name and email address must be set for your Git account.

Context

> **i Note**
>
> If your Git server uses certificate authentication, you need to upload the server certificate to the SAP Web IDE server. For more information, see Manage SSL Certificates.

By carrying out the following steps, you can either set your Git user settings or check whether they are already set correctly.

> **i Note**
>
> Your user name and email address will be stored on the remote Git server. This is mandatory for Git operations and cannot be undone.

Procedure

1. Open SAP Web IDE in one of the supported browsers using the subscription URL.
2. Choose (Preferences) and select *Git Settings*.
3. Enter your email address and name.

   > **i Note**
   >
   > The email address field is case-sensitive.

   > **i Note**
   >
   > If you have not set your Git user name and email address, SAP Web IDE extracts this information from the identity provider defined in your account and pre-populates these fields in the *Git Settings* page.
4. Choose *Save*.

15.6.2 Clone Repositories

You can clone an existing Git repository into your workspace.

Procedure

1. From the *File* menu, choose *Git* > *Clone Repository*.
2. In the *URL* field, enter the Git repository URL and press `Enter`.
3. Enter your Git repository user name and password.
4. If your remote Git system works with Gerrit, select *Add configuration for Gerrit*.
5. Choose . The cloning starts. When the process is finished, the content of the repository appears in the workspace.

15.6.2.1 Configure Git Repositories

You can configure the Git repository for your project by creating new entries or deleting and editing existing entries.

Procedure

1. Right-click a Git repository, and choose *Project Settings* > *Git Repository Configuration*.
2. Configure the Git repository as follows:
   - To create a new entry, choose *Add Entry* and type the relevant values in the *Key* and *Value* fields.

   > **i Note**
   >
   > Use the following format for the *Key* entry: `<section>.<name>`. For example, `user.name`

   > **i Note**
   >
   > The *Key* field is mandatory and cannot be duplicated.

   - To edit an entry, choose (Edit).
   - To delete an entry, choose (Delete).
3. Choose (Save).

15.6.3 Initialize a Local Git Repository

You can initialize a local repository for any project that is not already connected to a Git repository.

Context

You can create an empty local repository for your project. This local repository can then be connected to a remote repository.

Procedure

1. Select the desired project.
2. Right-click and select *Git* > *Initialize Local Repository*.

Related Information

Set a Remote Repository [page 1259]

15.6.4 Set a Remote Repository

After initializing a local repository for your project, you likely will want to set a remote repository for your project, so you can push your work to a central Git repository.

Procedure

1. Select a project that has been initialized as a local Git repository (with *File* > *Git* > *Initialize Local Repository*).
2. Right-click and select *Git* > *Set Remote*.
3. Enter the remote repository URL, and a name for the remote repository.

> **i Note**
>
> Once you have set a remote repository, you can change it from the project settings.
>
> 1. Right-click the desired project and select *Project Settings*.

2. Select *Git Repository Configuration*.
3. Click *Add Entry* and add the following:

Key	Value
remote.origin.fetch	`+refs/heads/*:refs/remotes/origin/*`
remote.origin.url	<Remote Git repository URL>
branch.master.merge	`refs/heads/master`
branch.master.remote	`origin`

> **i Note**
>
> When you connect your local repository to a remote repository, before you push your changes you need to first perform a fetch. Otherwise you will not be able to see the remote branches, such as `origin/master`.

4. If your remote Git system works with Gerrit, select *Add configuration for Gerrit*.
5. Choose *OK*.

Related Information

Initialize a Local Git Repository [page 1259]

15.6.5 Fetch Changes

Fetching enables you to download objects and references from another repository into your local repository. You can then merge or rebase the changes into your project.

Procedure

1. In the *Git Pane*, choose *Fetch*.
2. Enter your Git repository user name and password.
3. Choose *Remember Me* to avoid being asked for credentials again in this session.
4. Choose *OK*.
5. Choose *OK*.

15.6.6 Rebase Changes

Rebasing enables you to take all the committed changes from one branch and incorporate them into a different branch.

Procedure

1. In the *Git Pane*, choose *Rebase*.
2. Select the branch from which you want to obtain the changes.

   > **i** Note
   >
   > The branch that is currently checked out is automatically disabled. It cannot be selected. By default, the corresponding remote branch is selected.

3. Choose *OK*. The latest changes are integrated and shown in your workspace.

   > **i** Note
   >
   > Rebase can fail due to conflicts between the current branch and the branch whose changes you want to incorporate. When conflicts are identified, the Git pane switches to *Rebase Interactive* mode and different actions are available to enable conflict resolution.

Related Information

Rebase Interactive Mode [page 1261]

15.6.6.1 Rebase Interactive Mode

When conflicts occur while rebasing a branch, rebase interactive mode is triggered.

When rebase interactive mode is enabled, the caption `[rebase in progress]` is displayed next to the repository name in the Git pane.

The following actions can be executed in the rebase interactive state:

- Continue the rebase process.
  1. Fix the conflicts that are visible in the status table and save the fixed files.
  2. Stage the fixed files.
  3. Choose *Continue*.
  4. If all conflicts are resolved, the repository returns to its normal mode. If errors still exist, repeat the procedure.
- Abort the rebase process.

Choose *Abort*.

- Skip a specific conflicting commit.
  Choose *Skip Patch*.
- Reset changes. Resetting changes discards all staged and unstaged changes on the current local branch, so that it is identical to the remote branch.
  Choose *Reset*.

15.6.7 Merge Changes

You can incorporate all the changes from one branch into another in a single commit.

Procedure

The IDE recognizes merge conflicts. The differences are highlighted and you can choose whether to accept the local change or the new one. You can also accept both modifications at the same time.

1. In the *Git Pane*, choose *Merge*.
2. Select the branch from which you want to obtain the changes.

   > **i** Note
   >
   > The branch that is currently checked out is automatically disabled. It cannot be selected. By default, the corresponding remote branch is selected.

3. Choose *OK*. The latest changes are integrated and shown in your workspace.

   > **i** Note
   >
   > Merge operations can fail due to conflicts between the current branch and the branch you chose from which to incorporate the changes.

15.6.8 Pull Changes

Pulling is the same as fetching and merging. Pulling enables you to download objects and references from another repository into your local repository, and then merge the changes into your project.

Procedure

1. In the *Git Pane*, choose *Pull*.
2. Enter your Git repository user name and password.

3. Choose *Remember Me* to avoid being asked for credentials again in this session.
4. Choose *OK*. The changes are fetched from the specific branch and merged into your local checked-out branch.

15.6.9 Stage Files

The staging table shows changed files, and lets you select files to stage.

Context

Whenever a file is updated, added, or deleted, it appears in the staging table, but is not staged. After staging, you can commit the staged files.

Procedure

In the staging table of the *Git Pane*, choose the *Staged* toggle button to the right of the change that you want to stage.

> **i** Note
>
> To stage all files in the staging table, select the checkbox at the top of the table, and choose *Stage Selected* from the dropdown list.

You can edit a file listed in the staging table by clicking (More Actions) and choosing *Edit*. This option is enabled for all files, except for deleted.

You can delete a file listed in the staging table by clicking (More Actions) and choosing *Delete*. This removes the file from the staging table and from the workspace.

Related Information

Discard Changes [page 1265]

15.6.9.1 Compare and Merge Code

You can compare different versions of your code and merge the changes.

Context

Use the compare editor to compare a previous version of your code (the server version), with your modified version from the staging table in the *Git Pane*.

You can also compare two committed versions from the *Git History* pane.

Procedure

1. In the *Git Pane*, double-click on a modified file (*) in the staging table, or click (More Actions), and choose *Compare*.

   The title of each pane indicates whether it contains the current modified or the previous read only version.
2. Compare your modified code with the original code as follows:

   - Click (Sync/Unsync scrolling between panes) to decide if you want both panes to scroll together or independently.
   - Click (Next) and (Previous) to navigate between the highlighted differences in the code of the original and modified versions. As you navigate through the changes in the file, the color of the changed code deepens to indicate your cursor position.
   - Click (Show/Hide trailing spaces) to decide if you want to include the trailing white spaces in the compare.
   - Click (Undo) and (Redo) to undo or redo changes.
   - Click (Accept my changes) to add your changes to the current file. Click (Accept their changes) to overwrite your changes with the previous changes.
   - Click (Push to left) to move selected code from the previous version to the modified version. The highlighted lines on the right side will replace the highlighted lines on the left side.
3. If there is a conflict between the 2 versions, a three-way merge editor opens.

   The left pane contains your changes, the right pane contains the previous changes, and the middle pane is the editor in which you perform the code merge.

   - Click (Push to left) to move selected code from the original version to the merged version.
   - Click (Push to right) to move the selected code from your version to the merged vesrsion.

15.6.9.2 Discard Changes

Discarding removes all changes from an existing file in the local environment. For example, discarding a new file deletes the file from the branch.

Context

> **i** Note
>
> Only unstaged files can be discarded.

Procedure

In the staging table of the *Git Pane*, select the checkbox in the row that contains the change that you want to discard, and choose *Discard*.

All changes that you made to the file are removed.

> **i** Note
>
> To discard all files, select the checkbox at the top of the table, and choose *Discard*. All unstaged files in the staging table are discarded.

15.6.10 Commit Changes

You can commit changes to the repository locally.

Procedure

1. In the *Git Pane* status table, select the *Stage* checkbox for the files you want to stage (or click *Stage All* above the table).
2. Enter a description of the change in *Commit Description*.
3. If you want to add the current changes to the last commit, select the *Amend Changes* checkbox. The commit description of the last committed change appears in *Commit Description*, which you can modify.
4. Choose *Commit*. The changes are committed locally, and one is added to the counter for unsynched commits at the top of the , next to the repository name.

Results

The user name and email listed for your commits are the user name and email from your SAP HANA system, not the user name and email from your Git system.

15.6.11 Push Changes

The Push option incorporates all unsynced committed changes into the remote branch of the currently checked-out local branch. The number of unsynced committed changes is displayed next to the repository name. All tags created within the open repository are pushed.

Procedure

1. In the *Git Pane*, choose *Push*.

   For your convenience, you can instead use *Commit and Push* to commit the currently staged changes and then immediately push them to a remote branch. Before choosing *Commit and Push*, remember to stage your changes and to add a description for the commit.
2. Choose one of the following from the dropdown list:
   - *origin/<remote branch>* if your local branch is based on a specific remote branch.
   - *Remote Branch* to select a different remote branch.
3. Enter your Git repository user name and password.
4. Choose *Remember Me* to avoid being asked for credentials again in this session.

15.6.12 Multiple Local Branches

From the *Git Pane*, you can check out a local branch, add a new local branch, and remove a local branch.

Checking Out a Local Branch

Procedure

In the *Git Pane*, select the desired local branch.

> **i** Note
>
> If you have uncommitted changes in your workspace, a dialog box containing the list of conflicting files opens. Choose *Cancel* to abort, or *Reset and Checkout* to remove all uncommitted changes.

The selected branch is checked out. The name of the selected branch is shown in the workspace next to the name of the project.

Creating a New Local Branch

Context

You can create a new local branch referencing any available remote or local branch.

Procedure

1. In the *Git Pane*, choose + (Add Branch).

   The *Create a New Branch* dialog box appears.
2. From the *Source Branch* dropdown list, select the desired local or remote branch.
3. Enter a name for the new local branch.
4. Choose *OK*.

   > **i** Note
   >
   > If you have uncommitted changes in your workspace, a dialog box containing the list of conflicting files opens. Choose *Cancel* to abort, or *Reset and Checkout* to remove all uncommitted changes.

   The new local branch is created and checked out.

Deleting a Local Branch

Context

If there is only one branch available, it cannot be deleted.

Procedure

1. In the *Git Pane*, choose − (Delete Branch). The *Delete Branch* dialog box appears showing all the branches of the selected repository.
2. Select one or more branches that are not checked out.
3. Choose *Delete*.

   The selected branches are deleted.

15.6.12.1 Reset Local Branches

You can delete all new objects and references that were added to an existing local branch to make it identical to its remote branch.

Context

When you reset a branch, all unsynced committed changes are removed, and all staged and unstaged files are reverted to their original state in the local copy of the respective remote branch.

Procedure

1. In the *Git Pane*, choose *Reset*.
2. Select the branch that you want to revert back to.

   > **i Note**
   >
   > The branch that is currently checked out is automatically disabled. It cannot be selected. By default, the corresponding remote branch is selected.

3. Choose a *Reset Type*.

   > **i Note**
   >
   > If you choose a *Hard* reset, all changes are removed.

4. Choose *OK* to reset the branch.

15.6.13 Create Remote Branches

You can create a new branch in the remote repository.

Context

To work on the remote branch you just created, you still must check out the branch by creating a new local branch for this new remote branch.

> **i Note**
>
> To create a local branch, you can choose the menu *Git* > *Create Local Branch* or click the plus sign in the *Git Pane*.

Procedure

1. Select your project.
2. From the menu, select *Git* > *Create Remote Branch*.

> **i Note**
>
> Deleting remote branches is currently not supported. You can delete a remote branch using a different Git client.

Related Information

Multiple Local Branches [page 1266]

15.6.14 Git History

From the Git *History* pane, you can explore the history of committed changes that were made for repositories, folders, and files in a specific project.

Git history is located in a dedicated pane that you can access in one of the following ways:

- Using the main menu: *File* > *Git* > *History*.
- Using the context menu of the selected Git repository.
- Using the dedicated icon (Git History pane) from the right sidebar.
- Using the *Show Git Blame* option from the context menu of the file editor.

Once you have selected a Git repository in the workspace, use the Git *History* pane to:

- Explore the history of any branch of your committed code by selecting one or more branches from the *History of Branch* dropdown list. When you first select a file or folder in the workspace, details of the master branch are displayed in the Git *History* pane by default – you can either select a different branch or all branches.
- View a list of commits (descriptions of the commits) in the central area of the pane. A commit graph on the left gives you a visual representation of the commit history.

  Any tagged commit is shown with the tag icon ( ); hover over the icon to see a list of tags.
- Search commits by:
  - The author of the commit
  - Person who committed the change
  - Commit ID
  - Date

    The results of your search are highlighted and you can toggle between them using the arrows on the right of your filter.
- Select a commit in the list to view details of that commit in the area below.

- View a list of all the files packaged in this commit in the *File* column in the area below. From the *Status* column, you can see one of the following statuses of the file:

Status	Description
N	New
D	Deleted
M	Modified
C	Contains conflicts

- Execute the following Git commands:
  - Tag a commit.
  - Cherry-pick a change.
  - Revert a commit.
  - Check out a commit.
  - Compare commits.

  Git Commands from the Git History Pane [page 1270]
  From the Git *History* pane, you can execute a number of Git commands.

  Git Blame [page 1272]
  Use the Git Blame feature to view the details of the last revision for each line in a code file.

15.6.14.1 Git Commands from the Git History Pane

From the Git *History* pane, you can execute a number of Git commands.

Context

Following are the main Git commands that you can execute from the Git *History* pane.

Operation	Description	Steps
Tag a commit	Allows you to mark specific points in history as important, usually used for release points.	1. Choose *Tag*. 2. Enter a name for the new tag. 3. Choose *OK*.

Cherry-pick a change	Allows you to apply the changes of the selected commit to only the current branch.
	
	i Note
	You can only cherry-pick a commit if your Git user settings have been defined. The IDE checks if your Git user name and email address exist. If not, a dialog box opens and by default displays the user name and email that was extracted from the identity provider defined in your account. You can either correct these entries or enter new details.
		1. Choose *Cherry-Pick*.
		2. Choose *OK* to confirm the commit.
		3. If there is a conflict, a warning is displayed. From the Git pane, resolve any conflicts as follows:
		  1. Open the Git pane and find the conflicting files in the staging table.
		  2. Edit the files to resolve the conflicts.
		  3. Stage the edited files.
		  4. Commit the changes.

Revert a commit	Allows you to undo all the changes that were incorporated in the selected commit. **i Note** You can only revert a commit if your Git user settings have been defined. The IDE checks if your Git user name and email address exist. If not, a dialog box opens and by default displays the user name and email that was extracted from the identity provider defined in your account. You can either correct these entries or enter new details.	1. Select the commit in the list of commits and choose *Revert*. 2. If there is a conflict, a warning is displayed. Open the Git pane to resolve any conflicts as above.
Check out a commit	Allows you to take the code from a specific commit and create a new branch based on it.	1. Select the commit in the list of commits, and choose *Check Out*. 2. Enter a name for the new branch. 3. Choose *OK*. The new branch appears as checked out in the Git pane, and all the commit content is refreshed in the workspace. **i Note** When checking out an already committed change, conflicts can occur. To resolve these conflicts, make sure that the conflicting file is not open in the staging table.
Compare commits	Allows you to compare a modified version of your code with the original version from the staging tables in the Git pane. You can also compare two committed versions.	Double-click the changed file to open a compare view in the code editor, so that you can compare the different versions of the file. For more information, see Compare and Merge Code [page 1264].

15.6.14.2 Git Blame

Use the Git Blame feature to view the details of the last revision for each line in a code file.

When you open a code file in the code editor, and choose *Show Git Blame* in the context menu, a read-only file named `Blame-<original file name>` opens in a new tab. Details of the last revision, such as commit ID, date, and author, are displayed for each code line in the file's gutter.

You can further explore the revision details:

- Hover over a line to display a tooltip with more revision details.
- Click a line to open the respective commit in the *History* pane and display all the commit details.

Press **Esc** to hide the blame annotations.

15.6.15 Set Up Git to Work with Gerrit

Gerrit is a web-based software code review tool for reviewing, approving, or rejecting changes to the source code developed by your colleagues. Gerrit works as an intermediate environment for source control between the local environment and the remote Git repository.

Procedure

1. From the *File* > *Git* menu, choose *Clone Repository* or *Set Remote*.

   If you select *Set Remote*, your project must have been initialized as a local repository with *File* > *Git* > *Initialize Local Repository*.
2. Select the *Add configuration for Gerrit* checkbox.

Results

Anytime you push, the changes will be sent to Gerrit for code review.

> **i Note**
>
> Make sure to only select the checkbox if your Git uses Gerrit.

Related Information

Clone Repositories [page 1258]
Initialize a Local Git Repository [page 1259]

Set a Remote Repository [page 1259]
Fetch Changes from Gerrit [page 1273]

Manual Setup

You can still set up a local Git repository to work with Gerrit even if you did not specify the Gerrit configuration when cloning or setting a remote repository.

Procedure

1. Right-click your project and choose *Project Settings* > *Git Repository Configuration*.
2. Choose *Add Entry*.
3. In the *Key* field, enter **`gerrit.createchangeid`**
4. In the *Value* field, enter **`true`**..
5. Choose *OK*.

15.6.15.1 Fetch Changes from Gerrit

When your repository is set up to work with Gerrit, you work with Git as normal, and review code changes in your Gerrit system. You can also fetch a change from Gerrit and create a local branch from the change, and then collaborate with a colleague using Gerrit before merging the change in Git.

Procedure

1. From the workspace, select your project.
2. Open the *Git Pane*.
3. Choose *Fetch from Gerrit*.
4. Enter the change that you want to download, and choose *OK*.

   Use the ref specification for the change, in the form of **`refs/changes/79/2565079/3`**. This is available under *Download* in the standard Gerrit UI.

   The changes are fetched into a new local branch, and the branch is checked out.

   > **i Note**
   >
   > If you have uncommitted changes in your workspace, a dialog box containing the list of conflicting files opens. Choose *Cancel* to abort, or *Reset and Checkout* to remove all uncommitted changes.

Results

When you change or update your code in a repository associated with Gerrit, and then stage, commit and push the changes to Gerrit, a notification confirms that the push operation was successful. A link to the committed change in the Gerrit tool is displayed.

15.7 Protecting User Data

SAP Web IDE provides administrative features that support user data protection.

Export User Workspaces

Upon request of an SAP Web IDE user, an administrator can export a .zip file containing all the workspaces created by this user.

Prerequisites

To export user workspaces, you need the following:

- An administrator role for the user account.
- The ID of the user whose workspaces you want to export.

Procedure

1. Obtain the URL of the XS Advanced administration tool by running the command `xs app di-cert-admin-ui --urls` in the CLI of XS Advanced.
2. In the browser address bar, enter `https://{admin tool URL}/che/workspace/export-all/{userid}`, where `userid` is the ID of the user whose workspaces you want to export.

   > **i** Note
   >
   > If the user ID contains special characters, encode it into URL format.

3. Log in as a system administrator.

   A .zip file containing all workspaces of the specified user is available for download.

Manage User Service Instances

Context

For a resource defined in the MTA descriptor (`mta.yaml`) of a user's application, SAP Web IDE may generate a service instance in XS Advanced, adding the corresponding user ID as a prefix to the service instance name.

To manage the service instances created for a particular user, perform the following steps:

Procedure

1. Open SAP HANA XS Advanced cockpit.
2. In the home navigation pane, choose *Organization*, your organization and space, and then *Services* > *Service Instances*.
3. Filter the list of services by entering the relevant user ID in the *Search* field.
4. Delete any or all of the services, as required.

Delete User Data

Context

SAP Web IDE users can delete all data related to them by simply deleting all projects in the workspace and by removing the Git settings under the project settings. An administrator can delete a user's workspaces and data from SAP Web IDE. For this, an administrator needs to open an incident through Product Support with component CA-WDE-OFF.

> **i** Note
>
> Your Git user name and email address will be stored on the remote Git server. This is mandatory for Git operations and cannot be undone.

Log Changes of Personal Data

Context

Changes to personal data in SAP Web IDE are written to the SAP HANA audit log. As a prerequisite, an administrator needs to activate SAP HANA audit log and create an audit log policy in SAP HANA for XSA, as described here.

Related Information

Access the SAP Web IDE Administration and Development Tools
Inside an MTA Descriptor [page 1079]

15.8 Troubleshooting

Here are some common troubleshooting issues in SAP Web IDE.

Archive Import Troubleshooting [page 1276]
: If you are having trouble importing an archive (.zip file) into SAP Web IDE Full-Stack, you can try the following possible solutions.

Git Troubleshooting [page 1277]
: Steps you can take if you have trouble using Git.

Grunt Build Troubleshooting [page 1279]
: If you are having trouble running the Grunt build for your project, you can try the following possible solutions.

Running Applications Troubleshooting [page 1281]
: If you are having trouble running an application, you can try the following possible solutions.

Remote Connectivity Troubleshooting [page 1283]
: Options for troubleshooting SAP Web IDE remote connectivity issues.

15.8.1 Archive Import Troubleshooting

If you are having trouble importing an archive (.zip file) into SAP Web IDE Full-Stack, you can try the following possible solutions.

- **UTF-8 Encoding**
  Make sure that the archive files and folder names in your project contain only characters that are encoded with UTF-8 encoding. In other words, these names must not contain any special characters such as a question mark (?) or an ampersand (&). In addition, make sure to use only Latin characters. Non-Latin characters such as Chinese, Japanese, or Hebrew cannot be used.
- **Check File Size**
  The `.zip` file you want to import must be less than 100 MB.
- **Import `.zip` Files Only**
  Make sure that the archive you want to import is a `.zip` file. Archives with other extensions, such as `.rar`, cannot be imported.

Multitarget Application (MTA) Project Troubleshooting

- Check the correctness of the `mta.yaml` file.
  For more information, see Inside an MTA Descriptor [page 1079].
- Make sure that the path element of each module in the `mta.yaml` file points to a folder that actually exists in the archive.
- Make sure that the path value specified in the `mta.yaml` file is a relative path; in other words, that it does not start with a period (.), slash (/), or backslash (\).

15.8.2 Git Troubleshooting

Steps you can take if you have trouble using Git.

The following lists error messages you may receive when using Git, possible causes of the error, and possible solutions.

Cannot convert folders to modules

This error occurs when folders of MTA applications cannot be converted to modules according to the `mta.yaml` file.

- Check the correctness of the `mta.yaml` file. For more information, see Inside an MTA Descriptor [page 1079].
- Make sure that the path element of each module in the `mta.yaml` file points to a folder that actually exists in the archive.
- Make sure that the path value specified in the `mta.yaml` file is a relative path; in other words, that it does not start with a period (.), slash (/), or backslash (\).
- Make sure types of modules are one of the following values: `html5`, `hdb`, `nodejs`, `java`, `cds`, `sitecontent`, `dwf`.
- After you have identified and fixed the root cause of the failure:
  - Create a new empty module of the required type.
  - Copy the content from the problematic folder to the new module
  - Update the `mta.yaml` file so that the corresponding `path` property in the module points the new module root folder.
  - Delete the obsolete folder.

Invalid committer

The e-mail listed in the Git repository is not the same as the e-mail assigned to you in Gerrit.

1. Right-click on your project, select *Project Settings* > *Git Repository Configuration* and change the e-mail address in the `user.email` field.

2. Open the *Preferences* perspective, choose *Git Settings*, and verify that the *Git Emaill Address* value is correct.
3. Commit your changes again, this time by selecting the *Amend Changes* checkbox and then selecting *Commit*. You can then try to push your changes again.

NON FAST FORWARD

Someone else pushed new changes to the remote repository.

Sync your repository (either *Fetch* and then *Rebase*, or *Pull*).

Prohibited by Gerrit

If you did not select the *Add configuration for Gerrit* checkbox when cloning your project, and the Git repository is connected to a Gerrit server, your remote repository will not work.

Right-click on your project, select *Project Settings* > *Git Repository Configuration*, and add an entry with the key *gerrit.createchangeid* and set the value to `true`.

Cannot upload review

You do not have permission to push changes to Gerrit. Request from your administrator permissions on Gerrit.

Checkout failed

One of the files listed in the .gitignore file is preventing you from checking out a different branch.

1. Open `.gitignore` file (original branch).
2. Remove lines containing files blocking checkout.
3. Click *Discard* for the file if it appears in the staging table in the Git pane.
4. Stage the `.gitignore` file and commit it.
5. Check out your branch.
6. Create `.gitignore` file and save it. (If the file already exists, then make a small change and save it.)
7. In staging table, right-click the files blocking checkout and select *Untrack and Ignore*.
8. Stage, commit, and push your changes.
9. Merge your change in Gerrit.
10. Check out the original branch.
11. Select *Reset*.

Not authorized

You are not authorized in the Git system. This error may occur simply because the password was incorrect, for one of the following reasons:

- You entered the wrong password.
- The wrong password was cached in the browser. Clear the browser cache of passwords.
- The Git server requires a personal access token authorization.

Clone request failed

A clone request may fail for a variety of reasons. Check the error message for the specific reason.

- **Invalid Git repository URL:** You entered the wrong clone URL.
- **Cannot open git-upload-pack:** You are using a Git system within your corporate network, and you did not configure the proxy settings properly. For more information on configuring proxy settings for an internal Git system, see Optional: Customizing the Environment.
- **SAP Web IDE is not configured to trust the security certificate provided by the Git server:** If your Git server uses certificate authentication, you need to upload the server certificate to the SAP Web IDE server. For more information, see Manage SSL Certificates.

Fetch request failed (wrong remote URL)

This error occurs after you initialize a local Git repository from a project (*Initialize Local Repository*) and then you set a remote repository with the wrong URL. When setting a remote repository, a fetch is automatically performed, and if the wrong URL is entered, the fetch will fail. The error message includes *Git repository not found*.

If you continue and do other actions with the remote repository, these actions will also fail.

To fix the URL, right-click your project and go to *Project Settings* > *Git Repository Configuration*, and then change the *remote.origin.url* field to the correct URL.

15.8.3 Grunt Build Troubleshooting

If you are having trouble running the Grunt build for your project, you can try the following possible solutions.

Running Automated Tests with the Grunt Build's Test Task

You can execute unit and integration tests by triggering the `"@sap/grunt-sapui5-bestpractice-test"` Grunt task from the CLI.

1. Enter the following command to add the `grunt-sapui5-bestpractice-test` to your `package.json` file:
   `npm install –save-dev @sap/grunt-sapui5-bestpractice-test`
2. Add the following script to your `package.json` file to enable running the unit and integration tests via npm:
   `“scripts”: { “test”: “grunt unit_and_integration_tests” }`
3. Add the following test task to your `Gruntfile.js` file:

```
grunt.loadNpmTasks(“@sap/grunt-sapui5-bestpractice-test”);
grunt.registerTask(“unit_and_integration_tests”,
      [
“test”
      ]
);
grunt.config.merge({
coverage_threshold: {
statements: 0,
branches: 100,
functions: 0,
lines: 0
}
});
```

4. Run the following command from the CLI to execute the unit and integration tests:
   `npm test`

Running a Grunt Build with an Older SAPUI5 Version

When building a project using the `grunt-sapui5-bestpractice-build`, a build task is performed.

If you are using an SAPUI5 version below 1.52, include the following tasks in the build task:

```
grunt.registerTask('build', [
           'copy:copyToTmp',
           'devxUpdateManifest',
           'cssmin',
           'openui5_preload:preloadTmp',
           'copy:copyToDbg',
           'copy:copyDbgToDist',
           'copy:copyProjectFilesToDist',
           'uglify:uglifyPreload',
           'copy:copyTmpToDist',
           'devxUpdateNeoApp',
           'createManifestBundle',
           'cleanTmp',
           'createsCachebusterInfoJson',
           'cleanupChanges',
           'createResourcesJson'
       ]);
```

15.8.4 Running Applications Troubleshooting

If you are having trouble running an application, you can try the following possible solutions.

Running Applications

When running an application, the preview tab is blank. No errors appear in the console.

- This is caused by a missing destination to the DI application.
  For more information, see Running Applications in Development Mode.

When running an application, the preview tab opens with `404 Not Found` or `500 internal error` errors.

- The URL property in the destination to the DI application is not configured correctly.
  For more information, see Running Applications in Development Mode.

When running an application, the preview tab does not open.

- If this is your first time running your application in a browser, there might be a popup blocker. Make sure to disable it in your browser preferences.
  For more information, see this blog post.

Preview In Frame

When running an application with the *Preview Mode* > *With Frame* option selected, the preview is disabled.

- Check if the application is protected against clickjacking.
  1. In your application, search for the file containing the `sap-ui-bootstrap` section.

     Sample Code

     ```
     <script id="sap-ui-bootstrap"
             src="../../resources/sap-ui-core.js"
             data-sap-ui-async="true"
             data-sap-ui-libs="sap.m"
             data-sap-ui-theme="sap_belize"
             data-sap-ui-compatVersion="edge"
             data-sap-ui-resourceroots='{"demo.demo": "./"}'>
         </script>
     ```

  2. Check if the section contains the line `data-sap-ui-frameOptions="deny"`. This indicates that the application will be disabled when displayed in an iframe.
  3. Right click the on the project and select *Run* > *Run Configurations*.
  4. Select the configuration you are using and, in the *General* tab, select *Preview Mode* > *Without Frame*.
  5. Run the application. The application opens without a frame and can be edited.

Optionally, you can change the `data-sap-ui-frameOption` line to `"allow"`. However, this will remove the clickjacking protection from your application.
For more information, see Frame Options.

When running an application with the *Preview Mode* > *With Frame* option selected, the preview displays the SAP BTP logon screen.

- Make sure the *Accept 3rd-party cookie* checkbox is selected in your Chrome settings.

When running an application with the *Preview Mode* > *With Frame* option selected in MAC Safari private mode, the text in the buttons is not properly displayed.

- This is a known limitation of the preview.

AppCacheBuster

When running an application, the preview does not reflect the changes made to the source code.

- In the `neo-app.json` file, search for the `cacheControl` entry and delete it.

When running an application, the application `appcahebuster` settings are overwritten by the SAP Web IDE `appcachebuster`.

1. In your application, search for the file containing the `sap-ui-bootstrap` section.

   Sample Code

   ```
   <script id="sap-ui-bootstrap"
           src="../../resources/sap-ui-core.js"
           data-sap-ui-async="true"
           data-sap-ui-libs="sap.m"
           data-sap-ui-theme="sap_belize"
           data-sap-ui-compatVersion="edge"
           data-sap-ui-resourceroots='{"demo.demo": "./"}'>
       </script>
   ```

2. Check if the section contains the line `data-sap-ui-appCacheBuster="../,<path to external library appcachebuster>"` . For example, `data-sap-ui-appCacheBuster="../,sap/bc/ui5_ui5/ARTEC/PPCUTIL/,/sap/bc/ui5_ui5/ARTEC/PPSUTIL/"`
3. Delete the `sap-ui-appCacheBuster` parameter from the URL.

Security

Cannot run the application after configuring the Active Directory Federation Services (ADFS).

- Set the ACS (Assertion Consumer Service) in the Trusted Identity Providers settings to *Assertion Consumer Service*.

Other Issues

Header request parameter removed when using SAP Web IDE.

- To send header parameter from SAP Web IDE, add the header parameter names in the `neo-app.json` file in the `'headerWhiteList'` attribute.
  See Run Applications from the Workspace.

15.8.5 Remote Connectivity Troubleshooting

Options for troubleshooting SAP Web IDE remote connectivity issues.

Whenever there is a need to troubleshoot an SAP Web IDE remote connectivity issue, whether it is for a newly developed SAP Web IDE extension or because of a software failure, it is highly recommended to start checking **without** using:

- **SAP Web IDE logic**
  Avoid superfluous logic and simplify reproducibility.
- **Web browser XHR** (XMLHttpRequest, such as Ajax, Asynchronous JavaScript, and XML):
  Avoid the Same Origin Policy (SOP) web browser security mechanism, which stops scripts from accessing data from a different origin (domain).

Accordingly, check the connectivity with a web browser (enter the URL in the web browser address bar) and HTTP destination with the following two alternatives:

- Based on an HTML5 app:
  `https://<app-url>/<route-path>/<resource>`
  For more information, see Accessing REST Services.
- Based on SAP Web IDE routing:
  `https://<webide-url>/destinations/<destination-name>/<resource>`
  In the SAP BTP cockpit destination settings, under *Additional Properties*, add the `WebIDEEnabled` property and set the value to **`true`**.

If connectivity is still not working properly, continue with deeper analysis options:

- Check the connectivity using a web browser without an HTTP destination.
  It is still preferable to proceed without XHR; however, if XHR is used and there is a Cross-Origin Resource Sharing (CORS) failure, then disable the web browser security feature (just during troubleshooting) by using a web browser argument or a web browser add-on.
- Check the connectivity without a web browser and without an HTTP destination; for example, based on a tool such as Postman.

Related Information

Connection to External Systems

Important Disclaimer for Features in SAP HANA

For information about the capabilities available for your license and installation scenario, refer to the Feature Scope Description for SAP HANA.

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.
About the icons:

- Links with the icon : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
  - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
  - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon : You are leaving the documentation for that particular SAP product or service and are entering an SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.
The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Bias-Free Language

SAP supports a culture of diversity and inclusion. Whenever possible, we use unbiased language in our documentation to refer to people of all cultures, ethnicities, genders, and abilities.

www.sap.com/contactsap

© 2023 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. The information contained herein may be changed without prior notice.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

Please see https://www.sap.com/about/legal/trademark.html for additional trademark information and notices.

THE BEST RUN SAP