Testing JavaScript with XSUnit. ...................................................... 123
1  Getting Started with XS JavaScript

SAP HANA XS Javascript (XSJS) is a language that can be used by application developers to create native SAP HANA applications that expose data to UI clients on request.

The application developer defines the business logic that is used to expose data in response to client requests via HTTP. This guide takes you through the tasks required to use the XS JavaScript syntax to define the services that comprise your business logic, for example:

- Create applications using XS JavaScript
- Create reusable XS JavaScript libraries
- Make use of the XS JavaScript application programming interface (API)
- Debug XS JavaScript
- Trace XS JavaScript applications

This SAP HANA XS JavaScript Reference also provides code examples that illustrate how to use the XS JavaScript classes and methods.

Building the data model is the first step in the overall process of developing applications that provide access to the SAP HANA database. When you have created the underlying data persistence model, application developers can build the application services that expose selected elements of the data model to client application by means of so-called “data end-points”. The client applications bind UI controls such as buttons or charts and graphs to the application services which in turn retrieve and display the requested data.

Prerequisites

Before you can start using XS JavaScript to define the applications that represent your business logic, you need to ensure that the following prerequisites are met:

- You must have access to an SAP HANA system.
- You must have already created a development workspace and a project.
- You must have shared a project for the XS JavaScript artifacts so that the newly created files can be committed to (and synchronized with) the repository.

1.1  Developing Native SAP HANA Applications

In SAP HANA, native applications use the technology and services provided by the integrated SAP HANA XS platform.

The term “native application” refers to a scenario where applications are developed in the design-time environment provided by SAP HANA extended application services (SAP HANA XS) and use the integrated SAP HANA XS platform illustrated in the following graphic.
A program that consists purely of SQLScript is also considered a native SAP HANA application.

The server-centric approach to native application development envisaged for SAP HANA assumes the following high-level scenario:

- All application artifacts are stored in the SAP HANA repository
- Server-side procedural logic is defined in server-side (XS) JavaScript or SQLScript
- UI rendering occurs completely in the client (browser, mobile applications)

Each of the levels illustrated in the graphic is manifested in a particular technology and dedicated languages:

- **Calculation Logic** - data-processing technology:
  - Data:
    - SQL / SQLScript, Core Data Services (CDS), DDL, HDBtable
  - SQL / SQLScript
  - Calculation Engine Functions (CE_*)
  - Application Function Library (AFL)

  **Note**
  SAP recommends you use SQL rather than the Calculation Engine functions.

- **Control-flow logic with SAP HANA XS**:
  - OData

  Validation models for OData services can be written in XS JavaScript or SQLScript
Server-Side JavaScript (XSJS)
HTTP requests are implemented directly in XS JavaScript

XMLA

Client UI/Front-end technology:
- HTML5 / SAPUI5
- Client-side JavaScript

The development scenarios for native application development are aimed at the following broadly defined audiences:

Table 1: Target Development Audience for Native SAP HANA Applications

<table>
<thead>
<tr>
<th>Audience</th>
<th>Language</th>
<th>Tools</th>
<th>Development Artifacts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database developers</td>
<td>SQLScript, CDS, hdb*, SAP</td>
<td>• SAP HANA studio</td>
<td>Database tables, views, procedures; user-defined functions (UDF) and triggers; analytic objects; data authorization...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SAP HANA Web-based Workbench</td>
<td></td>
</tr>
<tr>
<td>Application developers:</td>
<td>XS JavaScript, OData, SQLScript,...</td>
<td>• SAP HANA studio</td>
<td>Control-flow logic, data services, calculation logic...</td>
</tr>
<tr>
<td>Professional (XS JS)</td>
<td></td>
<td>• SAP HANA Web-based Workbench</td>
<td></td>
</tr>
<tr>
<td>Casual/business</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UI/client developers</td>
<td>SAPUI5, JavaScript,...</td>
<td>• SAP HANA studio</td>
<td>UI shell, navigation, themes (look/feel), controls, events,...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SAP HANA Web-based Workbench</td>
<td></td>
</tr>
</tbody>
</table>

Related Information

Professional Application Development Scenarios [page 22]

1.2 Roles and Permissions for XS Development

An overview of the authorizations required to develop database artifacts for SAP HANA using the CDS syntax.

To enable application-developers to start building native applications that take advantage of the SAP HANA Extended Application Services (SAP HANA XS), the SAP HANA administrator must ensure that developers have access to the tools and objects that they need to perform the tasks required during the application- and database-development process.

Before you start developing applications using the features and tools provided by the SAP HANA XS, bear in mind the following prerequisites. Developers who want to build applications to run on SAP HANA XS need the following tools, accounts, and privileges:

- SAP HANA XS Classic Model [page 7]
- SAP HANA XS Advanced Model [page 8]
The required privileges can only be granted by someone who has the necessary authorizations in SAP HANA, for example, an SAP HANA administrator.

SAP HANA XS Classic Model

To develop database artifacts for use by applications running in the SAP HANA XS classic environment, bear in mind the following prerequisites:

- Access to a running SAP HANA development system (with SAP HANA XS classic)
- A valid user account in the SAP HANA database on that system
- Access to development tools, for example, provided in:
  - SAP HANA studio
  - SAP HANA Web-based Development Workbench
- Access to the SAP HANA repository
- Access to selected run-time catalog objects

To provide access to the repository for application developers, you can use a predefined role or create your own custom role to which you assign the privileges that the application developers need to perform the everyday tasks associated with the application-development process.

To provide access to the repository from the SAP HANA studio, the EXECUTE privilege is required for SYS.REPOSITORY_REST, the database procedure through with the REST API is tunneled. To enable the activation and data preview of information views, the technical user _SYS_REPO also requires SELECT privilege on all schemas where source tables reside.

In SAP HANA, you can use roles to assign one or more privileges to a user according to the area in which the user works; the role defines the privileges the user is granted. For example, a role enables you to assign SQL privileges, analytic privileges, system privileges, package privileges, and so on. To create and maintain artifacts in the SAP HANA repository, you can assign application-development users the following roles:

- One of the following:
  - MODELING
    The predefined MODELING role assigns wide-ranging SQL privileges, for example, on _SYS_BI and _SYS_BIC. It also assigns the analytic privilege _SYS_BI_CP_ALL, and some system privileges. If these permissions are more than your development team requires, you can create your own role with a set of privileges designed to meet the needs of the application-development team.
  - Custom DEVELOPMENT role
    A user with the appropriate authorization can create a custom DEVELOPMENT role specially for application developers. The new role would specify only those privileges an application-developer needs to perform the everyday tasks associated with application development, for example: maintaining packages in the repository, executing SQL statements, displaying data previews for views, and so on.
- PUBLIC
  This is a role that is assigned to all users by default.
Before you start using the SAP HANA Web-based Development Workbench, the SAP HANA administrator must set up a user account for you in the database and assign the required developer roles to the new user account.

**Tip**

The role `sap.hana.xs.ide.roles::Developer` grants the privileges required to use all the tools included in the SAP HANA Web-based Development Workbench. However, to enable a developer to use the debugging features of the browser-based IDE, your administrator must also assign the role `sap.hana.xs.debugger::Debugger`. In addition, the section `debugger` with the parameter `enabled` and the value `true` must be added to the file `xsengine.ini`, for example, in the SAP HANA studio Administration perspective.

**SAP HANA XS Advanced Model**

To develop database artifacts for use by applications running in the SAP HANA XS advanced environment, bear in mind the following prerequisites:

- Access to a running SAP HANA development system (with SAP HANA XS advanced)
- A valid user account in the SAP HANA database on that system
- Access to development tools, for example, provided in:
  - SAP Web IDE for SAP HANA
  - SAP HANA Run-time Tools (included in the SAP Web IDE for SAP HANA)
- Access to the SAP HANA XS advanced design-time workspace and repository
- Access to selected run-time catalog objects
- Access to the XS command-line interface (CLI); the XS CLI client needs to be downloaded and installed

**1.3 Developer Information Map**

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.

The following image displays essential information sources for people planning to develop applications in SAP HANA Extended Application Services classic model.

![Diagram](image)

**Figure 2: Application Development in SAP HANA XS Classic Model**

The following image displays the essential information sources for developing applications in SAP HANA Extended Application Services advanced model.
With such a wide variety of people needing to find such a broad range of information about so many different tasks, it is sometimes not easy to know what information is available or where to look to find it. This section is designed to help you navigate the documentation landscape by looking at the information available for the SAP HANA developer from the following perspectives:

- Information by developer guide
- Information by developer task
- Information by developer scenario

**Related Information**

SAP HANA Developer Information by Guide [page 11]
SAP HANA Developer Information by Task [page 15]
SAP HANA Developer Information by Scenario [page 17]
### 1.3.1 SAP HANA Developer Information by Guide

The design and organization of the SAP HANA developer documentation library makes it easy to use the name of a guide to find the relevant information. For example, the SAP HANA SQLScript Reference describes how to use the SQL extension SAP HANA SQLScript to define a data model.

The SAP HANA developer information set includes a selection of guides that describe the complete application-development process, from defining user roles, privileges, and data models through application setup to UI design and testing; the information available covers background and concepts, task-based tutorials, and detailed reference material. The following tables illustrate which guides are available for the developer who wants to build applications for SAP HANA, what information the guide contains, and which typical high-level tasks the selected guide covers:

**Table 2: Core SAP HANA Developer Guides**

<table>
<thead>
<tr>
<th>SAP HANA Guide</th>
<th>Description</th>
<th>Typical Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Developer Quick Start Guide (for XS classic)</td>
<td>Contains a selection of tutorials which teach the basic steps required to build data models and the native applications that run on SAP HANA XS classic model.</td>
<td>Build data-persistence models</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Write procedures and user-defined functions (UDF)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Build applications with SAP HANA XS JavaScript or OData.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Build a simple UI with SAPUI5</td>
</tr>
<tr>
<td>Developer Guide (for XS classic and SAP HANA studio)</td>
<td>Describes the complete application-development process for SAP HANA Extended Application Services Classic Model using the tools included in SAP HANA studio.</td>
<td>Build a data model</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Build XS classic applications (XS JavaScript)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Build SAPUI5 clients</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Manage the application lifecycle</td>
</tr>
<tr>
<td>Developer Guide (for XS classic and Web Workbench)</td>
<td>Describes the complete application-development process for SAP HANA Extended Application Services Classic Model using the tools included in SAP HANA Web-based Development Workbench.</td>
<td>Build a data model</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Build XS classic applications (XS JavaScript)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Build SAPUI5 clients</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Manage the application lifecycle</td>
</tr>
<tr>
<td>Developer Guide (for XS advanced)</td>
<td>Describes the complete application-development process for SAP HANA Extended Application Services Advanced Model.</td>
<td>Build a data model</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Build XS advanced applications (Node.js, JavaScript, Java, ...)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Build SAPUI5 clients</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Deploy applications to desired run-time environment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Manage the application lifecycle</td>
</tr>
</tbody>
</table>
The SAP HANA developer information set also includes a selection of reference guides that describe the various languages that you use to define the underlying data model (for example, SQL, CDS, or HDBTable) as well as the application business logic (for example, XS JavaScript with XS classic). The following tables illustrate which reference guides are available for the more experienced developer who needs more detailed information about how to build specific components.

Table 3: SAP HANA Language Reference Guides

<table>
<thead>
<tr>
<th>SAP HANA Guide</th>
<th>Description</th>
<th>Typical Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SQL System Views</strong></td>
<td>Describes all SQL data types, predicates, operators, expressions, functions, statements, and error codes.</td>
<td>Query state of SAP HANA using SQL commands</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alter system configuration/initialization services *</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Manage extended storage *</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Manage remote sources, subscriptions, adapters, tasks *</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Perform data analysis/mining *</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Manage data streams *</td>
</tr>
<tr>
<td><strong>SQLScript Reference</strong></td>
<td>Describes how to use the SQL extension SAP HANA SQLScript to embed data-intensive application logic into SAP HANA.</td>
<td>Build SQL scripts</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Create UDFs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Build SQL procedures</td>
</tr>
<tr>
<td>SAP HANA Guide</td>
<td>Description</td>
<td>Typical Tasks</td>
</tr>
<tr>
<td>----------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SAP HANA Analytics Catalog (BIMC Views) Reference</td>
<td>Describes how to use the SAP HANA analytics catalog for tables and views with the BIMC prefix. The catalog contains metadata used by analytics clients such as the Business Objects Cloud and for access to SAP HANA via MDX.</td>
<td>Pass and map variables and parameters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Use parameters in hierarchies</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Read values from BIMC tables</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Build SQL queries</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Build MDX queries</td>
</tr>
<tr>
<td>SQLScript Command Network Protocol</td>
<td>Describes the SQL Command Network Protocol that is used by SAP HANA clients to communicate with SAP HANA.</td>
<td>Define routes for SQL statements</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Set up authentication (SAML...)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Handle large data objects</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Enable distributed transactions</td>
</tr>
<tr>
<td>Spatial Reference(*)</td>
<td>Describes how to store, manipulate, and manage spatial data, for example, geographic locations, routing information, and shape data.</td>
<td>Store and manage spatial data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Access and manipulate spatial data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Calculate the distance between geometries</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Determine the union/intersection of multiple objects</td>
</tr>
<tr>
<td>XS JavaScript Reference</td>
<td>Describes how to use XS JavaScript to build native SAP HANA applications for XS classic, explains the underlying concepts, and lists the various APIs that are available.</td>
<td>Create XSJS services (for XS classic)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Create XSJS libraries</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Build application logic</td>
</tr>
<tr>
<td>XS JavaScript API Reference</td>
<td>Describes the API functions, methods, and classes provided for use with server-side JavaScript code running inside (SAP HANA XS).</td>
<td>Use the XS JavaScript API</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Search for XSJS API classes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Locate XSJS methods</td>
</tr>
<tr>
<td>XSUnit JavaScript API Reference</td>
<td>Describes the API functions, methods, and classes provided with the XSUnit test framework to automate the tests that you want to run for SAP HANA XS applications.</td>
<td>Test server-side (XS) JavaScript code</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Test SQLScript code (stored procedures, views)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Test modeled calculation view</td>
</tr>
<tr>
<td>XS DB Utilities JavaScript API Reference</td>
<td>Describes the API that provides access to a library of JavaScript utilities, which can be used to enable server-side JavaScript applications to consume data models that are defined using Core Data Services or call stored procedures as if they were JavaScript objects.</td>
<td>Call a stored procedure</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Query a CDS entity</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Update a CDS entity</td>
</tr>
<tr>
<td>SAP HANA Guide</td>
<td>Description</td>
<td>Typical Tasks</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
<td>---------------</td>
</tr>
</tbody>
</table>
| SINA Search JavaScript Reference * | Describes the SAP HANA simple information access (SINA) API, a client-side JavaScript API for developing browser-based search UIs | Create a search query  
Create a suggestion query  
Create a bar- or line-chart query  
Create a SAPUI5 bar chart |
| Core Data Services (CDS) Reference (for XS classic) | Explains how to use Core Data Services (CDS) to build design-time data-persistence models in SAP HANA Extended Application Services (for XS classic). The data-persistence model defines the data to expose in response to client requests via HTTP, for example, from an SAPUI5-based application. | Create CDS Documents (for XS classic)  
Define tables, table types, and SQL views  
Define associations between data objects  
Import data into a table |
| HDBTable Syntax Reference | Explains how to use the hdbtable syntax to build design-time data-persistence models in SAP HANA XS (for XS classic). The data-persistence model is used to define the data to expose in response to client requests via HTTP, for example, from an SAPUI5-based application. | Define schemas and sequences (for XS classic)  
Define tables, SQL views, and table types  
Import data into a table |
| SAP HANA REST API (for XS classic) | Describes the REST API for SAP HANA (for XS classic), which enables development tools to access SAP HANA platform components such as the for XS classic repository using REST-based calls. | Maintain repository workspaces  
Maintain projects and files  
Interact with the SAP HANA repository  
Access catalog objects |
| BFL Reference | Describes the SAP HANA Business Function Library (BFL), which contains pre-built financial functions. | Assign AFL user roles/privileges  
Create the AFL wrapper procedure  
Generate and call a PAL procedure |
| PAL Reference | Describes the SAP HANA Predictive Analysis Library (PAL), which contains functions that can be called from within SAP HANA SQLScript procedures to perform analytic algorithms. | Assign AFL user roles/privileges  
Create the AFL wrapper procedure  
Generate and call a BFL procedure  
Create input/output tables |

⚠️ Caution

(*) SAP HANA server software and tools can be used for several SAP HANA platform and options scenarios as well as the respective capabilities used in these scenarios. The availability of these is based on the available SAP HANA licenses and the SAP HANA landscape, including the type and version of the back-end systems the SAP HANA administration and development tools are connected to. There are several types of licenses available for SAP HANA. Depending on your SAP HANA installation license type, some of the features and tools described in the SAP HANA platform documentation may only be available in the SAP HANA options and capabilities, which may be released independently of an SAP HANA Platform Support...
Package Stack (SPS). Although various features included in SAP HANA options and capabilities are cited in the SAP HANA platform documentation, each SAP HANA edition governs the options and capabilities available. Based on this, customers do not necessarily have the right to use features included in SAP HANA options and capabilities. For customers to whom these license restrictions apply, the use of features included in SAP HANA options and capabilities in a production system requires purchasing the corresponding software license(s) from SAP. The documentation for the SAP HANA optional components is available in SAP Help Portal at http://help.sap.com/hana_options. If you have additional questions about what your particular license provides, or wish to discuss licensing features available in SAP HANA options, please contact your SAP account team representative.

1.3.2 SAP HANA Developer Information by Task

The design and organization of the SAP HANA developer documentation library enables easy access to information according to the particular development task to be performed, for example, creating a view or procedure, or setting up an application project.

The SAP HANA developer can make use of a large number of guides that include information describing the complete application-development process. The following figure shows the information that is available from the perspective of the development tasks that must be performed in a particular development area, for example, setting up the persistence model; creating an XSJS or OData service, or managing the development lifecycle. Each of the tasks described is supported by information covering the underlying concepts and detailed reference material. The figure also indicates where to find information based on the development task you want to perform. The tasks are split according to development area, for example, database development, application development, or UI design.
<table>
<thead>
<tr>
<th>SAP HANA Development Area</th>
<th>Typical Tasks</th>
<th>SAP HANA Guide</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Developer</td>
</tr>
<tr>
<td>Database</td>
<td>Set up the persistence model</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Set up the analytic model</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Create SQLScript procedures</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Create user-defined functions</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Create decision tables</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Set up lifecycle management</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Create full-text search</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Build SQL search queries</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Model spatial data</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Manage extended storage</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Perform data analysis/mining</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Manage data streams</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Manage remote sources</td>
<td>X</td>
</tr>
<tr>
<td>Applications</td>
<td>Set up an application</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Set up a project</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Create an OData/XMLA service</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Create an XSJS service</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Bind XS service to UI</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Set up lifecycle management</td>
<td>X</td>
</tr>
<tr>
<td>User Interface &amp; Clients</td>
<td>Set up SAPUI5 tools</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Create SAPUI5 apps</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Create UI views</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Define UI event handlers</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Bind data to a view</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Create UI widgets</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Define search UIs</td>
<td>X</td>
</tr>
<tr>
<td>Repository Access</td>
<td>Logon credentials</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Roles and privileges</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Application artifacts</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>Life-cycle management</td>
<td>X</td>
</tr>
</tbody>
</table>

**Figure 4: SAP HANA Information by Developer Task**

⚠️ Caution

(*) SAP HANA server software and tools can be used for several SAP HANA platform and options scenarios as well as the respective capabilities used in these scenarios. The availability of these is based on the available SAP HANA licenses and the SAP HANA landscape, including the type and version of the back-end systems the SAP HANA administration and development tools are connected to. There are several types of licenses available for SAP HANA. Depending on your SAP HANA installation license type, some of the
features and tools described in the SAP HANA platform documentation may only be available in the SAP HANA options and capabilities, which may be released independently of an SAP HANA Platform Support Package Stack (SPS). Although various features included in SAP HANA options and capabilities are cited in the SAP HANA platform documentation, each SAP HANA edition governs the options and capabilities available. Based on this, customers do not necessarily have the right to use features included in SAP HANA options and capabilities. For customers to whom these license restrictions apply, the use of features included in SAP HANA options and capabilities in a production system requires purchasing the corresponding software license(s) from SAP. The documentation for the SAP HANA optional components is available in SAP Help Portal at http://help.sap.com/hana_options. If you have additional questions about what your particular license provides, or wish to discuss licensing features available in SAP HANA options, please contact your SAP account team representative.

1.3.3 SAP HANA Developer Information by Scenario

The design and organization of the SAP HANA developer documentation library enables easy access to information according to the underlying development scenario, for example, lifecycle management, or application development.

The SAP HANA developer can make use of a large number of guides that include information describing the complete application-development process from the perspective of the development scenario, for example, database development, application development, or client UI design and testing; the information available covers background and concepts, task-based tutorials, and detailed reference material. The following table indicates where to find information based on the development scenario you choose, for example:

- Database Development [page 17]
- Application Development [page 19]
- UI Client Design [page 20]

The particular scenario you select can be based on the underlying development area you are assigned to, the choice of programming language, the required development objects, or the tools you want to use:

Database Development Scenarios

A database developer uses a variety of languages to develop a data model that can be exposed to a UI client, for example by HTTP (with an application service) or SQL (with a client interface such as ADBC or JDBC). In a database-development scenario, developers typically use languages such as SQLScript, the .hdtable syntax family (hdbview, hdbsequence,...), or Core Data Services (CDS) to define the data-persistence model to which you add the corresponding analytic model. If you want to develop a data model that can be exposed to client requests, use the following table to help find the information you need to complete the most common development tasks.

Using features available in additional optional components (*), you can develop applications that perform the following tasks:

- Manage data streams, for example, to filter, aggregate or enrich data before it is committed to the database
• Perform real-time operational analytics on data
• Enhance, cleanse, and transform data from local or remote sources to make it more accurate and useful
• Analyze and process geospatial information
• Analyze and mine both structured and unstructured textual data and interlinked structured data

Table 4: Information by Database-Development Scenario

<table>
<thead>
<tr>
<th>Programming Language</th>
<th>Development Artifacts</th>
<th>Tools</th>
<th>SAP HANA Information Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL</td>
<td>Database elements, services, policies, extended storage/index, adapters and agents, remote sources and subscriptions, search, text mining and analysis, data streams (filters aggregators, and enrichment), information management, geo-spatial data *</td>
<td>Eclipse (SAP HANA studio) SAP HANA Web Workbench SAP IDE for SAP HANA</td>
<td>SQL and System Views Reference Text Analysis Developer Guide * Text Mining Developer Guide * Smart Data Streaming Developer Guide *</td>
</tr>
<tr>
<td>SQLScript</td>
<td>Tables, SQL Views, Procedures, UDFs, application &amp; business functions...</td>
<td>Eclipse (SAP HANA studio) SAP HANA Web Workbench</td>
<td>SQLScript Reference BFL Reference PAL Reference</td>
</tr>
<tr>
<td>SAP HANA DB (.hdb * syntax; hdbtable, hdbview...)</td>
<td>Tables, SQL Views, Procedures, Search index (InA) ...</td>
<td>Eclipse (SAP HANA studio) SAP HANA Web Workbench</td>
<td>Developer Guide (XS classic) SQLScript Reference</td>
</tr>
<tr>
<td>SAP HANA DB (CDS syntax)</td>
<td>Entities, data types, contexts, SQL views, geo-spatial *, ...</td>
<td>Eclipse (SAP HANA studio) SAP HANA Web Workbench SAP IDE for SAP HANA</td>
<td>Developer Guide (XS classic) Developer Guide (for XS advanced) Spatial Reference</td>
</tr>
<tr>
<td>SAP HANA HDI</td>
<td>Tables, indexes, data types, procedures, SQL views, triggers, calculation views, analytic privileges, ...</td>
<td>SAP Web IDE for SAP HANA</td>
<td>Developer Guide (for XS advanced)</td>
</tr>
<tr>
<td>MDX</td>
<td>Analytics, BIMC tables and views</td>
<td>Eclipse (SAP HANA studio) SAP HANA Web Workbench SAP Web IDE for SAP HANA</td>
<td>SAP HANA Analytics Catalog (BIMC Views) Reference</td>
</tr>
</tbody>
</table>
Caution

(*) SAP HANA server software and tools can be used for several SAP HANA platform and options scenarios as well as the respective capabilities used in these scenarios. The availability of these is based on the available SAP HANA licenses and the SAP HANA landscape, including the type and version of the back-end systems the SAP HANA administration and development tools are connected to. There are several types of licenses available for SAP HANA. Depending on your SAP HANA installation license type, some of the features and tools described in the SAP HANA platform documentation may only be available in the SAP HANA options and capabilities, which may be released independently of an SAP HANA Platform Support Package Stack (SPS). Although various features included in SAP HANA options and capabilities are cited in the SAP HANA platform documentation, each SAP HANA edition governs the options and capabilities available. Based on this, customers do not necessarily have the right to use features included in SAP HANA options and capabilities. For customers to whom these license restrictions apply, the use of features included in SAP HANA options and capabilities in a production system requires purchasing the corresponding software license(s) from SAP. The documentation for the SAP HANA optional components is available in SAP Help Portal at http://help.sap.com/hana_options. If you have additional questions about what your particular license provides, or wish to discuss licensing features available in SAP HANA options, please contact your SAP account team representative.

Application Development Scenarios

As an application developer, you use a variety of languages to develop applications that expose a SAP HANA data model to requests from UI clients. In an application-development scenario, developers typically use languages such as server-side JavaScript (XSJS) or an OData service to define the application business model that exposes the data model built by the database developer. You can call the application service from a client interface, for example, a browser or UI client. If you want to develop an application service that exposes an SAP HANA data model to client requests, use the following table to help find the information you need to complete the most common development tasks.

Table 5: Information by Application-Development Scenario

<table>
<thead>
<tr>
<th>Programming Language</th>
<th>Development Artifacts</th>
<th>Tools</th>
<th>Information Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQLScript</td>
<td>SQLScript</td>
<td>Eclipse (SAP HANA studio)</td>
<td>Developer Guide</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SAP HANA Web Workbench</td>
<td>SQLScript Reference</td>
</tr>
<tr>
<td>XSJS (server-side Java-Script)</td>
<td>Server-side JavaScript services, libraries, API</td>
<td>Eclipse (SAP HANA studio)</td>
<td>Developer Guide (for XS classic)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SAP HANA Web Workbench</td>
<td>XS JavaScript API Reference</td>
</tr>
<tr>
<td>Node.js</td>
<td>Server-side services, modules, libraries, API</td>
<td>SAP Web IDE for SAP HANA</td>
<td>Developer Guide (for XS advanced)</td>
</tr>
<tr>
<td>Java</td>
<td>Server-side services, libraries, API</td>
<td>SAP Web IDE for SAP HANA</td>
<td>Developer Guide (for XS advanced)</td>
</tr>
<tr>
<td>OData</td>
<td>OData services, query options, parameters</td>
<td>Eclipse (SAP HANA studio)</td>
<td>Developer Guide</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SAP HANA Web IDE</td>
<td>OData Reference</td>
</tr>
</tbody>
</table>
UI Client Development Scenarios

As a developer of client applications, you use a variety of languages to develop a user interface (UI) client that displays permitted elements of an SAP HANA data model. In a UI-client development scenario, developers typically use languages such as SAPUI5 (HTML5) or JavaScript to define the UI client application. The UI client binds interface controls to actions that request data and display it in the required format. If you want to develop a UI client application that can be used to display an SAP HANA data model, use the following table to help find the information you need to complete the most common development tasks.

Table 6: Information by UI-Client Development Scenario

<table>
<thead>
<tr>
<th>Programming Language</th>
<th>Development Artifacts</th>
<th>Tools</th>
<th>Information Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAPUI5</td>
<td>JS, UI5 Library, View, Control, ...</td>
<td>Eclipse (SAP HANA studio)</td>
<td>Developer Guide</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SAPUI5 Demo Kit and Documentation</td>
</tr>
<tr>
<td>JavaScript</td>
<td>Search queries, results, suggestions</td>
<td>Eclipse (SAP HANA studio)</td>
<td>Developer Guide</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>SINA Search JavaScript Reference</td>
</tr>
</tbody>
</table>

Caution

(*) SAP HANA server software and tools can be used for several SAP HANA platform and options scenarios as well as the respective capabilities used in these scenarios. The availability of these is based on the available SAP HANA licenses and the SAP HANA landscape, including the type and version of the back-end systems the SAP HANA administration and development tools are connected to. There are several types of licenses available for SAP HANA. Depending on your SAP HANA installation license type, some of the features and tools described in the SAP HANA platform documentation may only be available in the SAP HANA options and capabilities, which may be released independently of an SAP HANA Platform Support Package Stack (SPS). Although various features included in SAP HANA options and capabilities are cited in the SAP HANA platform documentation, each SAP HANA edition governs the options and capabilities available. Based on this, customers do not necessarily have the right to use features included in SAP HANA options and capabilities. For customers to whom these license restrictions apply, the use of features included in SAP HANA options and capabilities in a production system requires purchasing the corresponding software license(s) from SAP. The documentation for the SAP HANA optional components is available in SAP Help Portal at [http://help.sap.com/hana_options](http://help.sap.com/hana_options). If you have additional questions about what your particular license provides, or wish to discuss licensing features available in SAP HANA options, please contact your SAP account team representative.

1.3.3.1 Developer Scenarios

The possibility to run application specific code in SAP HANA creates several possibilities for developing SAP HANA based applications, representing various integration scenarios, and corresponding development processes.

Application developers can choose between the following scenarios when designing and building applications that access an SAP HANA data model:
Native Application Development
Native applications are developed and run in SAP HANA, for example, using just SQLScript or the extended application services provided by the SAP HANA XS platform (or both).

Non-native Application Development
Non-native applications are developed in a separate, external environment (for example, ABAP or Java) and connected to SAP HANA by means of an external application server and a client connection: ADBC, JDBC, ODBC, or ODBO. These more traditional scenarios only use SQL and native SQLScript procedures.

The following diagram shows the scope of the languages and the environment you use in the various phases of the process of developing applications that harness the power of SAP HANA. For example, if you are developing native SAP HANA applications you can use CDS, HDBtable, or SQLScript to create design-time representations of objects that make up your data persistence model; you can use server-side JavaScript (XSJS) or OData services to build the application’s business logic; and you can use SAPUI5 to build client user interfaces that are bound to the XSJS or OData services.

If you are developing non-native SAP HANA applications, you can choose between any of the languages that can connect by means of the client interfaces that SAP HANA supports, for example, ABAP (via ADBC) or Java (JDBC).
1.3.3.2 Professional Application Development Scenarios

The primary focus of the professional application developer is to create applications.

The professional application developer creates server-side applications that define the business logic required to serve client requests, for example, for data created and exposed by the database developer. The following table lists some of the tasks typically performed by the professional application developer and indicates where to find the information that is required to perform the task.

Table 7: Typical Application-Development Tasks

<table>
<thead>
<tr>
<th>Task</th>
<th>Details</th>
<th>Information Source</th>
</tr>
</thead>
</table>
| Create an XSJS service:  
  - Extract data from SAP HANA  
  - Control application response  
  - Bind to a UI control/event | Context, examples, libraries, debugging, implementation, ... | Developer Guide |
| Create an OData service (for example, to bind a UI control/event to existing data tables or views) | Context, service syntax, examples, libraries, debugging, implementation, ... | Developer Guide |

The XS JavaScript Reference
Getting Started with XS JavaScript
<table>
<thead>
<tr>
<th>Task</th>
<th>Details</th>
<th>Information Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Query options, syntax...</td>
<td>OData Reference</td>
<td></td>
</tr>
<tr>
<td>UI controls, events...</td>
<td>SAPUI5 Demo Kit (version 1.28)</td>
<td></td>
</tr>
</tbody>
</table>
2 Writing Server-Side JavaScript Code

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. Applications written in server-side JavaScript can make use of a powerful set of specially developed API functions, for example, to enable access to the current request session or the database. This section describes how to write server-side JavaScript code that enables you to expose data, for example, using a Web Browser or any other HTTP client.

2.1 Data Access with JavaScript in SAP HANA XS

In SAP HANA Extended Application Services, the persistence model (for example, tables, views and stored procedures) is mapped to the consumption model that is exposed via HTTP to clients - the applications you write to extract data from SAP HANA.

You can map the persistence and consumption models in the following way:

- **Application-specific code**
  Write code that runs in SAP HANA application services. Application-specific code (for example, server-side JavaScript) is used in SAP HANA application services to provide the consumption model for client applications.

Applications running in SAP HANA XS enable you to accurately control the flow of data between the presentational layer, for example, in the 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 a server-side JavaScript application running in SAP HANA XS, you can take advantage of the embedded access to SAP HANA that SAP HANA XS provides; the embedded access greatly improves end-to-end performance.
2.2 Using Server-Side JavaScript in SAP HANA XS

SAP HANA application services (XS server) supports server-side application programming in JavaScript. The server-side application you develop can use a collection of JavaScript APIs to expose authorized data to client requests, for example, to be consumed by a client GUI such as a Web browser or any other HTTP client.

The functions provided by the JavaScript APIs enable server-side JavaScript applications not only to expose data but to update, insert, and delete data, too. You can use the JavaScript APIs to perform the following actions:

- Interact with the SAP HANA XS runtime environment
- Directly access SAP HANA database capabilities
- Interact with services on defined HTTP destinations.

JavaScript programs are stored in the repository along with all the other development resources. When the programs are activated, the code is stored in the repository as a runtime object.

**Tip**

To enable the Web Browser to display more helpful information if your JavaScript code causes an HTTP 500 exception on the SAP HANA XS Web server, ask someone with administrator privileges to start the SAP HANA studio’s Administration Console perspective and add the parameter developer_mode to the xsengine.ini httpserver section of the Configuration tab and set it to true.

### Related Information

- Write XS Server-Side JavaScript [page 25]
- JavaScript Security Considerations [page 28]

2.2.1 Tutorial: Write Server-Side JavaScript Application Code

SAP HANA Extended Application Services (SAP HANA XS) supports server-side application programming in JavaScript. The server-side application you develop uses JavaScript APIs to expose authorized data to client requests, for example, for consumption by a client GUI such as a Web browser, SAPUI5 applications, or mobile clients.

### Prerequisites

- Access to a running SAP HANA system.
- Access to SAP HANA studio
- Access to an SAP HANA Repository workspace
• Access to a shared project in the SAP HANA Repository where you can create the artifacts required for this tutorial.

Context

Since JavaScript programs are stored in the SAP HANA Repository, the steps in this task description assume that you have already created a workspace and a project (of type **XS Project**), and that you have shared the project with other members of the development team. To write a server-side JavaScript application, you must perform the following high-level steps.

**Tip**

File extensions are important. If you are using SAP HANA studio to create artifacts in the SAP HANA Repository, the file-creation wizard adds the required file extension automatically and, if appropriate, enables direct editing of the new file in the corresponding editor.

Procedure

1. Create a root package for your application, for example, `helloxsjs`.
2. Create an application descriptor for your application and place it in the root package you created in the previous step.

   The application descriptor is the core file that you use to describe an application's availability within SAP HANA XS. The application-descriptor file has no contents and no name; it only has the file extension `.xsapp`.

   a. In the **Project Explorer** view, right-click the folder where you want to create the new application descriptor and choose **New** ➤ **Other** ➤ **SAP HANA** ➤ **Application Development** ➤ **XS Application Descriptor File** in the context-sensitive popup menu.

   b. Save the application-descriptor file.

   c. Activate the application-descriptor file in the repository.

      Locate and right-click the new application-descriptor file in the **Project Explorer** view. In the context-sensitive pop-up menu, choose **Team** ➤ **Activate**.
3. Create an application-access file and place it in the package to which you want to grant access.
   The application-access file does not have a name; it only has the file extension `.xsaccess`. The contents of the `.xsaccess` file must be formatted according to JavaScript Object Notation (JSON) rules and associated with the package the file belongs to. The rules defined in the `.xsaccess` file apply to the package it resides in as well as any subpackages lower in the package hierarchy.
   a. In the Project Explorer view, right-click the folder where you want to create the new application-access file and choose [New Other SAP HANA Application Development XS Application Access File] in the context-sensitive popup menu.
   b. Enter the following content in the `.xsaccess` file for your new XSJS application:

   ```json
   {
     "exposed": true,
     "authentication": {
       "method": "Form"
     },
     "prevent_xsrfs": true
   }
   ```

   **Note**

   These settings allows data to be exposed, require logon authentication to access the exposed data, and help protect against cross-site request-forgery (XSRF) attacks.
   c. Save and activate the application-access file in the repository.

4. Create the server-side JavaScript (XSJS) files that contain the application logic.
   Server-side JavaScript files have the file suffix `.xsjs`, for example, `hello.xsjs` and contain the code that is executed when SAP HANA XS handles a URL request.
   a. In the Project Explorer view, right-click the folder where you want to create the new XSJS file and choose [New Other SAP HANA Application Development XS JavaScript File] in the context-sensitive popup menu.
   b. Using the wizard, enter the following content in the `.xsjs` file for your new XSJS application:

   ```javascript
   $.response.contentType = "text/plain";
   $.response.setBody( "Hello, World!" );
   ```
   c. Save and activate the XSJS file in the repository.

5. Check the layout workspace.
   Your application package structure should have a structure that looks like the following example:

   ```
   \\helloxsjs
   \xsapp
   .xsaccess
   .xsprivileges  // optional
   hello.xsjs
   ```

6. Save and activate the changes and additions you made.

7. View the results in a Web browser.
   The SAP HANA XS Web server enables you to view the results immediately after activation in the repository, for example: `http://<SAP_HANA_hostname>:80<DB_Instance_Number>/helloxsjs/hello.xsjs`
2.2.1.1 JavaScript Editor

You can write server-side JavaScript using the SAP HANA studio JavaScript editor, which provides syntax validation, code highlighting and code completion.

The SAP HANA studio’s JavaScript editor includes the JSLint open-source library, which helps to validate JavaScript code. The editor highlights any code that does not conform to the JSLint standards.

To configure the JSLint library and determine which validations are performed, go to: Window ▶ Preferences ▶ SAP HANA ▶ Application Development ▶ JSLint. In the preferences window, each JSLint setting is followed by the corresponding JSLint command name, which you can use to lookup more information on the JSLint Web site.

Tip
To disable all JSLint validations for files in a specific project, right-click the project and choose Disable JSLint.

Related Information

http://www.jslint.com/lint.html

2.2.1.2 Server-Side JavaScript Security Considerations

If you choose to use server-side JavaScript to write your application code, you need to bear in mind the potential for (and risk of) external attacks such as cross-site scripting and forgery, and insufficient authentication.

The following list illustrates the areas where special attention is required to avoid security-related problems when writing server-side JavaScript. Each of the problems highlighted in the list is described in detail in its own dedicated section:

- SSL/HTTPS
  Enable secure HTTP (HTTPS) for inbound communication required by an SAP HANA application.
- Injection flaws
  In the context of SAP HANA Extended Application Services (SAP HANA XS) injection flaws concern SQL injection that modifies the URL to expand the scope of the original request.
- Cross-site scripting (XSS)
  Web-based vulnerability that involves an attacker injecting JavaScript into a link with the intention of running the injected code on the target computer.
- Broken authentication and session management
  Leaks or flaws in the authentication or session management functions allow attackers to impersonate users and gain access to unauthorized systems and data.
- Insecure direct object references
  An application lacks the proper authentication mechanism for target objects.
- Cross-site request forgery (XSRF)
  Exploits the trust boundaries that exist between different Web sites running in the same web browser session.
- Incorrect security configuration
  Attacks against the security configuration in place, for example, authentication mechanisms and authorization processes.
- Insecure cryptographic storage
  Sensitive information such as logon credentials is not securely stored, for example, with encryption tools.
- Missing restrictions on URL Access
  Sensitive information such as logon credentials is exposed.
- Insufficient transport layer protection
  Network traffic can be monitored, and attackers can steal sensitive information such as logon credentials or credit-card data.
- Invalid redirects and forwards
  Web applications redirect users to other pages or use internal forwards in a similar manner.
- XML processing issues
  Potential security issues related to processing XML as input or to generating XML as output

Related Information

SAP HANA Security Guide
SAP HANA SQL and System Views Reference
SSL/HTTPS [page 30]
Injection flaws [page 30]
Cross-site scripting (XSS) [page 32]
Broken authentication and session management [page 32]
Insecure direct object references [page 33]
Cross-site request forgery (XSRF) [page 34]
Incorrect security configuration [page 36]
Insecure cryptographic storage [page 36]
Missing restrictions on URL Access [page 37]
Insufficient transport layer protection [page 38]
XML processing issues [page 39]
2.2.1.2.1 Server-Side JavaScript: SSL/HTTPS

If you choose to use server-side JavaScript to write your application code, you need to bear in mind the potential for (and risk of) external attacks such as cross-site scripting and forgery, and insufficient authentication. You can set up SAP HANA to use secure HTTP (HTTPS).

SSL/HTTPS Problem

Incoming requests for data from client applications use secure HTTP (HTTPS), but the SAP HANA system is not configured to accept the HTTPS requests.

SSL/HTTPS Recommendation

Ensure the SAP Web Dispatcher is configured to accept incoming HTTPS requests. For more information, see the SAP HANA Security Guide.

**Note**

The HTTPS requests are forwarded internally from the SAP Web Dispatcher to SAP HANA XS as HTTP (clear text).

Related Information

SAP HANA Security Guide

2.2.1.2.2 Server-Side JavaScript: Injection Flaws

If you choose to use server-side JavaScript to write your application code, you need to bear in mind the potential for (and risk of) injection flaws. Typically, injection flaws concern SQL injection and involve modifying the URL to expand the scope of the original request.

The XS JavaScript API provides a number of different ways to interact with the SAP HANA database by using SQL commands. By default, these APIs allow you to read data, but they can also be used to update or delete data, and even to grant (or revoke) access rights at runtime. As a general rule, it is recommended to write a query which is either a call to an SQLScript procedure or a prepared statement where all parameters specified in the procedure or statement are escaped by using either `setString` or `setInt`, as illustrated in the examples provided in this section. Avoid using dynamic SQL commands with parameters that are not escaped.
Injection Flaws Problem

In the context of SAP HANA XS, injection flaws mostly concern SQL injection, which can occur in the SAP HANA XS JavaScript API or SQL script itself (both standard and dynamic). For example, the URL http://xsengine/customer.xsjs?id=3 runs the code in the JavaScript file customer.xsjs shown below:

```javascript
var conn = $.db.getConnection();
var pstmt = conn.prepareStatement("SELECT * FROM accounts WHERE custID=" + $.request.parameters.get("id"));
var rs = pstmt.executeQuery();
```

By modifying the URL, for example, to http://xsengine/customer.xsjs?id=3 'OR 1=1', an attacker can view not just one account but all the accounts in the database.

**Note**

SAP HANA XS applications rely on the authorization provided by the underlying SAP HANA database.

Users accessing an SAP HANA XS based application require the appropriate privileges on the database objects to execute database queries. The SAP HANA authorization system will enforce the appropriate authorizations. This means that in those cases, even if the user can manipulate a query, he will not gain more access than is assigned to him through roles or privileges. Definer mode SQL script procedures are an exception to this rule that you need to take into consideration.

Injection Flaws Recommendation

To prevent injection flaws in the JavaScript API, use prepared statements to create a query and place-holders to fill with results of function calls to the prepared-statement object; to prevent injection flaws in standard SQL Script, use stored procedures that run in caller mode; in caller mode, the stored procedures are executed with the credentials of the logged-on HANA user. Avoid using dynamic SQL if possible. For example, to guard against the SQL-injection attack illustrated in the problem example, you could use the following code:

```javascript
var conn = $.db.getConnection();
var pstmt = conn.prepareStatement("SELECT * FROM accounts WHERE custID=?");
pstmt.setInt(1, $.request.parameters.get("id"), 10);
var rs = pstmt.executeQuery();
```

Prepared statements enable you to create the actual query you want to run and then create several placeholders for the query parameters. The placeholders are replaced with the proper function calls to the prepared statement object. The calls are specific for each type in such a way that the SAP HANA XS JavaScript API is able to properly escape the input data. For example, to escape a string, you can use the setString function.

Related Information

- SAP HANA Security Guide
- SAP HANA SQL and System Views Reference
2.2.1.2.3  Server-Side JavaScript: Cross-Site Scripting

If you use server-side JavaScript to write your application code, bear in mind the potential for (and risk of) cross-site scripting (XSS) attacks. Cross-site scripting is a Web-based vulnerability that involves an attacker injecting JavaScript into a link with the intention of running the injected code on the target computer.

**Cross-Site Scripting Problem**

The vulnerability to cross-site scripting attacks comes in the following forms:

- **Reflected (non-persistent)**
  
  Code affects individual users in their local Web browser

- **Stored (persistent)**
  
  Code is stored on a server and affects all users who visit the served page

A successful cross-site scripting attack could result in a user obtaining elevated privileges or access to information that should not be exposed.

**Cross-Site Scripting Recommendation**

Since there are currently no libraries provided by the standard SAP HANA XS JavaScript API to provide proper escaping, the best solution for generating HTML on SAP HANA XS is to use the ESAPI JavaScript libraries as a starting point. In addition, we recommend not to write custom interfaces but to rely on well-tested technologies supplied by SAP, for example, OData or JSON together with SAPUI5 libraries.

**Related Information**

[ SAP HANA Security Guide ]

2.2.1.2.4  Server-Side JavaScript: Broken Authentication

If you choose to use server-side JavaScript to write your application code, you need to bear in mind the potential for (and risk of) attack against authentication infrastructure. Leaks or flaws in the authentication or
session management functions allow attackers to impersonate users and gain access to unauthorized systems and data.

Authentication Problem

Leaks or flaws in the authentication or session management functions allow attackers to impersonate users; the attackers can be external as well as users with their own accounts to obtain the privileges of those users they impersonate.

Authentication Recommendation

Use the built-in SAP HANA XS authentication mechanism and session management (cookies). For example, use the "authentication" keyword to enable an authentication method and set it according to the authentication method you want implement, for example: SAP logon ticket, form-based, or basic (user name and password) in the application’s .xsaccess file, which ensures that all objects in the application path are available only to authenticated users.

Related Information

SAP HANA Security Guide

2.2.1.2.5 Server-Side JavaScript: Insecure Object Reference

If you choose to use server-side JavaScript to write your application code, you need to bear in mind the potential for (and risk of) attacks using insecure references to objects.

Object Reference Problem

An SAP HANA XS application is vulnerable to insecure direct object reference if the application lacks the proper authentication mechanism for target objects.

Object Reference Recommendation

Make sure that only authenticated users are allowed to access a particular object. In the context of SAP HANA XS, use the "authentication" keyword to enable an authentication method and set it according to the
authentication method you implement, for example: SAP logon ticket, form-based, or basic (user name and password) in the application's .xsaccess file, which ensures that all objects in the application path are available only to authenticated users.

Related Information

SAP HANA Security Guide

2.2.1.2.6 Server-Side JavaScript: Cross-Site Request Forgery

If you choose to use server-side JavaScript to write your application code, you need to bear in mind the potential for (and risk of) cross-site request forgery (XSRF). Cross-site scripting is a web-based vulnerability that exploits the trust boundaries that exist between different websites running in the same web browser session.

Cross-Site Request-Forgery Problem

Since there are no clear trust boundaries between different Web sites running in the same Web-browser session, an attacker can trick users (for example, by luring them to a popular Web site that is under the attacker's control) into clicking a specific hyperlink. The hyperlink displays a Web site that performs actions on the visitor's behalf, for example, in a hidden iframe. If the targeted end user is logged in and browsing using an account with elevated privileges, the XSRF attack can compromise the entire Web application.

Cross-Site Request-Forgery Recommendation

SAP HANA XS provides a way to include a random token in the POST submission which is validated on the server-side. Only if this token is non-predictable for attackers can one prevent cross-site, request-forgery attacks. The easiest way to prevent cross-site, request-forgery attacks is by using the standard SAP HANA XS cookie. This cookie is randomly and securely generated and provides a good random token which is unpredictable by an attacker ($.session.getSecurityToken()).

To protect SAP HANA XS applications from cross-site request-forgery (XSRF) attacks, make sure you always set the prevent_xsr$ keyword in the application-access (.xsaccess) file to true, as illustrated in the following example:

```javascript
{
    "prevent_xsr$" : true
}
```

The prevent_xsr$ keyword prevents the XSRF attacks by ensuring that checks are performed to establish that a valid security token is available for given Browser session. The existence of a valid security token determines
if an application responds to the client’s request to display content. A security token is considered to be valid if it matches the token that SAP HANA XS generates in the backend for the corresponding session.

### Note

The default setting is false, which means there is no automatic prevention of XSRF attacks. If no value is assigned to the `prevent_xsr` keyword, the default setting (false) applies.

The following client-side JavaScript code snippet show how to use the HTTP request header to fetch, check, and apply the XSRF security token required to protect against XSRF attacks.

```html
<html>
<head>
  <title>Example</title>
  <script id="sap-ui-bootstrap" type="text/javascript"
    src="/sap/ui5/1/resources/sap-ui-core.js"
    data-sap-ui-language="en"
    data-sap-ui-theme="sap_goldreflection"
    data-sap-ui-libs="sap.ui.core,sap.ui.commons,sap.ui.ux3,sap.ui.table">
  </script>
  <script type="text/javascript" src="/sap/ui5/1/resources/jquery-sap.js"></script>
  <script>
    function doSomething() {
      $.ajax({
        url: "logic.xsjs",
        type: "GET",
        beforeSend: function(xhr) {
          xhr.setRequestHeader("X-CSRF-Token", "Fetch");
        },
        success: function(data, textStatus, XMLHttpRequest) {
          var token = XMLHttpRequest.getResponseHeader('X-CSRF-Token');
          var data = "somePayload";
          $.ajax({
            url: "logic.xsjs",
            type: "POST",
            data: data,
            beforeSend: function(xhr) {
              xhr.setRequestHeader("X-CSRF-Token", token);
            },
            success: function() {
              alert("works");
            },
            error: function() {
              alert("works not");
            }
          });
        }
      });
    }
  </script>
</head>
<body>
  <a href="#" onClick="doSomething();">Do something</a>
</body>
</html>
```
Related Information

SAP HANA Security Guide

2.2.1.2.7 Server-Side JavaScript: Security Misconfiguration

If you choose to use server-side JavaScript to write your application code, you need to bear in mind the potential for (and risk of) attacks against the security configuration in place, for example, authentication mechanisms and authorization processes.

Insecure Configuration Problem

No or an inadequate authentication mechanism has been implemented.

Insecure Configuration Recommendation

Applications should have proper authentication in place, for example, by using SAP HANA built-in authentication mechanisms and, in addition, the SAP HANA XS cookie and session handling features. Application developers must also consider and control which paths are exposed by HTTP to the outside world and which of these paths require authentication.

Related Information

SAP HANA Security Guide

2.2.1.2.8 Server-Side JavaScript: Insecure Storage

If you choose to use server-side JavaScript to write your application code, you need to bear in mind the potential for (and risk of) attacks against the insecure or lack of encryption of data assets.

Storage-Encryption Problem

Sensitive information such as logon credentials is exposed.
Storage-Encryption Recommendation

To prevent unauthorized access, for example, in the event of a system break-in, data such as user logon credentials must be stored in an encrypted state.

Related Information

SAP HANA Security Guide

2.2.1.2.9 Server-Side JavaScript: Missing URL Restrictions

If you choose to use server-side JavaScript to write your application code, you need to bear in mind the potential for (and risk of) unauthorized access to URLs.

URL Access Problem

Unauthenticated users have access to URLs that expose confidential (unauthorized) data.

URL Access Recommendation

Make sure you have addressed the issues described in "Broken Authentication and Session Management" and "Insecure Direct Object References". In addition, check if a user is allowed to access a specific URL before actually executing the code behind that requested URL. Consider putting an authentication check in place for each JavaScript file before continuing to send any data back to the client's Web browser.

Related Information

SAP HANA Security Guide
2.2.1.2.10 Server-Side JavaScript: Transport Layer Protection

If you choose to use server-side JavaScript to write your application code, you need to bear in mind the potential for (and risk of) insufficient protection of the transport layer.

Transport Layer Protection Problem

Without transport-layer protection, the user’s network traffic can be monitored, and attackers can steal sensitive information such as logon credentials or credit-card data.

Transport Layer Protection Recommendation

Turn on transport-layer protection in SAP HANA XS; the procedure is described in the SAP HANA security guide.

Related Information

SAP HANA Security Guide

2.2.1.2.11 Server-Side JavaScript: Invalid Redirection

If you use server-side JavaScript to write your application code, bear in mind the potential for (and risk of) redirection and internal forwarding from the requested Web page.

Invalid Redirection Problem

Web applications frequently redirect users to other pages or use internal forwards in a similar manner. Sometimes the target page is specified in an invalid (not permitted) parameter. This enables an attacker to choose a destination page leading to the possibility of phishing attacks or the spamming of search engines.
Invalid Redirection Recommendation

To prevent invalidated redirects or forwards, application developers should validate the requested destination before forwarding, for example, by checking if the destination is present in a white list. If the destination URL specified in the redirection request is not present in the white list, the redirection is refused.

Tip

Avoid using redirection if you cannot control the final destination.

Alternatively, you can refuse to allow any direct user input; instead, the input can be used to determine the final destination for the redirection, as illustrated in the following example:

```javascript
var destination = $.request.parameters.get("dest");
switch (destination) {
    case "1": $.response.headers.set("location", "http://FirstWhitelistedURL.com"); break;
    case "2": $.response.headers.set("location", "http://SecondWhitelistedURL.com"); break;
    default: $.response.headers.set("location", "http://DefaultWhitelistedURL.com");
}
```

Related Information

SAP HANA Security Guide

2.2.1.2.12 Server-Side JavaScript: XML Processing Issues

If you choose to use server-side JavaScript to write your application code, you need to bear in mind the potential for (and risk of) attacks aimed at the process used to parse XML input and generate the XML output.

XML Processing Problem

There are several potential security issues related to processing XML as input or to generating XML as output. In addition, problems with related technologies (for example, XSL Transformations or XSLT) can enable the inclusion of other (unwanted) files.

XML Processing Recommendation

Turn on transport-layer protection in SAP HANA XS; the procedure is described in the SAP HANA security guide.
Bear in mind the following rules and suggestions when processing or generating XML output:

- When processing XML that originates from an untrusted source, disable DTD processing and entity expansion unless strictly required. This helps prevent Billion Laugh Attacks (Cross-Site Request Forgery), which can bring down the processing code and, depending on the configuration of the machine, an entire server.
- To prevent the inclusion (insertion) of unwanted and unauthorized files, restrict the ability to open files or URLs even in requests included in XML input that comes from a trusted source. In this way, you prevent the disclosure of internal file paths and internal machines.
- Ensure proper limits are in place on the maximum amount of memory that the XML processing engine can use, the amount of nested entities that the XML code can have, and the maximum length of entity names, attribute names, and so on. This practice helps prevent the triggering of potential issues.

Related Information

SAP HANA Security Guide

2.3 Using Server-Side JavaScript Libraries

The elements defined in normal server-side JavaScript programs cannot be accessed from other JavaScript programs. To enable the reuse of program elements, SAP HANA Extended Application Services support server-side JavaScript libraries.

Server-side JavaScript libraries are a special type of JavaScript program that can be imported and called in other JavaScript programs. You can use JavaScript libraries to perform simple, repetitive tasks, for example, to handle forms and form data, to manipulate date and time strings, to parse URLs, and so on.

Note

JavaScript libraries are internally developed extensions for SAP HANA.

The following example shows how to import a JavaScript mathematics library using the import function:

```javascript
// import math lib
$.import("sap.myapp.lib","math");
// use math lib
var max_res = $.sap.myapp.lib.math.max(3, 7);
```

The import function requires the following parameters:

- **Package name**
  Full name of the package containing the library object you want to import, for example, `sap.myapp.lib`

- **Library name**
  Name of the library object you want to import, for example, `math`
Note

Restrictions apply to the characters you can use in the names of JavaScript libraries and application packages. Permitted characters are: upper- and lower-case letters (Aa-Zz), digits 0-9, and the dollar sign ($).

The standard JavaScript limitations apply to the characters you can use in either the name of the XSJS library you create or the name of the package where the library is deployed. For example, you cannot use the hyphen (-) in the name of an XSJS library or, if you are referencing the library, the name of a package in the application package path. To prevent problems with activation of the object in the SAP HANA repository, you must follow the standard rules for accessing JavaScript property objects by name. The following example, shows how to use square brackets and quotes ("<STRING>") to access an object whose name uses non-permitted characters such as a hyphen (-):

```javascript
// import math lib
$.import("sap.myapp.lib.XS-QGP-SPS7","math");
// use math lib
var max_res = $.sap.myapp.lib["XS-QGP-SPS7"].math.max(3, 7);
```

Related Information

Import Server-Side JavaScript Libraries [page 41]
Write Server-Side JavaScript Libraries [page 43]

2.3.1 Import Server-Side JavaScript Libraries

Server-side JavaScript libraries are a special type of JavaScript program that can be imported and called in other JavaScript programs. You can use JavaScript libraries to perform simple, repetitive tasks, for example: handle forms and form date, manipulate date and time strings, parse URLs, and so on.

Context

JavaScript libraries are internally developed extensions for SAP HANA. The libraries exist in the context of a package, which is referenced when you import the library. The following example of a JavaScript library displays the word "Hello" along with a name and an exclamation mark as a suffix.

```javascript
var greetingPrefix = "Hello, ";
var greetingSuffix = "!";
function greet (name) {
    return greetingPrefix + name + greetingSuffix;
}
```
Note
This procedure uses the illustrated example JavaScript library to explain what happens when you import a JavaScript library, for example, which objects are created, when, and where. If you have your own library to import, substitute the library names and paths shown in the steps below as required.

To import a JavaScript library for use in your server-side JavaScript application, perform the following tasks

**Procedure**

1. Import the JavaScript library into a JavaScript application.
   
   Open the server-side JavaScript file into which you want to import the JavaScript library.
   
   Use the $.import function, as follows:

   ```
   $.import("<path.to.your.library.filename>","greetLib");
   var greeting = $.<path.to.your.library.filename>.greet("World");
   $.response.setBody(greeting);
   ```

2. Save and activate the changes to the JavaScript file.
   
   Although the operation is simple, bear in mind the following points:
   
   - **Additional objects in the package hierarchy**
     The import operation generates a hierarchy of objects below $ that resemble the library’s location in the repository, for example, for the library `path/to/your/library/greetLib.xsjslib`, you would see the following additional object:

   ```
   $.path.to.your.library.greetLib
   ```

   - **Additional properties for the newly generated library object:**

   ```
   $.path.to.your.library.greetLib.greet()
   $.path.to.your.library.greetLib.greetingSuffix
   $.path.to.your.library.greetLib.greetingPrefix
   ```

   - **Pre-import checks:**
     - It is not possible to import the referenced library if the import operation would override any predefined runtime objects.
     - Do not import the referenced library if it is already present in the package.

   - **Library context**
     Imported libraries exist in the context defined by their repository location.
2.3.2 Write Server-Side JavaScript Libraries

Server-side JavaScript libraries are a special type of JavaScript program that can be imported and called in other JavaScript programs. You can use JavaScript libraries to perform simple, repetitive tasks, for example, to handle forms and form date, to manipulate date and time strings, to parse URLs, and so on.

Context

JavaScript libraries are internally developed extensions for SAP HANA. However, you can write your own libraries, too. JavaScript libraries exist in the context of a package, which is referenced when you import the library. To write a JavaScript library to use in your server-side JavaScript application, perform the following steps:

Procedure

1. Create the file that contains the JavaScript library you want to add to the package and make available for import.

   In SAP HANA XS, server-side JavaScript libraries have the file extension .xsjslib, for example greetLib.xsjslib.

   a. In the Project Explorer view, right-click the folder where you want to create the new XSJS file and choose New Other SAP HANA Application Development XS JavaScript Library File in the context-sensitive popup menu.

   b. Type a name for the new XS JavaScript library file, for example greetLib and choose Finish.

   If you are using SAP HANA studio to create artifacts in the SAP HANA Repository, the file creation wizard adds a separator (.) and the required file extension automatically, for example, .xsjslib.

   c. Enter the following content in the greetLib.xsjslib XSJS library file for your new XSJS application. The following example creates a simple library that displays the word “Hello” along with a supplied name and adds an exclamation point (!) as a suffix.

   ```javascript
   var greetingPrefix = "Hello, ";
   var greetingSuffix = "!";
   function greet (name) {
     return greetingPrefix + name + greetingSuffix;
   }
   ```

2. Save the new JavaScript library.

   It is important to remember where the JavaScript library is located; you have to reference the package path when you import the library.

3. Activate your new library in the repository so that it is available for import by other JavaScript applications.
2.4 Using the Server-Side JavaScript APIs

SAP HANA Extended Application Services (SAP HANA XS) provides a set of server-side JavaScript application programming interfaces (API) that enable you to configure your applications to interact with SAP HANA.

The SAP HANA XS JavaScript Reference lists all the functions that are available for use when programming interaction between your application and SAP HANA. For example, you can use the database API to invoke SQL statements from inside your application, or access details of the current HTTP request for SAP HANA data with the request-processing API. SAP HANA XS includes the following set of server-side JavaScript APIs:

Table 8: XS JavaScript Application Programming Interfaces

<table>
<thead>
<tr>
<th>API</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database</td>
<td>Enables access to the SAP HANA by means of SQL statements. For example, you can open a connection to commit or rollback changes in SAP HANA, to prepare stored procedures (or SQL statements) for execution or to return details of a result set or a result set’s metadata.</td>
</tr>
<tr>
<td>Outbound connectivity</td>
<td>Enables outbound access to a defined HTTP destination that provides services which an application can use. For example, you can read the connection details for an HTTP destination, request data, and set details of the response body. You can also set up an SMTP connection for use by outgoing multipart e-mails.</td>
</tr>
<tr>
<td>Request processing</td>
<td>Enables access to the context of the current HTTP request, for example, for read requests and write responses. You can use the functions provided by this API to manipulate the content of the request and the response.</td>
</tr>
<tr>
<td>Session</td>
<td>Enables access to the SAP HANA XS session, for example, to determine the language used in the session or if a user has the privileges required to run an application.</td>
</tr>
<tr>
<td>Job Schedule</td>
<td>Enables access to the job-scheduling interface which allows you to define and trigger recurring tasks that run in the background. The XS jobs API allows you to add and remove schedules from jobs.</td>
</tr>
<tr>
<td>Security</td>
<td>Enables access to the $.security.crypto namespace and the classes AntiVirus and Store, which provide tools that allow you to configure a secure store, set up anti-virus scans, and generate hashes.</td>
</tr>
<tr>
<td>Trace</td>
<td>Enables access to the various trace levels you can use to generate and log information about application activity. You can view trace files in the diagnosis Files tab of the SAP HANA studio’s Administration perspective.</td>
</tr>
<tr>
<td>Utilities</td>
<td>Enables access to utilities that you can use to parse XML and manipulate Zip archives, for example, to zip and unzip files, add and remove entries from Zip archives, and encrypt Zip archives with password protection.</td>
</tr>
<tr>
<td>XS Data Services</td>
<td>Provides access to a library of JavaScript utilities, which can be used to enable server-side JavaScript applications to consume data models that are defined using Core Data Services.</td>
</tr>
<tr>
<td>XS Procedures</td>
<td>Provides access to a library of JavaScript utilities, which can be used to enable server-side JavaScript applications to call SAP HANA stored procedures as if the procedures were JavaScript functions.</td>
</tr>
</tbody>
</table>

Database API

The SAP HANA XS Database API ($\.hdb) provides tools that enable simple and convenient access to the database.
Caution

The `.hdb` namespace is intended as a replacement for the older `.db` namespace. Since different database connections are used for the `.hdb` and `.db` APIs, avoid using both APIs in a single http-request, for example, to update the same tables as this can lead to problems, including deadlocks.

You can use the Database API for the following operations

- `.hdb.Connection`
  - Establish a connection to the SAP HANA database
- `.hdb.ProcedureResult`
  - Represents the result of a stored procedure call to the SAP HANA database
- `.hdb.ResultSet`
  - Represents the result of a database query

The following example shows how to use the database API to connect to the SAP HANA database, commit some changes, and end the current transaction.

**Note**

By default, auto-commit mode is disabled, which means that all database changes must be explicitly committed.

```javascript
var connection = $.hdb.getConnection();
connection.executeUpdate('UPDATE "DB_EXAMPLE"."ICECREAM" SET QUANTITY=? WHERE FLAVOR=?', 9, 'CHOCOLATE');
connection.commit();
```

The following example of usage of the SAP HANA XS database API shows how to establish a connection with SAP HANA and return a result set from the specified procedure call. The example code assumes that a procedure exists with the following signature:

```javascript
PROCEDURE 'DB_EXAMPLE'.icecream.shop::sell(
    IN flavor VARCHAR,
    IN quantity INTEGER,
    IN payment DECIMAL,
    OUT change DECIMAL)
```

Note that the result can be accessed as if it were a JSON object with a structure similar to the following example: `{change: 1.50, $resultSets:[...])`.

**Tip**

$.resultSets is not enumerable; it does not show up in a for-each loop.

```javascript
var fnSell = connection.loadProcedure('DB_EXAMPLE', 'icecream.shop::sell');
var result = fnSell('CHOCOLATE', 3, 30.0);
// value of output parameter 'change'
var change = result['change'];
// array of $.hdb.ResultSet returned by the stored procedure
var resultSets = result['$resultSets'];
// iterate over all output parameters.
var params;
for (var outputParam in result) {
    params += outputParam + ' ';
    }
Outbound API

The Outbound API ($\$.net$) provides tools that you can use to perform the following actions:

- $\$.net.SMTPConnection$
  
  For sending $\$.net.Mail$ objects by means of an SMTP connection

- $\$.net.Mail$
  
  For constructing and sending multipart e-mails

- $\$.net.http$
  
  HTTP(s) client (and request) classes for outbound connectivity and an HTTP(s) destination class that hold metadata, for example: host, port, useSSL.

The following example shows how to use the $\$.net.SMTPConnection$ class to send e-mail objects ($\$.net.Mail$) by means of an SMTP connection object:

```javascript
subscribers = ["kofi@sap.com", "kwaku@sap.com"]; 
smtpConnection = new SMTPConnection(); 
var mail = new $.net.Mail({ sender: "manager@sap.com", 
  subject: "Promotion Notice", 
  subjectEncoding: "UTF-8", 
  parts: [new $.net.Mail.Part({
    type: $.net.Mail.Part.TYPE_TEXT, 
    contentType: "text/html", 
    encoding: "UTF-8"
  })]
}); 
for (var i = 0; i < subscribers.length; ++i) { 
  mail.to = subscribers[i]; 
  mail.parts[0].text = "Dear " + subscribers[i].split("@")[0] + ",
  you have been promoted. Congratulations!";
  smtpConnection.send(mail);
}
smtpConnection.close();
```

The following example shows how to use the $\$.net.Mail$ class to create an e-mail from an XS JavaScript object and send it to the named recipients:

### Note

If mandatory information is missing or an error occurs during the send operation, the `mail.send()` call fails and returns an error.

```javascript
var mail = new $.net.Mail({
  sender: {address: "sender@sap.com"},
  to: [{ name: "John Doe", address: "john.doe@sap.com", nameEncoding: "US-ASCII"}],
  cc: [{name: "Jane Doe", address: "jane.doe@sap.com"}],
  bcc: [{ name: "Jonnie Doe", address: "jonnie.doe@sap.com"}],
  subject: "subject",
  subjectEncoding: "UTF-8",
  parts: [ new $.net.Mail.Part({
    type: $.net.Mail.Part.TYPE_TEXT, 
    text: "The body of the mail.",
    contentType: "text/plain",
  })]
});
```
The following example of server-side JavaScript shows how to use the outbound API to get (read) an HTTP destination. You can also set the contents of the response, for example, to include details of the header, body, and any cookies. For HTTPS connections you need to maintain a certificate (CA or explicit server certificate) in a Trust Store; you use the certificate to check the connection against.

```javascript
var dest = $.net.http.readDestination("inject", "ipsec");
var client = new $.net.http.Client();
var req = new $.web.WebRequest($.net.http.GET, "");
client.setRequest(req, dest);
var response = client.getResponse();
var co = [], he = [];
for (var c in response.cookies) {
    co.push(response.cookies[c]);
}
for (var c in response.headers) {
    he.push(response.headers[c]);
}
var body = undefined;
if (response.body)
    var body = response.body.asString();
$.response.contentType = "text/html";
```

**Tip**

You define the HTTP destination in a text file using keyword=value pairs. You must activate the HTTP destination in the SAP HANA repository. After activation, you can view details of the HTTP destination in the SAP HANA XS Administration tool.

### Request-Processing API

The Request-Processing API ($\texttt{$.web}$) provides access to the body of HTTP request and response entities. For example, you can use the following classes:

- **$.web.Body**
  
  Represents the body of an HTTP request entity and provides access to the data included in the body of the HTTP request entity

- **$.web.EntityList**
  
  Represents a list of request or response entities; the EntityList holds WebEntityRequest or WebEntityResponse objects.

- **$.web.TupellList**
  
  Represents a list of name-value pairs. The TupellList is a container that provides tuples for cookies, headers, and parameters. A “tuple” is a JavaScript object with the properties “name” and “value”.

- **$.web.WebRequest**
  
  Enables access to the client HTTP request currently being processed.
$.web.WebResponse
Enables access to the client HTTP response currently being processed for the corresponding request object.

$.web.WebEntityRequest
Represents an HTTP request entity and provides access to the entity's metadata and (body) content.

$.web.WebEntityResponse
Represents the HTTP response currently being populated

The following example shows how to use the request-processing API to display the message “Hello World” in a browser.

```javascript
$.response.contentType = "text/plain";
$.response.setBody("Hello, World!");
```

In the following example, you can see how to use the request-processing API to get the value of parameters describing the name and vendor ID of a delivery unit (DU) and return the result set in JSON-compliant form.

```javascript
var duName = $.request.parameters.get("du_name");
var duVendor = $.request.parameters.get("du_vendor");
result = {
    content_id : contentId.toString()
};
$.response.status = $.net.http.OK;
$.response.contentType = 'application/json';
$.response.setBody(JSON.stringify(result));
```

In the following example of use of the request-processing API, we show how to access the request's metadata (and body) and, in addition, how to set and send the response.

```javascript
if ($.request.method === $.net.http.GET) {
    // get query parameter named id
    var qpId = $.request.parameters.get("id");
    // handle request for the given id parameter...
    var result = handleRequest(qpId);
    // send response
    $.response.contentType = "plain/test";
    $.response.setBody("result: " + result);
    $.response.status = $.net.http.OK;
} else {
    // unsupported method
    $.response.status = $.net.http.INTERNAL_SERVER_ERROR;
}
```

**Session API**

Enables access to the SAP HANA XS session, for example, to determine the language used in the session or check if a user has the privileges required to run an application.

You can use the XS JavaScript $.session API to request and check information about the currently open sessions. For example, you can find out the name of a user who is currently logged on to the database or get the session-specific security token. The $.session API also enables you to check if a user has sufficient privileges to call an application. The following example checks if the user has the `execute` privilege that is
required to run an application. If the check reveals that the user does not have the required privilege, an error message is generated indicating the name of the missing privilege.

```javascript
if (!$.session.hasAppPrivilege("sap.xse.test::Execute")) {
  $.response.setBody("Privilege sap.xse.test::Execute is missing");
  $.response.status = $.net.http.INTERNAL_SERVER_ERROR;
}
```

## Job Schedule API

In SAP HANA XS, a scheduled job is created by means of an `.xsjob` file, a design-time file you commit to (and activate in) the SAP HANA repository. The `.xsjob` file can be used to define recurring tasks that run in the background; the Job Schedule API allows developers to add and remove schedules from such jobs.

The Job Schedule API provides the following tools:

- **Job**
  - `$.jobs.Job` represents a scheduled XS job
- **JobLog**
  - `$.jobs.JobLog` provide access to the log entries of a scheduled job
- **JobSchedules**
  - `$.jobs.JobSchedules` enables control of an XS job’s schedules.

### Note

It is not possible to call the `$request` and `$response` objects as part of an XS job.

The XS jobs API `$.jobs.Job` enables you to add schedules to (and remove schedules from) jobs defined in an `.xsjob` file.

The following example of server-side JavaScript shows how to use the Job Schedule API to add a schedule to a existing job and delete a schedule from an existing job.

```javascript
var myjob = new $.jobs.Job({uri:"myJob.xsjob", sqlcc:"sqlcc/otheruser.xssqlcc"});
// add schedule to a job
var id = myjob.schedules.add(
    description: "Added at runtime, run every 10 minutes",
    xscron: "* * * * */10 0",
    parameter: {
        a: "c"
    }
);
// delete a schedule from a job
myjob.schedules.delete({id: id});
```

If the XS job file referred to in the URI is not in the same package as the XS JavaScript or SQLScript function being called, you must add the full package path to the XS job file specified in the URI illustrated in line 1 of the example above, for example, `<path/to/package.>MyXSjob.xsjob`.

### Note

The path specified in `<path/to/package.>` can be either absolute or relative.
In addition, the SQL connection defined in sqlcc/otheruser.xssqlcc is used to modify the job; it is not used to execute the job specified in myJob.xsjob.

To understand the cron-like syntax required by the xscron job scheduler, use the following examples:

- **2013 * * fri 12 0 0**
  Run the job every Friday in 2013 at 12:00.

- *** * 3:-2 * 12:14 0 0**
  Run every hour between 12:00 and 14:00 every day between the third and second-to-last day of the month.

- *** * * -1.sun 9 0 0**
  Run the job on the last Sunday of every month at 09:00.

**Security API**

The SAP HANA XS JavaScript security API $.security includes the $.security.crypto namespace and the following classes:

- **$.security.AntiVirus**
  Scan data with a supported external anti-virus engine

- **$.security.Store**
  Store data securely in name-value form

The $.security.crypto namespace includes methods (for example, md5(), sha1(), and sha256()) that enable you to compute an MD5 or SHA1/256 hash (or HMAC-MD5, HMAC-SHA1, and HMAC-SHA256).

The AntiVirus class includes a method scan() that enables you to set up a scan instance using one of the supported anti-virus engines. The Store class enables you to set up a secure store for an SAP HANA XS application; the secure store can be used to store sensitive information either at the application level (store()) or per user (storeForUser()).

The following code example shows how to use the SAP HANA XS virus-scan interface (VSI) to scan a specific object type: a Microsoft Word document.

```javascript
var data = //Some data to be checked
var av = new $.security.AntiVirus();
//AV scan data as Word document
av.scan(data, "myDocument.docx");
```

The following code example shows how to set up a simple scan for data uploads using the SAP HANA XS virus-scan interface.

```javascript
//scan a buffer with own "upload" profile
var av = new $.security.AntiVirus("upload");
av.scan(buffer);
```

**Note**

For more information about which antivirus engines SAP HANA supports, see SAP Note 786179.
The SAP HANA XS $.security.Store API can be used to store data safely and securely in name-value form. The security API enables you to define a secure store (in a design-time artifact) for each application and refer to this design time object in the application coding.

Note

The design-time secure store is a file with the file extension ".xssecurestore", for example, localStore.xssecurestore; the secure-store file must include only the following mandatory content: `{}.

SAP HANA XS looks after the encryption and decryption of data and also ensures the persistency of the data. For the stored data, you can choose between the following visibility options:

- Application-wide data visibility
  Use `store(parameters)` to ensure that all users of the corresponding application have access to one secure store where they can share the same data and can decrypt or encrypt data, for example, passwords for a remote system.

- Application-wide data visibility but with user-specific stores separation
  Use `storeForUser(parameters)` to ensure that each user of the corresponding application has a separate container to securely store personal, encrypted data, for example, credit card numbers or personal-information-number (PIN) codes; the encrypted data can only be decrypted by the owner of the secure store, the user who encrypted it.

```javascript
function store() {
  var config = {
    name: "foo",
    value: "bar"
  };
  var aStore = new $.security.Store("localStore.xssecurestore");
  aStore.store(config);
}

function read() {
  var config = {
    name: "foo"
  };
  try {
    var store = new $.security.Store("localStore.xssecurestore");
    var value = store.read(config);
  } catch(ex) {
    //do some error handling
  }
}
```

Trace API

Enables access to the various trace levels you can use to generate and log information about application activity. The specified error message is written to the appropriate trace file.

```
$.trace.error("This is an error message")
```

You can set the following trace levels:

- `.trace.debug(message)`
writes the string defined in (message) to the application trace with **debug** level

- $.trace.error(message)
  writes the string defined in (message) to the application trace with **error** level

- $.trace.fatal(message)
  writes the string defined in (message) to the application trace with **fatal** level

- $.trace.info(message)
  writes the string defined in (message) to the application trace with **info** level

- $.trace.warning(message)
  writes the string defined in (message) to the application trace with **warning** level

### Note

If tracing is enabled, messages generated by the $.trace API are logged in the SAP HANA trace file `xsengine_<host>_<instance>_<#.trc` on the SAP HANA server, for example, in `<installation_path>/<SID>/HDB/<hostname>/trace`. Trace messages with severity status “warning”, “error” and “fatal” are also written to a similarly named alert file, for example, `xsengine_alert_<host>.trc`.

### Utilities API

The SAP HANA XS JavaScript Utilities API includes the $.util namespace, which contains the following classes:

- $.util.SAXParser
  Tools for parsing XML content (for example, strings, array buffers, and the content of Web response body objects)

- $.util.Zip
  Compression tools for building, modifying, extracting, and encrypting archives

With the XS JavaScript Utilities APIs $.util.SAXParser class, you can create a new parser object and parse the XML content of an XMLString, an XML array buffer, or a $.web.Body object. The following example shows how to use the XML parsing capabilities of the $.util.SAXParser class:

### Note

You can **stop**, **reset**, and **resume** a parsing operation. If the content to be parsed does not contain XML, the parser throws an error.

```javascript
var parser = new $.util.SAXParser();
var xml = "<?xml version="1.0" encoding="UTF-8" standalone="yes"?>\n" +
"<!-- this is a note --\n" +
"<note noteName='NoteName'>\n" +
"<to>To</to>\n" +
"<from>From</from>\n" +
"<heading>Note heading</heading>\n" +
"<body>Note body</body>\n" +
"<note>\n";
var startElementHandlerConcat = "";
var endElementHandlerConcat = "";
var characterDataHandlerConcat = "";
```
The following code snippet shows how to use the $.util.SAXParser tools to parse the content of a $.web.Body object.

```javascript
var body = $.request.body
var parser = new $.util.SAXParser()
//... set handlers
parser.parse(body);
```

The following encodings are supported:

- UTF-8 (**default**)
- UTF-16
- US-ASCII

The SAP HANA XS JavaScript Utilities API also includes the $.util.Zip tool, which enables you to perform a series of actions on Zip archives, for example:

- Compress files into (zip) and extract files from (unzip) a Zip archive
- Add new entries to, update existing entries in, and remove entries from a Zip archive
- Encrypt Zip archives with password protection

The following code illustrates a simple usage of the Zip tool:

```javascript
var zip = new $.util.Zip("myPassword");
zip["entry.txt"] = "Two fish are in a tank. One turns to the other and asks 'How do you drive this thing?'";
$.response.status = $.net.http.OK;
$.response.contentType = "application/zip";
$.response.headers.set("Content-Disposition", "attachment; filename = Encrypted.zip");
$.response.setBody(zip.asArrayBuffer());
```

The following code snippets show how to use the $.util.Zip tools to work with Zip file content, for example, by adding, updating, extracting, and deleting entries. When modeling folder hierarchies, the Zip object behaves like an associative array; the entry names are the keys (the full paths to the indicated files). In the following example, we add an entry to a Zip file:

```javascript
var zip = new $.util.Zip();
zip["entry1"] = "old entry";
```

---

**Note**

"zip["entry1"]" is equivalent to "zip.entry1".
In the following example, we **update** an entry in a Zip file:

```javascript
var zip = new $.util.Zip();
zip["entry1"] = "new entry";
```

In the following example, we **extract** an entry from a Zip file: if the entry does not exist, this returns undefined.

```javascript
var zip = new $.util.Zip();
var content = zip["entry1"];```

In the following example, we **delete** an entry from a Zip file: if the entry does not exist, nothing happens.

```javascript
var zip = new $.util.Zip();
delete zip["entry1"];```

---

**Note**

There is a restriction on the amount of uncompressed data that can be extracted from a Zip archive using the XS JS utilities API.

When using the XS JS utilities API to extract data from a Zip archive, the maximum amount of uncompressed data allowed during the extraction process is defined with the parameter `max_uncompressed_size_in_bytes`, which you can set in the `zip` section of the `xsengine.ini` configuration file for a given SAP HANA system. If the `zip` section does not already exist, you must create it and add the parameter to it, for example, using the **SAP HANA Administration Console** in SAP HANA studio. If the parameter `max_uncompressed_size_in_bytes` is **not** set, a default value is assumed. The default value is the value assigned to the property `max_runtime_bytes` in section `jsvm` section of the `xsengine.ini` file.

You can deactivate the global check on the amount of uncompressed data. If the global system parameter `max_uncompressed_size_in_bytes` is set to `-1`, no check is performed on the amount of uncompressed data generated by an extraction process using the Utilities API, unless there is a specific user limitation in the XS JavaScript code, for example, with the `maxUncompressedSizeInBytes` parameter.

With the `$.util.Zip` class or the `$.util.compression` namespace, you can use the property `maxUncompressedSizeInBytes` to override the global setting and reduce the amount of uncompressed data allowed.

---

**Note**

Note that the parameter `max_uncompressed_size_in_bytes` cannot be used to increase the amount of uncompressed data allowed beyond the value specified in the global setting.

---

**XS Data Services API**

SAP HANA XS Data Services (XSDS) is a collection of tools that includes a native client for Core Data Services (CDS) and a query builder for SAP HANA Extended Application Services (SAP HANA XS) JavaScript. The XSDS API provides a high-level abstraction of the database API (`$.db`, `$.hdb`) and gives access to SAP HANA artifacts such as CDS entities or stored procedures. XSDS enables server-side JavaScript applications to consume data models that are defined using Core Data Services more efficiently.
The following example shows how to import a CDS entity and how to update a given entity instance in XSDS managed mode.

```javascript
// import CDS client library
var XSDS = $.import("sap.hana.xs.libs.dbutils", "xsds");
// import CDS entity
var MyEntity = XSDS.$importEntity("cds.namespace", "cds_context.cds_entity");
// retrieve entity instance
var instance = MyEntity.$get({ id: 69 });
// update instance
instance.stringProp = "new value";
instance.intProp++;
instance.assocProp.dateProp = new Date();
// persist changes
instance.$save();
```

The following example shows how to query the database using CDS model data in XSDS unmanaged mode.

```javascript
// import CDS client library
var XSDS = $.import("sap.hana.xs.libs.dbutils", "xsds");
// import CDS entity
var MyEntity = XSDS.$importEntity("cds.namespace", "cds_context.cds_entity");
// build query
var query = MyEntity.$query();
var projection = query.$project({
  stringProp: true,
  aliasProp: "aliasName",
  assocProp: { dateProp: true }
});
var filter = query.$where({ stringProp: { $like: "A%" } });
// retrieve result
var result = projection.$execute();
// process result
for (var i = 0; i < result.length; i++) {
  var diff = result[i].assocProp.dateProp - Date.now();
  // ...
}
```

**XS Procedures API**

SAP HANA XS Procedures is a library of JavaScript tools which enable you to call SAP HANA stored procedures from server-side JavaScript (XS JS) as if the stored procedures were native JavaScript functions. The following example shows how to consume a stored procedure using the XS Procedures API.

```javascript
// import XS Procedures library
var XSProc = $.import("sap.hana.xs.libs.dbutils", "procedures");
// set a schema where temporary tables can be created for passing table-valued parameters to the procedure
XSProc.setTempSchema($.session.getUsername().toUpperCase());
// load the procedure
var proc = XSProc.procedure("schema", "namespace", "procedureName");
// call the procedure
var result = proc(1, [{col1: 0, col2:1}, {col1: 1, col2:2}]);
// result is a JavaScript object
```
2.4.1 Tutorial: Use the XSJS Outbound API

The application package you put together in this tutorial includes all the artifacts you need to enable your server-side JavaScript application to use the Outbound Connectivity API to request and obtain data via HTTP from a service running on a remote host.

Prerequisites

Since the artifacts required to get the JavaScript application up and running are stored in the repository, it is assumed that you have already performed the following tasks:

- Create a development workspace in the SAP HANA repository
- Create a project in the workspace
- Share the new project
- The HTTPDestViewer SAP HANA user role

Context

SAP HANA Extended Application Services (SAP HANA XS) includes a server-side JavaScript API that enables outbound access to a defined HTTP destination. The HTTP destination provides services which an application can use, for example, to read live data. In this tutorial, you create a JavaScript application that queries financial services to display the latest stock values. The financial services are available on a remote server, whose details are specified in an HTTP destination configuration.

Procedure

1. Create a package for the SAP HANA XS application that will use the HTTP destination you define in this tutorial.
   For example, create a package called testApp. Make sure you can write to the schema where you create the new application.
   a. Start the SAP HANA studio and open the SAP HANA Development perspective.
   b. In the SAP HANA Systems view, right-click the node in the package hierarchy where you want to create the new package and, in the pop-up menu that displays, choose Packages...
c. In the New Package dialog that displays, enter the details of the new package (testApp) that you want to add and click OK.

2. Define the details of the HTTP destination.
You define the details of an HTTP destination in a configuration file that requires a specific syntax. The configuration file containing the details of the HTTP destination must have the file extension .xshttpdest.

⚠️ Caution
Place the HTTP destination configuration in the same package as the application that uses it. An application cannot reference an HTTP destination configuration that is located in another application package.

a. Create a plain-text file called yahoo.xshttpdest and open it in a text editor.
   You can use the file-creation wizard in the Project Explorer view to create this file, for example, New Other XS HTTP Destination Configuration.

b. Enter the following code in the new file yahoo.xshttpdest.

```javascript
host = "download.finance.yahoo.com";
port = 80;
description = "my stock-price checker";
useSSL = false;
pathPrefix = "/d/quotes.csv?f=a";
authType = none;
useProxy = false;
proxyHost = "";
proxyPort = 0;
timeout = 0;
```

c. Save and activate the file.

ℹ️ Note
Saving a file in a shared project automatically commits the saved version of the file to the repository.

3. View the activated HTTP destination.
You can use the SAP HANA XS Administration Tool to check the contents of an HTTP destination configuration.

ℹ️ Note
To make changes to the HTTP Destination configuration, you must use a text editor, save the changes and reactivate the file.

a. Open a Web browser.
b. Start the SAP HANA XS Administration Tool.
   The SAP HANA XS Administration Tool tool is available on the SAP HANA XS Web server at the following URL: http://<WebServerHost>:80<SAPHANAinstance>/sap/bAna/xs/admin/.

ℹ️ Note
Access to details of HTTP destinations in the SAP HANA XS Administration Tool requires the credentials of an authenticated database user and one of the following SAP HANA roles:
c. In the **XS Artifact Administration** screen, expand the nodes in the **Application Objects** tree to locate the application **testApp**.

d. Choose **yahoo.xshttpdest** to display details of the **HTTP destination**.

e. Check the details displayed and modify if required.

4. Create a server-side JavaScript application that uses the **HTTP destination** you have defined.

The **XSJS file** must have the file extension **.xsjs**, for example, **sapStock.xsjs**.

**Caution**

You must place the XSJS application and the **HTTP destination configuration** it references in the same application package. An application cannot use an **HTTP destination configuration** that is located in another application package.

a. Create a plain-text file called **sapStock.xsjs** and open it in a text editor.

b. Enter the following code in the new file **sapStock.xsjs**.

   In this example, you define the following:
   
   - A variable (**stock**) that defines the name of the stock, whose value you want to check, for example **SAP.DE**
   - A variable (**amount**) that defines the number of stocks you want to check, for example, 100
   - A variable (**dest**) that retrieves metadata defined for the specified **HTTP(S) destination**, for example: host, port, useSSL...
   - A variable (**client**) that creates the client for the outbound connection
   - A variable (**req**) that enables you to add details to the request URL
   - A variable (**res**) that calculates the value of the stock/amount
   - The format and content of the response body displayed in the browser

   ```js
   var stock = $.request.parameters.get("stock");
   var amount = $.request.parameters.get("amount");
   var dest = $.net.http.readDestination("testApp", "yahoo");
   var client = new $.net.http.Client();
   var req = new $.web.WebRequest($.net.http.GET, "/s=" + stock);
   client.request(req, dest);
   var response = client.getResponse();
   var co = [], he = []; 
   for(var c in response.cookies) {
       co.push(response.cookies[c]);
   }
   for(var c in response.headers) {
       he.push(response.headers[c]);
   }
   var body = undefined;
   if(response.body)
       var body = response.body.asString();
   $.response.contentType = "application/json";
   var res = parseInt(response.body.asString()) * amount;
   $.response.setBody(amount + " of your " + stock + " are worth: " + res);
   ```

c. Save and activate the file.

5. Call the service provided by the application **sapStock.xsjs**.

   a. Open a Web browser.
b. Enter the URL that calls your sapStock.xsjs application.

    http://<XS_Webserver>:80<SAPHANA_InstanceNr>/testApp/sapStock.xsjs?
    amount=100&stock=SAP.DE

- `<XS_Webserver>`
  Name of the system hosting the Web server for the SAP HANA XS instance where your sapStock.xsjs application is located.

- `<SAPHANA_InstanceNr>`
  Number of the SAP HANA instance where the SAP HANA XS Web server is running, for example, 00

6. Change the details specified in the URL used to run the application.

You can enter different values for the parameters `&amount` and `&stock` in the URL:

- `amount=250`
  Change the number of stocks to check from 100 to 250

- `&stock=SAP.DE`
  Change the name of stock to check from `SAP.DE` to `MCRO.L`

Related Information

SAP HANA XS JavaScript API Reference
### 2.4.2 Tutorial: Call an XS Procedure with Table-Value Arguments

You can use the XS Procedures library to call stored procedures as if they were JavaScript functions.

#### Prerequisites

- The delivery unit `HANA_XS_DBUTILS` contains the XS procedures library. The content is available in the package `sap.hana.xs.libs.dbutils`.
- Create a new (or use an existing) development workspace in the SAP HANA repository.
- Create a new (or use an existing) shared project in the workspace.
- Create a new (or use an existing) stored procedure.

This tutorial refers to the stored procedure `get_product_sales_price`, which is included in the demonstration content provided with the SAP HANA Interactive Education (SHINE) delivery unit (DU). The SHINE DU is available for download in the SAP Software Download Center.

#### Context

You can call stored procedures by using the contents of the XS Procedures library as if they were JavaScript functions. For example, the library allows you to pass arguments as a JavaScript object to a stored procedure that expects table arguments; XS Procedures manages the creation and use of the temporary tables needed to pass arguments to a table-valued procedure. You can use the functions provided with the XS procedures library to enable programmatic access to stored procedures in the SAP HANA database from an XS JavaScript service; the access is provided by binding the stored procedure to a JavaScript function. The result of the call to the bound function is a JavaScript object, whose properties are the outbound parameters of the procedure.

#### Procedure

1. **Import the XS procedures library.**
   
   In your server-side (XS) JavaScript code, ensure that the XS procedures are made available.
   
   ```javascript
   var XSProc = $.import("sap.hana.xs.libs.dbutils", "procedures");
   ```

2. **Specify a schema where temporary tables can be created and filled with the values that are passed as arguments to the stored procedure.**
XS procedures use temporary tables to pass table-valued parameters. As a user of XS procedures you must specify the name of a schema where these temporary tables reside, for example, a user's own schema.

**Note**
The application code using XS procedures must ensure that the necessary privileges have been granted to enable the creation and update of (and selection from) temporary tables in the specified schema.

```javascript
XSProc.setTempSchema($.session.getUsername().toUpperCase());
```

3. Bind the stored procedure to a JavaScript function.

This step creates one or more JavaScript functions which can later be used to call the stored procedure. You can also define functions which map your call arguments to the parameters of the stored procedure.

```javascript
var createPurchaseOrder = XSProc.procedure("SAP_HANA_DEMO", "sap.hana.democontent.epm.Procedures", "poCreate", {connection: conn});
```

**Note**
XS procedures uses the connection `configuration {connection: conn}` passed in a configuration object as a parameter. If no connection object is passed, the XS procedure library opens a separate connection for the call and closes the connection after the call completes.

4. Call the procedure.

Use the imported procedure like a normal JavaScript function using JavaScript object argument lists.

```javascript
var result = createPurchaseOrder({
  "PURCHASEORDERID": '0300009001',
  "HISTORY.CREATEDBY": '0000000044',
  "HISTORY.CREATEDAT": new Date(),
  "HISTORY.CHANGEDBY": '0000000044',
  "HISTORY.CHANGEDAT": new Date()
});
```

Table-valued input arguments are passed to the stored procedure using a Javascript array that corresponds to the rows of the table containing the values to pass. The row objects should contain the properties of the name of the columns. Skipped columns are filled with NULL; properties without a same-named column are ignored.

**Example**

```javascript
var XSProc = $.import("sap.hana.xs.libs.dbutils", "procedures");
XSProc.setTempSchema($.session.getUsername().toUpperCase());
var conn = $.db.getConnection();
var createPurchaseOrder = XSProc.procedure("SAP_HANA_DEMO", "sap.hana.democontent.epm.Procedures", "poCreate", {connection: conn});
var result = createPurchaseOrder({
  "PURCHASEORDERID": '0300009001',
  "HISTORY.CREATEDBY": '0000000044',
  "HISTORY.CREATEDAT": new Date(),
  "HISTORY.CHANGEDBY": '0000000044',
  "HISTORY.CHANGEDAT": new Date()
});
if (result && result.ERROR.length > 0) {
```
Related Information


2.4.2.1 Accessing Stored Procedures from XS JavaScript

Call stored SAP HANA procedures from XS server-side JavaScript (XSJS) and process the results of the calls in JavaScript.

XS procedures provide a convenient way to call stored procedures in SAP HANA from XS server-side JavaScript (XSJS) and process the results of the calls in JavaScript. The XS procedures library extends the features already available with the SAP HANA XS JavaScript database API. Using XS procedures, SAP HANA stored procedures can be considered as simple XS JavaScript functions for anyone developing XS JavaScript services.

For example, where an SAP HANA stored procedure uses a table as input parameter and a table as output, XS Procedures use JavaScript objects (or an array of objects) which can be passed to the procedure. Similarly, the result of the procedure call is provided as an array of JavaScript objects. You declare a stored procedure as an XS JavaScript function and then call the stored procedure as if it were a JavaScript function delivering a JavaScript object.

To use a stored procedure as an XS JavaScript function, the following steps are required:

Table 9:

<table>
<thead>
<tr>
<th>Step</th>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Import the XS Procedures library</td>
<td>Provide access to the XS procedures</td>
</tr>
<tr>
<td>2</td>
<td>Specify a schema for temporary tables</td>
<td>Temporary tables are used to store the JavaScript arguments provided for the function.</td>
</tr>
<tr>
<td>3</td>
<td>Import the procedure</td>
<td>Create the XS JavaScript functions, which can later be used to call the stored SAP HANA procedure. You can define functions which map your call arguments to the parameters of the stored procedure.</td>
</tr>
<tr>
<td>4</td>
<td>Call the procedure</td>
<td>Use the imported procedure in the same way as any normal JavaScript function, for example, using JavaScript object argument lists.</td>
</tr>
<tr>
<td></td>
<td>Use Arguments that Reference an Existing Table [page 63]</td>
<td>(Optional) Write the results or a procedure call into a physical table and pass the table as an argument rather than a JavaScript object</td>
</tr>
</tbody>
</table>
Calling Procedures with Arguments that Reference an Existing Table

If you want to pass a table as an argument rather than a JavaScript object, you must specify the name of the table (as a string) in the call statement as well as the name of the schema where the table is located. The following example shows how to reference the table `rating_table`.

```javascript
getRating('schema.rating_table', 3);
```

The SAP HANA database enables you to materialize the results of a procedure call; that is, to write the results into a physical table using the `WITH OVERVIEW` expression. In the `WITH OVERVIEW` expression, you pass a string value to the output parameter position that contains the result you want to materialize. The value returned is not the rating itself, but a reference to the table into which the results have been written. The results of the procedure call can now be retrieved from the specified table, in this example, `OUTPUT_TABLE`.

```javascript
var resCall = getRating(rating, 3, "schema.output_table");
// {"RESULT": [{"variable":"RESULT","table":"SCHEMA\"."OUTPUT_TABLE\""}]}
```

The `WITH OVERVIEW` expression also allows you to write the results of a procedure into a global temporary table; that is, a table that is truncated at session close. To use XS Procedures to write the results of a procedure into a global temporary table, you do not specify a name for the result table; you include an empty string (""), as illustrated in the following example:

```javascript
var conn = $.db.getConnection();
resCall = getRating(rating, 3, "", conn);
// {"RESULT": [{"variable":"RESULT","table":"SCHEMA\".\"RESULT_5270ECB8F7061B7EE10000000A379516\""}]}
```

The returned reference points to a global temporary table which can be queried for the procedure results with the same connection.

**Note**

To ensure access to the global temporary table, it is necessary to specify the connection object `conn`.

Using Table-Valued Arguments

XS Procedures enables you to call procedures with arguments stored as values in a table, as illustrated in the following example. Table-valued input arguments are passed using a JavaScript array that corresponds to the rows of the table to pass. These row objects must contain properties that correspond to the name of the
columns. Skipped columns are filled with NULL, and properties that do not correspond to an identically named

column are ignored.

```javascript
var XSProc = $.import("sap.hana.xs.libs.dbutils", "procedures");
XSProc.setTempSchema($.session.getUsername().toUpperCase());
var conn = $.db.getConnection();
var createPurchaseOrder = XSProc.procedure("SAP_HANA_DEMO",
  "sap.hana.democontent.epm.Procedures::poCreate", {
    connection: conn
  });
var result = createPurchaseOrder({
  "PURCHASEORDERID": '0300009001',
  "HISTORY.CREATEDBY": '0000000044',
  "HISTORY.CREATEDAT": new Date(),
  "HISTORY.CHANGEDBY": '0000000044',
  "HISTORY.CHANGEDAT": new Date()
});
if (result && result.ERROR.length > 0) {
  $.response.setBody(result.ERROR.length + " errors occurred.");
} else {
  $.response.setBody("no error occurred");
}
```

Related Information

SAP HANA XS JavaScript API Reference

2.4.3 Tutorial: Query a CDS Entity using XS Data Services

You can use the SAP HANA XS Data Services (XDS) library to query CDS entities as if they were JavaScript objects.

Prerequisites

- A new (or an existing) development workspace in the SAP HANA repository
- A new (or an existing) shared project in the workspace
- This tutorial refers to CDS models that are included in the demonstration content provided with the SAP HANA Interactive Education (SHINE) delivery unit (DU). The SHINE DU is available for download in the SAP Software Download Center.

**Note**

Access to the SAP Software Download Center is only available to SAP customers and requires logon credentials.
Context

XS Data Service queries are used to build incrementally advanced queries against data models that are defined with Core Data Service. Query results are arrays of nested JSON objects that correspond to instances of CDS entities and their associations.

Procedure

1. Import the XS DS library and reference it through a variable.

   ```javascript
   var XSDS = $.import("sap.hana.xs.libs.dbutils", "xsds");
   ```

2. Import the CDS entities you want to query.

   As a first step to working with CDS entities in SAP HANA XS JavaScript, you must import the CDS entities. The following example shows how to import the entities as defined in the SHINE demonstration content:

   ```javascript
   var soItem = XSDS.$importEntity("sap.hana.democontent.epm.data", "EPM.SO.Item");
   var soHeader = XSDS.$importEntity("sap.hana.democontent.epm.data", "EPM.SO.Header", {
      items: {
        $association: {
          $entity: soItem,
          $viaBacklink: "SALESORDERID"
        }
      }
   });
   ```

   In addition to the basic CDS definition, the code in the example above shows how to extend the definition of soHeader by an explicit association called items. This is done by using the keyword $association together with the referenced entity (soItem) and the type of the association. In this case, $viaBacklink is used as type, that is; the items of soHeader stored in soItem have a foreign key SALESORDERID referencing the key of the soHeader table.

3. Add a query.

   A general query related to an entity is built by calling the $query() method of the entity constructor.

   ```javascript
   var qOrders = soHeader.$query();
   ```

4. Refine the query if required.

   You can refine the query object as necessary to suit your use case. For example, you can specify that the query returns only the first three (3) entries.

   ```javascript
   qOrders = qOrders.$limit(3);
   ```

5. Execute the query.

   Use the $execute method to run the query.

   ```javascript
   var result = qOrders.$execute();
   ```

   result contains an array of unmanaged values, each of which represents a row of the Post entity.
6. Specify the fields the query should return.

Use the $project() method to create a query which specifies the fields the query should return. For example, you can return the IDs of the sales orders together with the net amount of the header and the net amount of all items.

```javascript
var qOrderAndItemTitles = qOrders.$project({
    SALESORDERID: true,
    NETAMOUNT: "TotalNet",
    items: {
        NETAMOUNT: true
    }
});
```

The list of projected fields is a JavaScript object, where desired fields are marked by either true or a String literal such as "TotalNet" denoting an alias name. The query illustrated in the example above would return the following result.

```javascript
[{
    "SALESORDERID": "0500000236",
    "TotalNet": 273.9,
    "items": {
        "NETAMOUNT": 29.9
    }
}, {
    "SALESORDERID": "0500000236",
    "TotalNet": 273.9,
    "items": {
        "NETAMOUNT": 102
    }
}, {
    "SALESORDERID": "0500000236",
    "TotalNet": 273.9,
    "items": {
        "NETAMOUNT": 55
    }
}]
```

The actual database query automatically JOINs all required tables based on the associations involved. In the example above, the generated SQL looks like the following:

```sql
SELECT "t0"."SALESORDERID" AS "t0.SALESORDERID",
"t0"."NETAMOUNT" AS "t0.NETAMOUNT",
"t0.items"."NETAMOUNT" AS "t0.items.NETAMOUNT"
FROM "Header" "t0"
LEFT OUTER JOIN "Item" "t0.items"
ON "t0"."SALESORDERID"="t0.items"."SALESORDERID"
LIMIT 10
```

7. Use conditions to restrict the result set.
You can use the `$where()` method to set conditions that restrict the result set returned by the query. The following example shows how to select all items with a net amount equal to a half (or more) of their order’s net amount.

```javascript
var qSelectedOrders = qOrderAndItemTitles.$where(soHeader.items.NETAMOUNT.$div(soHeader.NETAMOUNT).$gt(0.5));
```

References to fields and associations such as items are available as properties of the entity constructor function, for example, `soHeader.items`. As in the case with projections, XSDS generates all required JOINs for associations referenced by the conditions automatically, even if they are not part of the current projection. To build more complex expressions in `$where`, see the SAP HANA XS Data Services JavaScript API Reference.

8. Refine the query conditions to a specific matching pattern.

With the `$matching()` method you can specify conditional expressions using the JSON-like syntax of the `$find()` and `$findAll()` methods. The following code example shows how to further refine the selection returned by the result set, for example, to accept only those items with a EUR currency and quantity greater than 2.

```javascript
qSelectedOrders = qSelectedOrders.$matching({
  items: {
    CURRENCY: 'EUR',
    QUANTITY: {
      $gt: 2
    }
  }
});
```

Tip

Unlike `$findAll()`, `$matching()` returns an unmanaged plain value and ignores all unpersistent changes to any entity instances.

9. Add arbitrary values to the result set.

You can add arbitrary calculated values to the result set by using the `$addFields()` method. The following example shows how to query the days passed since the delivery of the sales item.

```javascript
qSelectedOrders = qSelectedOrders.$addFields({
  "DaysAgo": soHeader.items.DELIVERYDATE.$prefixOp("DAYS_BETWEEN", new Date())
});
```

Note

This query refers to the SQL function `DAYS_BETWEEN`, which is not a pre-defined function in XSDS. Instead, you can use the generic operator `$prefixOp`, which can be used for any SQL function $f$, for example, with the syntax $f(arg1, ... argN)$.

10. Use aggregations with calculated fields.

Aggregations are a special case of calculated fields that combine the `$addFields()` operator with an additional `$aggregate()` method. The following example shows how to retrieve the average quantity of the first 100 sales order IDs together with their product ID.

```javascript
var qAverageQuantity = soItem.$query().$limit(100).$aggregate({
```
Tip

In SQL terms, the $aggregate() operator creates a GROUP BY expression for the specified paths and automatically projects the result.

If you need to use a more restrictive projection, you can replace `true` with `false` in the `$aggregate` call, as illustrated in the following example, which removes the sales order IDs for the result set.

```javascript
var qAverageQuantity = soItem.$query().$limit(100).$aggregate({
  SALESORDERID: false,
  PRODUCTID: true
}).$addFields({
  averageQuantity: soItem.QUANTITY.$avg()
});
```

11. Specify the order of the result set.

To specify the order in the result set, you can use the `$order()` method, including a number of order criteria as arguments. Each order criteria contains a property "by" with an expression that defines the desired order. Optionally each criterion can contain a flag `$desc` to require a descending order and a `$nullslast` flag. The following example uses two criteria to display the result set first in descending order by the net amount in the header and then ascending order by the item net amount.

```javascript
qSelectedOrders = qSelectedOrders.$order({$by: soHeader.NETAMOUNT, $desc:true}, {$by: soHeader.items.NETAMOUNT});
```

12. Remove duplicates entries from the result set.

The `$distinct` operator removes duplicates from the result set. The following example shows how to display the set of all the currencies used in the sales orders.

```javascript
var qAllCurrencies = soHeader.$query().$project({CURRENCY: true}).$distinct();
```

Related Information

- SAP HANA XS JavaScript API Reference
- SAP HANA XS DB Utilities JavaScript API Reference
2.4.4 Tutorial: Update a CDS Entity Using XS Data Services

You can use the XS Data Services (XSDS) library to update CDS entities as if they were JavaScript objects.

Prerequisites

- A new (or an existing) development workspace in the SAP HANA repository
- A new (or an existing) shared project in the workspace
- This tutorial refers to CDS models that are included in the demonstration content provided with the SAP HANA Interactive Education (SHINE) delivery unit (DU). The SHINE DU is available for download in the SAP Software Download Center.

Note

Access to the SAP Software Download Center is only available to SAP customers and requires logon credentials.

Context

For read-write scenarios, SAP HANA XS Data Services (XSDS) offer a managed mode with automatic entity management and additional consistency guarantees. Managed mode shares CDS imports and transaction handling with unmanaged mode but uses a different set of methods provided by the entity constructors.

Procedure

1. Import the XSDS library and the CDS entities into your application.

   In your entity import, specify a SAP HANA sequence that is used to generate the required keys.

   ```javascript
   // import XSDS client library
   var XSDS = $.import("sap.hana.xs.libs.dbutils", "xsdns");
   // import CDS entity as XSDS entity
   var SOItem = XSDS.$importEntity("sap.hana.democontent.epm.data", "EPM.SO.Item");
   var SOHeader = XSDS.$importEntity("sap.hana.democontent.epm.data", "EPM.SO.Header", {
       SALESORDERID: { $key: "SAP_HANA_DEMO"."sap.hana.democontent.epm.data::salesOrderId" },
       items: {
         $association: {
           $entity: SOItem,
           $viaBacklink: "SALESORDERID"
         }
       }
   });
   ```

2. Retrieve an existing entity instance in managed mode.
The $importEntity() function returns a constructor for the entity imported. To retrieve an existing entity instance in managed mode, run a query using the entity’s key (for example, using $get), or retrieve multiple instances that satisfy a given condition.

```javascript
var order = SOHeader.$get({ SALESORDERID: "0500000236" }); // by key
var orders = SOHeader.$findAll({ LIFECYCLESTATUS: "N", TAXAMOUNT: { $gt: 17000 } }); // by filter
```

3. Use or modify entity instances as required.

Instances of CDS entities are regular JavaScript objects which you can use and modify as required.

```javascript
order.CURRENCY = "USD";
order.HISTORY.CHANGEDAT = new Date();
```

4. Ensure all changes are made persistent in the database.

Calling $save() flushes in-memory changes of the instance and all its reachable associated instances to the database. Only entity instances that have been changed will be updated in the database.

```javascript
order.$save();
```

5. Use the entity constructor to create a new CDS instance.

The key is generated automatically by the SAP HANA sequence supplied during the import of the XSDS library and the CDS entities into your application.

```javascript
var newOrder = new SoHeader({
  TAXAMOUNT": 69.04,
  NETAMOUNT": 190.9,
  GROSSAMOUNT": 325.94,
  CURRENCY": "EUR",
  PARTNERID": "0100000044",
  DELIVERYSTATUS": "I",
  BILLINGSTATUS": "I",
  LIFECYCLESTATUS": "N",
  HISTORY": {
    CHANGEDAT": Date.now(),
    CHANGEDBY": "0000000033",
    CREATEDAT": Date.now(),
    CREATEDBY": "0000000033"
  },
  items: []
});
newOrder.$save();
```

6. Discard any unwanted instances of a CDS entity.

Retrieved CDS entities are stored in the entity manager cache and subject to general JavaScript garbage-collection rules. Use the $discard() function to permanently delete an entity instance from the database.

```javascript
order.$discard();
```

7. Control how associations in a CDS document are followed.

By default, all associations are resolved, that is; association properties store a reference to their associated entity instance. For heavily connected data, this may lead to very large data structures in memory. A “lazy” association will delay the retrieval of the associated instances until the property is actually accessed. The first time the lazy association is accessed, the associated entity is queried from the entity cache or the database. After a lazy association has been resolved, it becomes a normal property of its parent entity instance.
To control how associations are being followed, declare “lazy” associations during the import operation, as shown in the following example:

```javascript
var SOHeader = XSDS.$importEntity("sap.hana.democontent.epm.data", 
"EPM.SO.Header", { 
SALESORDERID: { $key: "\"SAP_HANA_DEMO\". \"sap.hana.democontent.epm.data::salesOrderId\"" }, 
items: { 
$association: { 
$entity: SOItem, 
$viaBacklink: "SALESORDERID", 
$lazy: true 
} 
}
});
```

The retrieval of “Lazy” associations is handled transparently by XSDS.

```javascript
var order = SOHeader.$get({ SALESORDERID: "0500000236" }); // retrieve single SO header
if (order.DELIVERYSTATUS != "D")
    return; // return without loading SO items from database
for (var item in order.items) { … }; // now retrieve items for processing
```

8. Manually control transactions for your application where necessary.

Every SAP HANA XS application using XSDS is associated with one database connection and one transaction. This is also true if the application uses multiple imports of the XSDS library; XS libraries are single instances by default. Entities retrieved from the database are stored in the entity manager cache, and any updates need to be saved explicitly to the database. By default, database saves will automatically commit the changes to the database. However, you can manually control transactions for your application by disabling auto-commit and calling $commit and $rollback explicitly, as illustrated in the following example.

```javascript
// disable auto-commit
XSDS.Transaction.$setAutoCommit(false);
var order = SOHeader.$get({ SALESORDERID: "0500000236" });
order.CURRENCY = "JPY";
order.$save(); // persist update
XSDS.Transaction.$commit(); // commit change
order.CURRENCY = "EUR";
order.$save(); // persist update
order.HISTORY.CHANGEDAT = new Date();
order.$save(); // persist update
XSDS.Transaction.$rollback(); // database rollback
// order #0500000236 now has currency JPY again
```

Related Information

- SAP HANA XS JavaScript API Reference
- SAP HANA XS DB Utilities JavaScript API Reference
2.5 Creating Custom XS SQL Connections

In SAP HANA Extended Application Services (SAP HANA XS), you use the SQL-connection configuration file to configure a connection to the database; the connection enables the execution of SQL statements from inside a server-side JavaScript application with credentials that are different to the credentials of the requesting user.

In cases where it is necessary to execute SQL statements from inside your server-side JavaScript application with credentials that are different to the credentials of the requesting user, SAP HANA XS enables you to define and use a specific configuration for individual SQL connections. Each connection configuration has a unique name, for example, Registration or AdminConn, which is generated from the name of the corresponding connection-configuration file (Registration.xssqlcc or AdminConn.xssqlcc) on activation in the repository. The administrator can assign specific, individual database users to this configuration, and you can use the configuration name to reference the unique SQL connection configuration from inside your JavaScript application code.

The following code example shows how to use the XS SQL connection AdminConn.xssqlcc.

```javascript
function test() {
    var body;
    var conn;
    $.response.status = $.net.http.OK;
    try {
        conn = $.db.getConnection("sap.hana.sqlcon::AdminConn");
        var pStmt = conn.prepareStatement("select CURRENT_USER from dummy");
        var rs = pStmt.executeQuery();
        if (rs.next()) {
            body = rs.getNString(1);
        }
        rs.close();
        pStmt.close();
    } catch (e) {
        body = "Error: exception caught";
        $.response.status = $.net.http.BAD_REQUEST;
    }
    if (conn) {
        conn.close();
    }
    $.response.setBody( body );
}

test();
```

To use the SQL connection from your application during runtime, you must bind the SQL connection configuration to a registered database user and assign the user the appropriate permissions, for example, by assigning a pre-defined role to the user. To maintain this user mapping, SAP HANA XS provides the Web-based SAP HANA XS Administration Tool. When the run-time status of the XSSQLCC artifact is set to active, SAP HANA generates a new auto user (with the name XSSQLCC_AUTO_USER_[...]). The new user is granted the permissions specified in a role, which can be assigned using the parameter role_for_auto_user - either in the design-time artifact or the run-time configuration.

Note
Access to the tools provided by the XS Administration Tool requires the privileges granted by one or more specific user roles.
To use the SAP HANA XS Administration Tool to view or maintain an XS SQL connection configuration, you need the privileges granted by the following SAP HANA XS roles:

- `sap.hana.xs.admin.roles::SQLCCViewer`
  Required to display the available SQL Connections and the current user mapping
- `sap.hana.xs.admin.roles::SQLCCAdministrator`
  Required to modify details of the user mapping; the SQLCCAdministrator role includes the role SQLCCViewer.

**Troubleshooting Tips**

If you are having problems implementing the XS SQL connection feature using an `.xssqlcc` configuration, check the following points:

- **User permissions**
  Make sure that you grant the necessary user the activated role (for example, `sap.hana.xs.admin.roles::SQLCCAdministrator`). You can use the developer tools to grant roles (or privileges), as follows:

  - SAP HANA studio
    In the Systems view of the Administration Console perspective, choose Security > Users.
  - SAP HANA Web-based Development Workbench
    In the Security tool, expand the Users node, choose the target (or add a new) user, and use the Granted roles tab.
  - XS Administration Tools
    In the SQL Connection Details tab of the XSSQLCC artifact’s run time configuration. To edit user/role details here, you will need the role SQLCCAdministrator and, in addition, the appropriate administrator permissions required to set up (and assign roles to) a database user.

- **File location**
  Make sure that the SQL-role configuration file (.xssqlcc) you create is located in the same package as the application that references it.

- **Logon dependencies**
  If your application is using form-based logon (configured in the application’s .xsaccess file), make sure the libxsauthenticator library is present and specified in the list of trusted libraries displayed in the SAP HANA studio’s Administration Console perspective (Administration > Configuration Tab > xsengine.ini > application_container > application_list). If the libxsauthenticator library is not in the list of authorized libraries, an SAP HANA system administrator must add it.

  - **Note**
    If you have to authorize libxsauthenticator, you might also need to refresh the Web page in your browser the next time you want to access .xssqlcc to display the logon dialog again.
2.5.1 Create an XS SQL Connection Configuration

The .xssqlcc file enables you to establish a database connection that you can use to execute SQL statements from inside your server-side JavaScript application with credentials that are different to the credentials of the requesting user.

Prerequisites

- Access to an SAP HANA system
- Access to a development workspace and a shared project.
- The application package structure in which to save the artifacts you create and maintain in this task
- The SQL connection configuration file (.xssqlcc) you create must be located in the same package as the application that uses it.
- You have the privileges granted in the following SAP HANA user roles:
  - sap.hana.xs.admin.roles::SQLCCViewer
  - sap.hana.xs.admin.roles::SQLCCAdministrator

**Note**

This tutorial combines tasks that are typically performed by two different roles: the application developer and the database administrator. The developer would not normally require the privileges of the SAP HANA administrator or those granted by the SQLCCAdministrator user role.

Context

In this tutorial, you learn how to configure an SQL connection that enables you to execute SQL statements from inside your server-side JavaScript application with credentials that are different to the credentials of the user requesting the XSJS service.

To configure and use an XS SQL configuration connection file, perform the following steps:

Procedure

1. Start the SAP HANA studio.
   a. Open the SAP HANA Development perspective.
   b. Open the Project Explorer view.
2. Create the application descriptors for the new application.
   a. In the SAP HANA studio’s Project Explorer view, right-click the folder acme.com.xs.testApp1 where you want to create the new (.xsapp) file.
   b. In the context-sensitive popup menu, choose New Other...
c. In the Select a Wizard dialog, choose **SAP HANA ➤ Application Development ➤ XS Application Descriptor File**

The file-creation wizard adds the required file extension `.xsapp` automatically.

d. Choose **Finish**.

**Tip**

Files with names that begin with the period (.), for example, `.xsapp` or `.xsaccess`, are sometimes not visible in the Project Explorer. To enable the display of all files in the Project Explorer view, use the **Customize View ➤ Available Customization** option and clear all check boxes.

e. Activate the application descriptor file.

In the SAP HANA studio’s Project Explorer view, right-click the new (.xsapp) file and choose **Team ➤ Activate** from the context-sensitive popup menu.

3. Create the application access file for the new application.

a. In the SAP HANA studio’s SAP HANA Development perspective.

b. In the Project Explorer view, right-click the folder where you want to create the new (.xsaccess) file.

c. In the context-sensitive popup menu, choose **New ➤ Other...**

d. In the Select a Wizard dialog, choose **SAP HANA ➤ Application Development ➤ XS Application Access File**

The file-creation wizard adds the required file extension `.xsaccess` automatically and enables direct editing of the file.

**Note**

The default name for the core application-access file is `.xsaccess` and cannot be changed.

e. Choose **Finish**.

f. Check the contents of the `.xsaccess` file.

```json
{
    "exposed" : true,
    "authentication" : { "method" : "Form"},
    "prevent_xsrp" : true
}
```

The entries in the `.xsaccess` file ensure the following:

- Application data can be exposed to client requests
- Username and password credentials are required for logon authentication
- Protection against cross-site, request-forgery attacks is enabled

g. Activate the application access file.

In the SAP HANA studio’s Project Explorer view, right-click the new (.xsaccess) file and choose **Team ➤ Activate** from the context-sensitive popup menu.

4. Create the XS SQL connection configuration file.

Browse to the folder in your project workspace where you want to create the new SQL connection configuration file and perform the following steps:
The SQL connection configuration file (.xssqlcc) you create must be located in the same package as the application that references it.

a. Right-click the folder where you want to save the XS SQL connection configuration file and choose **New > Other... > Application Development > XS SQL Connection Configuration File** in the context-sensitive popup menu.

b. Enter the name of the SQL connection configuration file in the **File Name** box, for example, AdminConn.

c. Choose **Finish** to save the changes and commit the new XS SQL connection configuration file in the repository.

5. Configure the details of the SQL connection that the XS JavaScript service will use.

a. Define the required connection details.

```javascript
{
  "description" : "Admin SQL connection",
  "role_for_auto_user" : "com.acme.roles::JobAdministrator"
}
```

**Tip**

Replace the package path (com.acme.roles) and role name (JobAdministrator) with the suitable ones for your case.

b. Activate the XS SQL connection configuration file.

In the SAP HANA studio’s **Project Explorer** view, right-click the new (.xssqlcc) file and choose **Team > Activate** from the context-sensitive popup menu.

**Note**

Activating the SQL connection configuration file AdminConn.xssqlcc creates a catalog object with the name sap.hana.xs.testApp1::AdminConn, which can be referenced in a XS JavaScript application.

6. Write an XS JavaScript application that calls the XS SQL connection configuration.

To create a preconfigured SQL connection using the configuration object AdminConn, for example, from inside your JavaScript application code, you must reference the object using the object name with the full package path, as illustrated in the following code example.

```javascript
function test() {
  var body;
  var conn;
  $.response.status = $.net.http.OK;
  try {
    conn = $.db.getConnection("sap.hana.xs.testApp1::AdminConn");
```
```javascript
var pStmt = conn.prepareStatement("select CURRENT_USER from dummy");
var rs = pStmt.executeQuery();
if (rs.next()) {
    body = rs.getNString(1);
}
rs.close();
pStmt.close();
} catch (e) {
    body = "Error: exception caught";
    $.response.status = $.net.http.BAD_REQUEST;
}
if (conn) {
    conn.close();
}
$.response.setBody( body );
test();
```

7. Save the changes to the artifacts you have created.

### Note

Saving a file in a shared project automatically commits the saved version of the file to the repository. You do not need to explicitly commit it again.

8. Activate the changes in the repository.
   a. In the Project Explorer view, locate and right-click the package containing the new XS SQL and XS JavaScript artifacts.
   b. In the context-sensitive pop-up menu, choose Team >> Activate.

9. Bind the SQL connection configuration to a user.
   You use the Web-based SAP HANA XS Administration Tool to configure the runtime elements of the XS SQL connection.
   a. Start the SAP HANA XS Administration Tool.
      The SAP HANA XS Administration Tool is available on the SAP HANA XS Web server at the following URL: http://<WebServerHost>:80<SAPHANAinstance>/sap/hana/xs/admin/.

### Note

Access to details of HTTP destinations in the SAP HANA XS Administration Tool requires the credentials of an authenticated database user and one of the following SAP HANA roles:
- sap.hana.xs.admin.roles::SQLCCViewer
- sap.hana.xs.admin.roles::SQLCCAdministrator

b. In the XS Applications tab, expand the nodes in the application tree to locate the application testApp.
c. Choose AdminConn to display details of the XS SQL configuration connection.

10. Set the run-time status of the XS SQL connection configuration.
    You must change the status runtime status of the XS SQL connection configuration to Active. This run-time status can only be changed by an administrator. When the run-time status of the XSSQL connection configuration is set to active, SAP HANA automatically generates a new user (XSSQLCC_AUTO_USER_[...]) for the XSSQL connection configuration object and assigns the role defined in role_for_auto_user to the new auto-generated user.
2.5.1.1  The SQL Connection Configuration File

The SQL-connection configuration file specifies the details of a connection to the database that enables the execution of SQL statements from inside a server-side (XS) JavaScript application with credentials that are different to the credentials of the requesting user.

If you want to create an SQL connection configuration, you must create the configuration as a flat file and save the file with the suffix .xssqlcc, for example, MYSQLconnection.xssqlcc. The new configuration file must be located in the same package as the application that references it.

**Note**
An SQL connection configuration can only be accessed from an SAP HANA XS JavaScript application (.xsjs) file that is in the same package as the SQL connection configuration itself. Neither subpackages nor sibling packages are allowed to access an SQL connection configuration.

The following example shows the composition and structure of a configuration file AdminConn.xssqlcc for an SAP HANA XS SQL connection called AdminConn. On activation of the SQL connection configuration file AdminConn.xssqlcc (for example, in the package sap.hana.sqlcon), an SQL connection configuration with the name sap.hana.sqlcon::AdminConn is created, which can be referenced in your JavaScript application. In the xssqlcc artifact, you can set the following values:

- **description**
  A short description of the scope of the xs sql connection configuration
- **role_for_auto_user**
  The name of the role to be assigned to the auto user (if required) that the XSSQL connection uses, and the absolute path to the package where the role definition is located in the SAP HANA repository.

    ```json
    sap.hana.sqlcon::AdminConn.xssqlcc
    {
      "description" : "Admin SQL connection"
      "role_for_auto_user" : "com.acme.roles::JobAdministrator"
    }
    ```

The run-time status of an XSSQL connection configuration is **inactive** by default; the run-time status can only be activated by an SAP HANA user with administrator privileges, for example, using the *SAP HANA XS Administration Tools*. When the run-time status of the XSSQLCC artifact is set to **active**, SAP HANA generates a new auto user (with the name XSSQLCC_AUTO_USER_[...]) and assigns the role defined in role_for_auto_user to the new auto-generated user.

**Tip**
In the *SAP HANA XS Administration Tools*, it is possible to view and edit both the the user’s parameters and the role’s definition.
To create a preconfigured SQL connection using the configuration object `AdminConn`, for example, from inside your JavaScript application code, you reference the object using the object name and full package path, as illustrated in the following code example.

```javascript
{   conn = $.db.getConnection("sap.hana.sqlcon::AdminConn");
}
```

### Related Information

- SQL Connection Configuration Syntax [page 79]
- Create an XS SQL Connection Configuration [page 74]

## 2.5.1.2 SQL Connection Configuration Syntax

The XS SQL connection-configuration file `.xssqlcc` uses pairs of keywords and values to define the SQL connection.

### Example

#### The XS SQL Connection Configuration `.xssqlcc` File

**Code Syntax**

```javascript
{
    "description" : "Admin SQL connection",
    "role_for_auto_user" : "com.acme.roles::JobAdministrator"
}
```

### description

A short description of the selected SQL connection configuration.

**Sample Code**

```
"description" : "Admin SQL connection"
```
**role_for_auto_user**

The name of (and package path to) the role assigned to be assigned to the new user that is automatically generated on activation of the XSSQL connection-configuration artifact.

```
Sample Code

"role_for_auto_user" : "com.acme.roles::JobAdministrator"
```

Activating the design-time XSSQL connection configuration generates a run-time object whose status is “inactive” by default; the run-time status must be set to **active** by an SAP HANA user with administrator privileges, for example, using the **SAP HANA XS Administration Tools**. When the run-time status of the XSSQLCC artifact is set to **active**, SAP HANA generates a new auto user (with the name **XSSQLCC_AUTO_USER_[....]**) and assigns the role defined in **role_for_auto_user** to the new auto-generated user.

**Related Information**

- The SQL Connection Configuration File [page 78]
- Create an XS SQL Connection Configuration [page 74]

### 2.6 Setting the Connection Language in SAP HANA XS

HTTP requests can define the language used for communication in the HTTP header **Accept-Language**. This header contains a prioritized list of languages (defined in the Browser) that a user is willing to accept. SAP HANA XS uses the language with the highest priority to set the language for the requested connection. The language setting is passed to the database as the language to be used for the database connection, too.

In server-side JavaScript, the session object’s **language** property enables you to define the language an application should use for a requested connection. For example, your client JavaScript code could include the following string:

```
var application_language = $.session.language = 'de';
```

**Note**

Use the language-code format specified in BCP 47 to set the session language, for example: “en-US” (US English), “de-AT” (Austrian German), “fr-CA” (Canadian French).

As a client-side framework running in the JavaScript sandbox, the SAP UI5 library is not aware of the **Accept-Language** header in the HTTP request. Since the current language setting for SAPUI5 is almost never the same as the language specified in the SAP HANA XS server-side framework, SAPUI5 clients could have problems relating to text displayed in the wrong language or numbers and dates formatted incorrectly.
The application developer can inform the SAP UI5 client about the current server-side language setting, for example, by adding an entry to the `<script>` tag in the SAPUI5 HTML page, as illustrated in the following examples:

- **Script tag parameter:**

  ```html
  <script id="sap-ui-bootstrap"
  type="text/javascript"
  src="/sap/ui5/1/resources/sap-ui-core.js"
  data-sap-ui-theme="sap_goldreflection"
  data-sap-ui-libs="sap.ui.commons"
  data-sap-ui-language="de">
  </script>
  ```

- **Global sap-ui-config object:**

  ```javascript
  window["sap-ui-config"] = {"language": "de"};
  </script>
  […]
  <script id="sap-ui-bootstrap"
  […]
  </script>
  ```

  The `sap-ui-config` object must be created and filled before the `sap-ui-bootstrap` script.

It is important to understand that the session starts when a user logs on, and the specified language is associated with the session. Although the user can start any number of applications in the session, for example, in multiple Browser tabs, it is not possible to set a different language for individual applications called in the session.

### Setting the Session Language on the Server side

The script tag for the SAPUI5 startup can be generated on the server side, for example, using the `.session.language` property to set the `data-sap-ui-language` parameter. Applications that have the SAPUI5 `<script>` tag in a static HTML page can use this approach, as illustrated in the following example:

```html
<script id="sap-ui-bootstrap"
  type="text/javascript"
  src="/sap/ui5/1/resources/sap-ui-core.js"
  data-sap-ui-theme="sap_goldreflection"
  data-sap-ui-libs="sap.ui.commons"
  data-sap-ui-language="$UI5_LANGUAGE$">
  </script>
```

The called XSJS page can be instructed to replace the `$UI5_LANGUAGE$` parameter, for example, with the value stored in `$session.language` when loading the static HTML page.
Setting the Session Language with an AJAX Call

You can include an HTTP call in the static HTML page to fetch the correct language from the server using some server-side JavaScript code, as illustrated in the following example:

```javascript
<script>
var xmlHttp = new XMLHttpRequest();
xmlHttp.open( "GET", "getAcceptLanguage.xsjs", false );
xmlHttp.send( null );
window["sap-ui-config"] = {
    "language" : xmlHttp.getResponseHeader("Content-Language")
}
</script>
```

This approach requires an XSJS artifact (for example, `getAcceptLanguage.xsjs`) that responds to the AJAX call with the requested language setting, as illustrated in the following example:

```javascript
$.response.contentType = "text/plain";
$.response.headers.set("Content-Language", $.session.language);
$.response.setBody("");
```

2.7 Scheduling XS Jobs

Scheduled jobs define recurring tasks that run in the background. The JavaScript API `$.jobs` allows developers to add and remove schedules from such jobs.

If you want to define a recurring task, one that runs at a scheduled interval, you can specify details of the job in a `.xsjob` file. The time schedule is configured using `cron-like` syntax. You can use the job defined in an `.xsjob` file to run an XS Javascript or SQLScript at regular intervals. To create and enable a recurring task using the `xsjob` feature, you perform the following high-level tasks:

**Note**

The tasks required to set up a scheduled job in SAP HANA XS are performed by two distinct user roles: the application developer and the SAP HANA administrator. In addition, to maintain details of an XS job in the S**AP HANA XS Administration Tool**, the administrator user requires the privileges assigned by the user role `sap.hana.xs.admin.roles::JobAdministrator`.

<table>
<thead>
<tr>
<th>Step</th>
<th>Task</th>
<th>User Role</th>
<th>Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Create the function or script you want to run at regular intervals</td>
<td>Application developer</td>
<td>Text editor</td>
</tr>
<tr>
<td>2</td>
<td>Create the job file <code>.xsjob</code> that defines details of the recurring task</td>
<td>Application developer</td>
<td>Text editor</td>
</tr>
<tr>
<td>Step</td>
<td>Task</td>
<td>User Role</td>
<td>Tool</td>
</tr>
<tr>
<td>------</td>
<td>----------------------------------------------------------------------</td>
<td>---------------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>3</td>
<td>Maintain the corresponding runtime configuration for the xsjob</td>
<td>SAP HANA administrator</td>
<td>XS Job Dashboard</td>
</tr>
<tr>
<td>4</td>
<td>Enable the job-scheduling feature in SAP HANA XS</td>
<td>SAP HANA administrator</td>
<td>XS Job Dashboard</td>
</tr>
<tr>
<td>5</td>
<td>Check the job logs to ensure the job is running according to schedule.</td>
<td>SAP HANA administrator</td>
<td>XS Job Dashboard</td>
</tr>
</tbody>
</table>

### Related Information

- The XSJob File [page 86]
- Tutorial: Schedule an XS Job [page 83]
- XS Job File Keyword Options [page 88]

### 2.7.1 Tutorial: Schedule an XS Job

The `xsjob` file enables you to run a service (for example, an XS JavaScript or an SQLScript) at a scheduled interval.

### Prerequisites

- You have access to an SAP HANA system
- You have the privileges granted in the SAP HANA user role `sap.hana.xs.admin.roles::JobAdministrator`
- You have the privileges granted in the SAP HANA user role `sap.hana.xs.admin.roles::HTTPDestAdministrator`

**Note**

This tutorial combines tasks that are typically performed by two different roles: the application developer and the database administrator. The developer would not normally require the privileges granted to the `JobAdministrator` user role, the `sap.hana.xs.admin.roles::HTTPDestAdministrator` user role, or the SAP HANA administrator.

### Context

In this tutorial, you learn how to schedule a job that triggers an XS JavaScript application that reads the latest value of a share price from a public financial service available on the Internet. You also see how to check that the XS job is working and running on schedule.
To schedule an XS job to trigger an XS JavaScript to run at a specified interval, perform the following steps:

**Procedure**

1. Create the application package structure that contains the artifacts you create and maintain in this tutorial.

   Create a root package called `yahoo`. You use the new `yahoo` package to contain the files and artifacts required to complete this tutorial.

   ```
   /yahoo/
   .xsapp               // application descriptor
   yahoo.xsjob          // job schedule definition
   yahoo.xshttpdest     // HTTP destination details
   yahoo.xsjs           // Script to run on schedule
   ```

2. Write the XS JavaScript code that you want to run at the interval defined in an XS job schedule.

   The following XS JavaScript connects to a public financial service on the Internet to check and download the latest prices for stocks and shares.

   Create an XS JavaScript file called `yahoo.xsjs` and add the code shown in the following example:

   ```
   function readStock(input) {
     var stock = input.stock;
     var dest = $.net.http.readDestination("yahoo", "yahoo");
     var client = new $.net.http.Client();
     var req = new $.web.WebRequest($.net.http.GET, "/d/quotes.csv?f=a&s=" + stock);
     client.request(req, dest);
     var response = client.getResponse();
     var stockValue;
     if(response.body)
       stockValue = parseInt(response.body.asString(), 10);
     var sql = "INSERT INTO stock_values VALUES (NOW(), ?)";
     var conn = $.db.getConnection();
     var pstmt = conn.prepareStatement(sql);
     pstmt.setDouble(1, stockValue);
     pstmt.execute();
     conn.commit();
     conn.close();
   }
   ```

   Save and activate the changes in the SAP HANA Repository.

   **Note**

   Saving a file in a shared project automatically commits the saved version of the file to the repository. To explicitly commit a file to the repository, right-click the file (or the project containing the file) and choose `Team` > `Commit` from the context-sensitive popup menu.

3. Create an HTTP destination file using the wizard to provide access to the external service (via an outbound connection).

   Since the financial service used in this tutorial is hosted on an external server, you must create an HTTP destination file, which provides details of the server, for example, the server name and the port to use for HTTP access.
To maintain the runtime configuration details using the Web-based XS Administration Tool you need the privileges granted in the SAP HANA user role sap.hana.xs.admin.roles::HTTPDestAdministrator.

Create a file called `yahoo.xshttpdest` and add the following content:

```javascript
host = "download.finance.yahoo.com";
port = 80;
```

Save and activate the changes in the SAP HANA Repository.

4. Create the XS job file using the wizard to define the details of the schedule at which the job runs.

The XS job file uses a cron-like syntax to define the schedule at which the XS JavaScript must run. This job file triggers the script `yahoo.xsjs` on the 59th second of every minute and provides the name “SAP.DE” as the parameter for the stock value to check.

Create a file called `yahoo.xsjob` and add the following code:

```json
{
  "description": "Read stock value",
  "action": "yahoo:yahoo.xsjs::readStock",
  "schedules": [
    {
      "description": "Read current stock value",
      "xscron": "* * * * * 59",
      "parameter": {
        "stock": "SAP.DE"
      }
    }
  ]
}
```

Save and activate the changes in the SAP HANA Repository.

5. Maintain the XS job’s runtime configuration.

You maintain details of an XS Job’s runtime configuration in the XS Job Dashboard.

a. Start the SAP HANA XS Administration Tool.

The SAP HANA XS Administration Tool is available on the SAP HANA XS Web server at the following URL: http://<WebServerHost>:80<SAPHANAInstance>/sap/hana/xs/admin/.

b. Maintain the details of the XS job.

To maintain details of an XS job using the Web-based XS Administration Tool you need the privileges granted in the SAP HANA user role sap.hana.xs.admin.roles::JobAdministrator.

You need to specify the following details:

- **User**
  The user account in which the job runs, for example, `SYSTEM`.

- **Password**
  The password required for user, whose account is used to run the job.

- **Locale**
  The language encoding required for the locale in which the job runs, for example, `en_US`.

- **Start/Stop time**
An optional value to set the period of time during which the job runs. Enter the values using the syntax used for the SAP HANA data type `LocalDate` and `LocalTime`, for example, **2014-11-05 00:30:00** (thirty minutes past midnight on the 5th of November 2014).

- **Active**
  Enable or disable the job schedule

c. Save the job.

Choose **Save Job** to save and activate the changes to the job schedule.

6. Enable the job-scheduling feature in SAP HANA XS.

This step requires the permissions granted to the SAP HANA administrator.

i. **Note**

It is not possible to enable the scheduler for more than one host in a distributed SAP HANA XS landscape.

a. In the **XS Job Dashboard** set the **Scheduler Enabled** toggle button to **YES**.

Toggling the setting for the **Scheduler Enabled** button in the **XS Job Dashboard** changes the value set for the SAP HANA configuration variable `xsengine.ini scheduler enabled`, which is set in the Configuration tab of the SAP HANA studio’s Administration perspective.

7. Check the job logs to ensure the XS job is active and running according to the defined schedule.

You can view the `xsjob` logs in the **XS Job Dashboard** tab of the **SAP HANA XS Administration Tool**.

i. **Note**

To maintain details of an XS job using the Web-based **XS Administration Tool** you need the privileges granted in the SAP HANA user role `sap.hana.xs.admin.roles::JobAdministrator`.

If the job does not run at the expected schedule, the information displayed in the `xsjob` logs includes details of the error that caused the job to fail.

### Related Information

- The **XS Job File** [page 86]
- **XS Job-File Keyword Options** [page 88]

#### 2.7.1.1 The XS Job File

The `.xsjob` file defines the details of a task that you want to run (for example, an XS JavaScript or an SQLScript) at a scheduled interval.

The XS job file uses a **cron**-like syntax to define the schedule at which the service defined in an XS JavaScript or SQLScript must run, as you can see in the following example, which runs the specified job (the stock-price checking service `yahoo.xsjs`) on the 59th second minute of every minute.

```
When defining the job schedule in the xsjob file, pay particular attention to the entries for the following keywords:

- **action**
  Text string used to specify the path to the function to be called as part of the job.

Example:
```
"action": "<package_path>::XSJS_Service::<FunctionName>",
```

- **description**
  Text string used to provide context when the XSjob file is displayed in the SAP HANA XS Administration tool.

- **xscron**
  The schedule for the specified task (defined in the “action” keyword); the schedule is defined using cron-like syntax.

- **parameter**
  A value to be used during the action operation. In this example, the parameter is the name of the stock SAP.DE provided as an input for the parameter (stock) defined in the readStock function triggered by the xsjob action. You can add as many parameters as you like as long as they are mapped to a parameter in the function itself.

The following examples illustrate how to define an xscron entry including how to use expressions in the various xscron entries (day, month, hour, minutes,...):

- **2013 * * fri 12 0 0**
  Every Friday of 2013 at 12:00 hours

- *** * 3:-2 * 12:14 0 0**
  Every hour between 12:00 and 14:00 hours on every day of the month between the third day of the month and the second-last day.

**Tip**

In the day field, third from the left, you can use a negative value to count days backwards from the end of the month. For example, *** * -3 * 9 0 0** means: three days from the end of every month at 09:00.

- *** * * * */5 * **
  Every five minutes (*/5) and at any point (*) within the specified minute.
**Note**

Using the asterisk (*) as a wild card in the seconds field can lead to some unexpected consequences, if the scheduled job takes less than 59 seconds to complete; namely, the scheduled job restarts on completion. If the scheduled job is very short (for example, 10 seconds long), it restarts repeatedly until the specified minute ends.

To prevent short-running jobs from restarting on completion, schedule the job to start at a specific second in the minute. For example, * * * * * /5 20 indicates that the scheduled job should run every five minutes and, in addition, at the 20th second in the specified minute.

- * * * -1.sun 9 0 0
  Every last Sunday of a month at 09:00 hours

**Related Information**

XS Job File Keywords [page 88]
Tutorial: Schedule an XS Job [page 83]

2.7.1.2  **XS Job File Keyword Options**

The XS job file .xsjob uses a number of keywords to define the job that must be run at a scheduled interval.

**Example**

The XS Job (.xsjob) File

```json
{
  "description": "Read stock value",
  "action": "yahoo:yahoo.xsjs::readStock",
  "schedules": [  
    {  
      "description": "Read current stock value",
      "signature_version": 1,
      "xscron": "* * * * * 59",
      "parameter": {  
        "stock": "SAP.DE"
      }
    }
  ]
}
```

description

```json
{
  "description": "Read stock value",
```
The `description` keyword enables you define a text string used to provide context when the XS job is displayed for maintenance in the SAP HANA XS Administration Tool. The text string is used to populate the `Description` field in the SCHEDULED JOB tab.

**action**

```json
{
    "action": "myapps.finance.yahoo.yahoo.xsjs::readStock",
}
```

The `action` keyword enables you to define the function to run as part of the XS job, for example, an XS JavaScript or an SQLScript. The following syntax is required: 

```
"action": 
  "<package.path>:\<XSJS_Service>.xsjs::\<functionName>"
```

**i Note**

If you want to use the action to call an SQLScript, replace the name of the XSJS service in the example, with the corresponding SQLScript name.

**schedules**

```json
{
    "schedules": [
        {
            "description": "Read current stock value",
            "xscron": "* * * * * 59",
            "parameter": {
                "stock": "SAP.DE"
            }
        }
    ]
}
```

The `schedules` keyword enables you define the details of the XS job you want to run. Use the following additional keywords to provide the required information:

- **description** (optional)
  Short text string to provide context
- **xscron**
  Uses cron-like syntax to define the schedule at which the job runs
- **parameter** (optional)
  Defines any values to be used as input parameters by the (XSJS or SQLScript) function called as part of the job
**signature_version**

```json
{
  "signature_version": 1,
}
```

The **signature_version** keyword enables you manage the version “signature” of an XS job. You change the XS job version if, for example, the parameter signature of the job action changes; that is, an XS job accepts more (or less) parameters, or the types of parameters differ compared with a previous version of an XS job. On activation in the SAP HANA Repository, the signature of an XS job is compared to the previous one and, if the job’s signature has changed, any job schedules created at runtime will be deactivated.

**Note**

The default value for **signature_version** is 0 (zero).

Deactivation of any associated runtime job schedules prevents the schedules from silently failing (no information provided) and enables you to adjust the parameters and reactivate the job schedules as required, for example, using the enhanced XS JS API for schedules. Schedules defined in a design-time XS Job artifact are replaced with the schedules defined in the new version of the XS job artifact.

**Tip**

Minor numbers (for example, 1.2) are not allowed; the job scheduler interprets “1.2” as “12”.

**xscron**

```json
{
  "schedules": [
    {
      "description": "Read current stock value",
      "xscron": "* * * * * 59",
      "parameter": {
        "stock": "SAP.DE"
      }
    }
  ]
}
```

The **xscron** keyword is used in combination with the **schedules** keyword. The **xscron** keyword enables you to define the schedule at which the job runs. As the name suggests, the **xscron** keyword requires a cron-like syntax.

The following table explains the order of the fields (*) used in the “**xscron**” entry of the **.xsjob** file and lists the permitted value in each field.

<table>
<thead>
<tr>
<th>xscron Field (* from left to right)</th>
<th>Meaning and Permitted Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year</td>
<td>4-digit, for example, 2013</td>
</tr>
</tbody>
</table>
### xscron Field (* from left to right) | Meaning and Permitted Value
--- | ---
Month | 1 to 12
Day | -31 to 31
DayOfWeek | mon, tue, wed, thu, fri, sat, sun
Hour | 0 to 23
Minute | 0 to 59
Second | 0 to 59

#### Note

Using the asterisk (*) as a wild card in the seconds field can lead to some unexpected consequences, if the scheduled job takes less than 59 seconds to complete; namely, the scheduled job restarts on completion. If the scheduled job is very short (for example, 10 seconds long), it restarts repeatedly until the specified minute ends.

To prevent short-running jobs from restarting on completion, schedule the job to start at a specific second in the minute. For example, */5 */5 20 indicates that the scheduled job should run every five minutes and, in addition, at the 20th second in the specified minute. The job starts at precisely 20 seconds into the specified minute and runs only once.

The following table illustrates the syntax allowed to define expressions in the “xscron” entry of the .xsjob file.

#### Table 12:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Where used...</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Anywhere</td>
<td>Any value</td>
</tr>
<tr>
<td>*/a</td>
<td>Anywhere</td>
<td>Any a-th value</td>
</tr>
<tr>
<td>a:b</td>
<td>Anywhere</td>
<td>Values in range a to b</td>
</tr>
<tr>
<td>a:b/c</td>
<td>Anywhere</td>
<td>Every c-th value between a and b</td>
</tr>
<tr>
<td>a:y</td>
<td>DayOfWeek</td>
<td>On the a-th occurrence of the weekday y (a = -5 to 5)</td>
</tr>
<tr>
<td>a.b.c</td>
<td>Anywhere</td>
<td>a or b or c</td>
</tr>
</tbody>
</table>

### parameter

```javascript
{
    "schedules": [
        {
            "description": "Read current stock value",
            "xscron": "* * * * * * 59",
            "parameter": {
                "stock": "SAP.DE",
                "share": "BMW.DE"
            }
        }
    ]
}
```
The optional `parameter` keyword is used in combination with the `schedules` keyword. The `parameter` keyword defines values to be used as input parameters by the XSJS function called as part of the job. You can list as many parameters as you like, separated by a comma (,) and using the JSON-compliant syntax quotations ("").

2.7.2 Add or Delete a Job Schedule during Runtime

The `$jobs` application programming interface (API) enables you to manipulate the schedules for an XS job at runtime.

**Context**

You can use the `$jobs.JobSchedules` API to add a schedule to (or delete a schedule from) a job defined in an `.xsjob` file at runtime.

**Note**

Schedules added at runtime are deleted when the `.xsjob` file is redeployed.

**Procedure**

1. Create an XS job file using the `.xsjob` syntax.

   **Note**
   
   If you have already created this XS job file, for example, in another tutorial, you can skip this step.

   Create a file called `yahoo.xsjob` and add the following code:

   ```json
   {
   "description": "Read stock value",
   "action": "yahoo:yahoo.xsjs::readStock",
   "schedules": [ {
               "description": "Read current stock value",
               "xscron": "* * * * * 59",
               "parameter": {
               "stock": "SAP.DE"
               }
   }
   ]
   }
   ```

   Save and activate the changes in the SAP HANA Repository.
2. Create the XS JavaScript (.xsjs) file you want to use to define the automatic scheduling of a job at runtime.

Name the file schedule.xsjs.

3. Use the $.jobs JavaScript API to add or delete a schedule to a job at runtime.

The following example schedule.xsjs adds a new schedule at runtime for the XS job defined in yahoo.xsjob, but uses the parameter keyword to change the name of the stock price to be checked.

```javascript
var myjob = new $.jobs.Job({uri:"yahoo.xsjob"});
var id = myjob.schedules.add({
description: "Query another stock",
xcron: "** * * * * * /10",
parameter: {
    stock: "APC.DE"
}
});
// delete a job schedule
// myjob.schedules.delete( {id: id } );
```

4. Save and activate the changes in the SAP HANA Repository.

5. Call the XS JavaScript service schedule.xsjs to add the new job schedule at runtime.

Related Information

SAP HANA XS JavaScript Reference
XS Job File Keyword Options [page 88]

2.8 Tracing Server-Side JavaScript

The SAP HANA XS server-side JavaScript API provides tracing functions that enable your application to write predefined messages in the form of application-specific trace output in the xsengine trace files (xsengine*.trc) according to the trace level you specify, for example, “info” (information) or “error”.

If you use the server-side JavaScript API to enable your application to write trace output, you can choose from the following trace levels:

- debug
- info
- warning
- error
- fatal
For example, to enable debug-level tracing for your JavaScript application, include the following code:

```
$.trace.debug("request path: " + $.request.path);
```

### 2.8.1 Trace Server-Side JavaScript Applications

The server-side JavaScript API for SAP HANA XS enables you to activate the writing of messages into a trace file; the following trace levels are available: debug, error, fatal, info, and warning.

**Context**

By default, applications write messages of severity level error to the `xsengine*.trc` trace files; you can increase the trace level manually, for example, to fatal. In SAP HANA XS, the following steps are required to enable trace output for your server-side JavaScript application:

**Procedure**

1. Open the SAP HANA studio.
2. In the Systems view, double-click the SAP HANA instance to open the Administration view for the repository where your server-side JavaScript source files are located.
3. Choose the Trace Configuration view.
4. In the Database Trace screen area, choose Edit Configuration. The Edit Configuration icon is only visible if you have the required privileges on the selected SAP HANA system.
If the Database Trace screen area is not displayed, check that you are using a version of SAP HANA studio that is compatible (the same as) with the SAP HANA server where you want to set up tracing.

5. Select the Show All Components checkbox.
6. Enter the partial or full name of your application into the search box.
7. Find the trace matching your application name and choose the trace level you want to use to generate output.
   The application name is the location (package) of the .xsapp file associated with the application you are tracing. The trace topic is named xsa:<package.path> <appName>.

8. Choose Finish to activate the trace level changes.

### 2.8.2 View XS JavaScript Application Trace Files

The server-side JavaScript API for SAP HANA XS enables you to instruct your JavaScript applications to write application-specific trace messages in the xsengine*.trc trace files. You can view the trace files in the Diagnosis Files tab page of the Administration perspective in the SAP HANA studio.

#### Context

The trace levels “debug”, “error”, “fatal”, “info”, and “warning” are available. To view trace output for your server-side JavaScript application, perform the following steps:

#### Procedure

1. Open the SAP HANA studio.
2. In the Systems view, double-click the SAP HANA instance to open the Administration view for the repository where your server-side JavaScript source files are located.
3. Choose the Diagnosis Files tab page.
4. In the Filter box, enter a string to filter the list of search files displayed, for example, xsengine*.trc.
   The timestamp displayed in the Modified column does not always reflect the precise time at which the trace file was written or most recently modified.
5. Locate the trace file for your SAP HANA XS application and double-click the entry to display the contents of the selected trace-file in a separate tab page.
2.9 Debugging Server-Side JavaScript

SAP HANA XS provides a set of dedicated tools to enable you to debug the XS JavaScript code that you write. To trigger debugging, you need an XS JavaScript configuration.

Overview

To prepare the system for debugging, you need to perform the following high-level steps:

- Ensure all prerequisites listed below are met.
- Create a debug configuration or choose an existing debug configuration to use.
- Set breakpoints in the file you want to debug.
- Execute XS JavaScript debugging.

To trigger debugging, you need to choose an XS JavaScript configuration; each configuration type represents a different starting point for debugging an XS JavaScript file. To debug XS JavaScript, you must choose one of the following types of configuration:

- **XS JavaScript**
  Use to debug a stand-alone XS JavaScript service.

- **XS JavaScript: Manual Session**
  Use to debug an XS JavaScript initiated from any remote client using that specific XS session.

- **XS JavaScript: HTML-based**
  Use to debug an XS JavaScript initiated from HTML.

- **XS JavaScript: XS OData-based**
  Use to debug an XS JavaScript initiated from an XS OData breakout.

**Note**

Before you start debugging server-side JavaScript on SAP HANA Extended Application Services (SAP HANA XS), first check that you have fulfilled the following prerequisites:

- Ensure the delivery unit for SAPHANA XS debugging tools is imported
  To import the HANA_XS_BASE.tgz delivery unit (DU) that contains the XS JavaScript debugging tools, in SAP HANA Studio, choose the option **New > Import > Delivery Unit**.
Enable debugging on the system level:

1. Ensure the SAP HANA XS Web server is running, and that you have HTTP access to the following URL:
   http:<SAP_HANA_HOSTNAME>:<PortNumber>:/

---

Select File

Select the required file to import content objects.

- **Select file**
  - Server
  - Client

- **File**
  - /usr/sap/XSSYS/global/hdb/auto_content/HANA_XS_BASE.tgz

- **Actions**
  - Overwrite inactive versions
  - Activate objects
    - Bypass clientside validation rules

Object import simulation

<table>
<thead>
<tr>
<th>Status</th>
<th>Object name</th>
<th>Package name</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔️</td>
<td>config</td>
<td>sap.hana.xs.admin.config</td>
</tr>
<tr>
<td>✔️</td>
<td>Debugger</td>
<td>sap.hana.xs.debugger</td>
</tr>
<tr>
<td>✔️</td>
<td>Debugger</td>
<td>sap.hana.xs.debugger.api</td>
</tr>
</tbody>
</table>
2. Start SAP HANA Studio and open the Administration perspective.

3. In the Configuration tab, add a section called `xsengine.ini` (if it does not exist) and add (or set) the following parameter: `enabled=true`.

   - Assign the debugging role to your user
     SAP HANA XS provides a dedicated debugger user role; the role must be assigned to any user who wants to start a debugging session for server-side JavaScript in SAP HANA XS.

   - Assign the debugging role to another user (optional)
     You can grant a user global access to any of your debug sessions or grant access to a debug session that is flagged with a specified token. You can also restrict access to a debug session to a specified period of time.

   **Note**
   By default, other users do not have the permissions required to access your XS JavaScript debugging sessions. However, SAP HANA XS enables you to grant access to your debug sessions to other users, and vice versa.
2. In the **Systems** view, expand the **Security** node and double-click the user to whom you want to assign the debugger role.

3. In the **Granted Roles** view, choose the [+] icon and, in the **Select Role** dialog, enter **debugger** to search for the debugger role and choose **OK**.

---

**Note**

Debugging can also be done in other settings, for example, when a server is cloud-based or when it is a secured server.

- **Debugging with HANA Cloud Platform (HCP) (optional)**
  
  Debugging using HCP requires prerequisites to be fulfilled. For more information, see **Getting Started** in the SAP HANA Cloud Documentation.

- **Debugging using a secure server (optional)**
  
  Debugging using a secure server requires specific prerequisites to be fulfilled. For more information, see **Configure SSL for SAP HANA Studio Connections** in the SAP HANA Security Guide.

---

**Related Information**

- [Debug Session Access](#)
- [The XSJS Debugger Role](#)

---

### 2.9.1 Create a Debug Configuration

**Context**

To create an XS JavaScript debug configuration, do the following:

**Procedure**

1. Open the **Debug** perspective.
2. Choose ![Open Perspective](image) and select **Debug Configurations**.
3. Choose the debug configuration type you want to debug.
   
   It can be one of the following:
   - **XS JavaScript**: Use to debug a standalone XS JavaScript service.
   - **XS JavaScript: Manual Session**: Use to debug an XS JavaScript initiated from any remote client using that specific XS session.
○ **XS JavaScript: HTML-based**: Use to debug an XS JavaScript initiated from HTML.

○ **XS JavaScript: XS OData-based**: Use to debug an XS JavaScript initiated from an XS OData breakout.

**Note**

You can use an existing configuration, change it or create a new debug configuration by selecting the file type to use for debugging, and clicking on the New button.

4. In the General tab, enter a name for the new debug configuration.

5. The external browser is your default debug mode. You can also choose to debug using the internal SAP HANA Studio.

6. To build the URL, select the file and resource path or add parameters where relevant. Parameters can be entered using raw text or a table format.
   ○ When creating a manual session debug configuration, you only need to select the system to debug.
   ○ If a system is logged off, it will not show in the system dropdown list.

7. You can include stored procedures in your debug configuration which will enable SQL script to be debugged along with XS JavaScript. If your XS JavaScript code triggers stored procedures, you can set breakpoints and debug them using the same debug configuration. You do not need to create a separate, dedicated debug configuration for the stored procedures.

8. For configurations with an Input Parameters tab, select the method, and enter the header and body information as relevant. Body details can be entered as raw text or in the x-www-form-urlencoded format.

9. Choose **Apply**.

10. Choose **Close** to save the configuration for later use or **Debug** to start debugging.

### 2.9.2 Execute XS JavaScript Debugging

SAP HANA studio enables you to debug XS JavaScript files, including setting breakpoints and inspecting variables.

**Context**

To enable the display of more helpful and verbose information for HTTP 500 exceptions on the SAP HANA XS Web server, add the parameter `developer_mode` to the `xsengine.ini` `httpserver` section and set it to `true`. `xsengine.ini` is in the Configuration tab of the Administration perspective in SAP HANA studio.

**Prerequisites**

- Ensure that debugging is enabled on the SAP HANA server.
- You have the debugger role assigned to your user.
- User authentication is enabled. This is required to open the debugging session.
To start debugging, do the following:

**Procedure**

1. In a Web browser, run the XS JavaScript source file that you want to debug.
2. Create or choose a debug configuration for debug sessions for a specific SAP HANA installation.
   a. Open the *Debug* view.
   b. Choose a debug configuration.
      You can also create a new configuration by doing one of the following:
      ○ From the menu bar, click *Run > Debug Configurations*.
      ○ Select the file to be debugged and right-click, choose *Debug As > Debug Configurations*.
   c. Choose *Apply*.
   d. Choose *Close*.
3. Set Breakpoints
   Set breakpoints in the JavaScript code by double-clicking the left vertical ruler.
4. Run the new debug configuration for your server by choosing *Run* and selecting your debug configuration.
   You can also run the debug configuration by doing one of the following:
   ○ From the menu bar, click *Run > Debug Configurations* then choose the debug configuration you want to use.
   ○ Select the file to be debugged and right-click on it, and then choose *Debug As*.
   ○ From *Debug Configurations*, click the debug configuration you want to use.
   ○ For an HTML file, select the file to be debugged and right-click on it, then choose *Debug As > HTML*.

**Note**
When using the external debug mode, you can only have one open XS debug session per system. This is relevant for the following debug configurations:
2.9.2.1 The Debug Perspective

SAP HANA studio includes a dedicated debug perspective, which provides the tools needed by a developer who wants to debug server-side JavaScript code.

Application developers can use the SAP HANA studio’s Debug perspective to perform standard debugging tasks, for example: starting and resuming code execution, stepping through code execution, adding breakpoints to the code. Developers can also inspect variables and check the validity of expressions. The following views are available as part of the standard Debug perspective:

- **Debug**
  Displays the stack frame for the suspended or terminated threads for each target you are debugging. Each thread in your program appears as a node in the tree. You can also see which process is associated with each target.

- **Breakpoints**
  Displays a list of the breakpoints set in the source file you are currently debugging.

- **Variables**
  Displays a list of the variables used in the source file you are currently debugging.

- **Expressions**
  Displays global variables, such as `.request` and other SAP HANA XS JavaScript API objects.

- **Outline**
  Displays a structural view of the source file you are currently debugging. You can double-click an element to expand and collapse the contents.

- **Source-code editor**
  SAP HANA studio uses the file extension (for example, `.js` or `.xsjs`) of the source file you want to debug and opens the selected file in the appropriate editor. For example, files with the `.js` or `.xsjs` file extension are displayed in the built-in JavaScript editor.

**Note**

Unified Debugger

In the unified debugger, if you choose to include the SQL script layer in the debugging session, you will see the targets of both the XS JavaScript and SQL script in the debug view.

If a breakpoint is set in the XS JavaScript or in an SQL script procedure, you will see the breakpoints in the breakpoint view. The debugger will stop at the breakpoints in the relevant XS JavaScript or in the SQL script as usual. SQL script debugging behavior is the same in the SQL script debugger as it is in the unified.
debugger, with the exception of the call stack behavior. For more information about debugging SQL script procedures, see Debugging Procedures.

2.9.2.2 The XSJS Debugger Role

The JavaScript debugger included with SAP HANA Extended Application Services (SAP HANA XS) requires user authentication to start a debug session. SAP HANA XS includes a dedicated debugger role, which defines the permissions needed by a developer who wants to debug server-side JavaScript code.

Debugging application code is an essential part of the application-development process. SAP HANA Extended Application Services (SAP HANA XS) includes a debug perspective, a debug view, and a dedicated debugger role that must be assigned to any developer who wants to debug XS JavaScript. The debugging role is named sap.hana.xs.debugger::Debugger and can be assigned to a user (or a role) either with the standard role-assignment feature included in SAP HANA studio (the Application Privileges tab in the Security area of the Systems view) or in a design-time, role-configuration file (.hdbrole).

Since a developer primarily needs to debug his own HTTP calls, the following limitations apply to a debug session:

- Only authenticated users can start a debug session, for example, by providing a user name and password when logging in to a debug session
- A user can debug his own sessions.
A user can debug any session to which access has been explicitly granted, for example, by the owner of the session.

**Note**

It is also possible to use SSL for debugging. If SSL is configured, the server redirects the Web-socket connect call to the corresponding SSL (secure HTTP) URL, for example, if sent by plain HTTP.

SAP HANA studio includes a graphical user interface (GUI) which you can use to grant access to debug sessions at both the session level and the user level.

### 2.9.2.3 Debug Session Access

You can grant other developers access to the debug sessions you use for debugging server-side JavaScript on SAP HANA XS.

By default, other users are not allowed to access your XSJS debugging sessions. However, SAP HANA XS provides a tool that enables you to grant access to your debugging sessions to other users, too.

**Note**

You can grant a user global access to any of your sessions or grant access to a session that is flagged with a specified token. You can also restrict access to a debug session to a specified period of time.

The **XS Debugging** tool is available on the SAP HANA XS Web server at the following URL:

```
<SAPHANAWebServer>80<SAPHANAInstance>/sap/hana/xs/debugger/.
```

When you are grant access to your debugging session, the following options are available:

- **User Name**
  The name of the database user who requires access to your debug session

- **Privilege Expires**
  The point in time that marks the end of the period for which access to one or more debug sessions is allowed.

- **grant debug permission for all sessions**
  You can grant a user global access to any of your debug sessions.

**Restriction**

The user you grant access to must already be registered and authenticated in the SAP HANA database.

- **grant debug permission for this session only**
  You can grant access to a debug session that is flagged with a specific token:

**Restriction**

Unauthenticated users must use the token-based option.

The following rules apply to access to debug sessions flagged with a token:

- The session used for granting access to the debug sessions is flagged automatically.
The session token is distributed by means of a session cookie; the cookie is inherited by any session created with the current browser session.

- **Session Name**
  A freely definable name that can be used to distinguish your debug session in the context of multiple sessions.

### Related Information

- The XSJS Debugger Role [page 103]
- Debugging Server-Side JavaScript [page 96]

### 2.9.3 Troubleshoot Server-Side JavaScript Debugging

When debugging your JavaScript code, you sometimes need to solve problems, not only with the code itself, but the configuration of the sessions and the tools you use to perform the debugging.

### Prerequisites

- Start a Web-browser session with the SAP HANA server **before** starting a debug session.
  Make sure you open a session with the SAP HANA server by calling an XS JavaScript file from your Web browser **before** starting the debug operation.
- Select the session ID.
  Before starting to debug, select the session whose ID is specified in the `xsSessionId` cookie in your open Web-browser session.

### Context

If you are having problems using the embedded debugging tools to debug your server-side XSJS (JavaScript) code, check the following solutions:

- **Breakpoints**
  The execution of your XS JavaScript code is not stopping at a breakpoint.
- **Network connections**
  Your SAP HANA server is behind a proxy or a firewall.
**Procedure**

1. Restart the SAP HANA studio with the -clean option.

   ```
   Sample Code
   hdbstudio.exe -clean
   ```

   To determine if a clean restart of SAP HANA studio is required, check if the Breakpoints view in SAP HANA studio’s Debug perspective displays the breakpoints as type SAP HANA XSE Script, as follows:
   a. In the Breakpoints view, choose the View Menu.
   b. Choose Group By Breakpoint Types

2. Remove breakpoints.

   Try removing all the existing breakpoints from the debug session and recreating them.

3. Create a new workspace.

   If a restart of SAP HANA studio with the -clean option does not solve the problem of unrecognized breakpoints, it might be necessary to create a new Eclipse (not repository) Workspace.

4. Set the Active Provider feature to manual.

   If your SAP HANA server is behind a proxy or firewall, check that your Network Connections are configured for using a proxy, as follows:

   a. In SAP HANA studio, choose Window Preferences General Network Connections.
   b. Set the Active Provider to Manual.
   c. The default setting is Native.
   d. Update the schemas.
   e. Add the relevant proxy host and port.

5. Configure the Debug Configuration Connection properties.

   a. Select and right-click your SAP HANA system.
   c. Check that your system’s SAP HANA XS properties match the Debug Configuration Connection properties.

**Related Information**

*Execute XS JavaScript Debugging [page 100]*
2.10 Testing XS JavaScript Applications

SAP HANA provides a test framework called XSUnit that enables you to set up automatic tests for SAP HANA XS applications.

The test framework SAP HANA XSUnit (XSUnit) is a custom version of the open-source JavaScript test framework, Jasmine, adapted for use with SAP HANA XS. You can use the XSUnit test framework to automate the tests that you want to run for SAP HANA XS applications, for example, to test the following elements:

- Server side JavaScript code
- SQLScript code (stored procedures and views)
- Modeled calculation views

To use the tools and features provided with the XSUnit test framework, you must perform the following high-level steps:

1. Set up the client-side environment:
   - Install the latest version of SAP HANA studio (optional).
   - Ensure that the hdbclient tool is installed and running.
2. Set up the server-side environment.
   - The XSUnit test framework is included in the delivery unit HANA_TEST_TOOLS, which you must install manually, for example, using the SAP HANA studio or the SAP HANA Application Lifecycle Management tool. After the installation completes, the tools are available in the package sap.hana.testtools.

   \[ \text{Note} \]
   Importing a delivery unit into an SAP HANA system requires the REPO.IMPORT privilege, which is normally granted only to the system administrator.

3. Maintain SAP HANA user privileges.
   - The system administrator must grant test users the privileges required to use the test tools. The privileges are defined in roles, which the SAP HANA administrator can assign to all developers by default.
4. Maintain the test schema (optional).
   - If you write XSUnit tests that are designed to test database content, you require a test schema in which you create test tables during your test execution and fill the tables with test data. To avoid conflicts when different users run the same test at the same time, it is recommended that individual developers place test tables in their corresponding user schema.

   \[ \text{Note} \]
   You must ensure that _SYS_REPO has select permission to schema where the tables are located (for example, either your user schema or the test schema).

   \[ \text{grant select on schema MY_TEST_SCHEMA to _SYS_REPO with grant option; } \]

**Related Information**

Automated Tests with XSUnit in SAP HANA [page 108]
2.10.1 Automated Tests with XSUnit in SAP HANA

XSUnit is an integrated test environment that enables you to set up automatic tests for SAP HANA XS applications.

People developing applications in the context of the SAP HANA database need to understand how to implement a test-automation strategy. Especially for new applications which are designed to work exclusively with SAP HANA, it is a good idea to consider the adoption of best practices and tools.

If you want to develop content that is designed to run specifically in SAP HANA, it is strongly recommended to use the XSUnit test framework that is integrated in SAP HANA XS; this is the only way to transport your test code with your SAP HANA content. The XSUnit tools are based on a Java Script unit test framework that uses Jasmine as the underlying test library.

Test Isolation and Simulation

To write self-contained unit tests that are executable in any system, you have to test the various SAP HANA objects in isolation. For example, an SAP HANA view typically has dependencies to other views or to database tables; these dependencies pass data to the view that is being tested and must not be controlled or overwritten by the test. For this reason, you need to be able to simulate dependencies on the tested view. XSUnit includes a test-isolation tool that provides this functionality; it allows you to copy a table for testing purposes.

Note

Although you cannot copy a view for testing purposes, you can create a table that acts like a view.

All (or specific) dependencies on any tables or views are replaced by references to temporary tables, which can be created, controlled, and populated with values provided by the automated test.

Test Data

Preparing and organizing test data is an important part of the process of testing SAP HANA content such as views and procedures; specific data constellations are required that have to be stable in order to produce reliable regression tests. In addition, test-isolation tools help reduce the scope of a test by enabling you to test a view without worrying about dependent tables and views. Limiting the scope of a test also helps to reduce the amount of data which needs to be prepared for the test.
2.10.2 Application Development Testing Roles

Dedicated roles enable developers to access and use the tools provided with the SAP HANA XS test framework (XSUnit).

To grant access to the SAP HANA XS test framework that enables developers to set up automatic testing for SAP HANA applications, the SAP HANA system administrator must ensure that the appropriate roles are assigned. The following table lists the roles that are available; one (or more) of the listed roles must be assigned to the application developers who want to use the XSUnit testing tools.

Table 13: Default Developer Testing Roles

<table>
<thead>
<tr>
<th>Role Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sap.hana.testtools.common::TestExecute</td>
<td>Enables you to view the persisted test results produced by the XSUnit test framework and to execute the examples included in the demonstration package (sap.hana.testtools.demo).</td>
</tr>
<tr>
<td>sap.hana.xs.debugger::Debugger</td>
<td>Enables you to debug your server side JavaScript (test-)code</td>
</tr>
<tr>
<td>sap.hana.xs.ide.roles::Developer</td>
<td>Enables you to view source files in the SAP HANA Web-based Work Bench (Web IDE)</td>
</tr>
</tbody>
</table>

2.10.3 Test an SAP HANA XS Application with XSUnit

Use the XSUnit tools to set up automated testing of your applications in SAP HANA XS.

Prerequisites

The following prerequisites apply if you are using SAP HANA studio to set up and run tests with XSUnit:

- SAP HANA studio
  - You will need access to a shared development project in the SAP HANA system where you plan to run the tests.
Context

If you want to develop content that is designed to run specifically with SAP HANA, you can use the XSUnit tools that are integrated in SAP HANA XS. The XS Unit tools are based on a Java Script unit test framework that uses Jasmine as the underlying test library.

Procedure

1. Create an Eclipse project.
   If you want to create your first unit test, you need an XS Project that will contain the test code. You can either create a new shared XS Project or, if a shared project already exists, you can checkout and import the existing project from the SAP HANA Repository. Within that project you can structure your tests in folders.

   To create a shared Eclipse project, start SAP HANA studio and, in the SAP HANA Development perspective, perform the following steps:
   a. In the Systems view, add the SAP HANA system you want to work and test on.
   b. In the Repositories view, add a repository workspace for your SAP HANA system
   c. Create and share a project of type XS Project.

   ➤ Tip
   You can also checkout and import an existing project from the SAP HANA Repository.

2. Create an XSUnit test.
   XSUnit test files are XSlibrary files (files with the .xsjslib suffix).
   a. Create an XSlibrary file, for example, called `<MyFirstTest>.xsjslib`.
      You can use the file-creation Wizard in SAP HANA studio, for example, File ➤ New ➤ Other ➤ SAP HANA Development ➤ XS JavaScript Library File ➤
   b. Add the following content to the new XSlibrary test file `<MyFirstTest>.xsjslib`:

   /*global jasmine, describe, beforeOnce, beforeEach, it, xit, expect*/
   describe("My First Test Suite using Jasmine", function() {
     it('should show an assertion that passes', function() {
       expect(1).toBe(1);
     });
     it("should show an negative assertion", function() {
       expect(true).not.toBe(false);
     });
     it("should throw an expected error", function() {
       expect(function() {
         throw new Error("expected error");
       }).toThrowError("expected error");
     });
     // xit = this test case is excluded
     xit("should show an assertion that fails", function() {
       expect(1).toBe(2);
     });
   });
The ESLint tool that SAP HANA studio uses to check your XSJS code tells you that functions (for example, describe, it, expect) do not exist. This is not true; the functions do exist but they are defined in another library. To ensure that ESLint considers functions to be globally available, add the following comment as the first line of the XSUnit test file:
/*global jasmine, describe, beforeOnce, beforeEach, it, xit, expect*/

`Note`
You can extend the list of globally available functions to include any additional functions that you use in your test.

c. Save the test file.
d. Activate the test file.

In the SAP HANA studio’s SAP HANA Development perspective, open the Project Explorer view, right-click the test file, and choose Team > Activate.

3. Execute the XSUnit test.

How you execute an XSUnit test depends on the development tool suite you are using, for example, SAP HANA studio.

You execute an XSUnit test by entering the following URL in a Web Browser:

http://<hostname>:<port>/sap/hana/testtools/unit/jasminexs/TestRunner.xsjs?package=<packageName>

Where <hostname> is the name of the SAP HANA system where you are running your application test, and <port> is the port number that the SAP HANA instance is available on.

The TestRunner tool recursively searches the package <packageName> for any files with the suffix .xsjslib whose names match the pattern “*Test”.

`Note`
If you want to search for a string other than “*Test”, you must pass a custom pattern to TestRunner using the parameter pattern.

Related Information

XSUnit Test Run Options [page 114]

2.10.3.1 XSUnit's Enhanced Jasmine Syntax

The XSUnit test framework is a custom version of the JavaScript test framework Jasmine adapted to suit SAP HANA XS.

A test specifications begin with a call to the global Jasmine function describe. The describe functions define suites that enable you to group together related test suites and specifications. Test-suite specifications
are defined by calling the global Jasmine function `it`. You can group several test suites in one test file. The following code snippet shows one test suite (introduced by the function “describe”) and two test specifications, indicated by the function “it”.

```javascript
/*jslint undef:true */
describe('testSuiteDescription', function() {
    beforeEach(function() {
        // beforeEach function is called before each specification
    });
    afterEach(function() {
        // afterEach function is called after each specification
    });
    it('testSpecDescription', function() {
        expect(1).toEqual(1);
    });
    it('anotherTestSpecDescription', function() {
        expect(1).not.toEqual(0);
    });
});
```

To enable a test suite to remove any duplicate setup and teardown code, Jasmine provides the global functions `beforeEach` and `afterEach`. As the name implies, the `beforeEach` function is executed before each specification in the enclosing suite and all sub-suites; the `afterEach` function is called after each specification. Similarly, the special methods `beforeOnce` and `afterOnce` are called once per test suite.

- **beforeOnce**
  Executed once before all specifications of the test suite
- **afterOnce**
  Executed once after all specifications of the test suite

### Database Connection Setup

The XSUnit framework provides a managed database connection called `jasmine.dbConnection`, which is globally available. You can use it in the following scenarios:

- Directly (in the function “it”)
- In the functions “beforeEach” and “afterEach”
- In other functions defined in your test libraries
- In imported libraries (if you have moved test code to external libraries)

One obvious advantage of this is that you no longer have to pass the database connection as a parameter or define it as a global variable. The `jasmine.dbConnection` is opened automatically and rolled back (and closed). However, if you want to persist your data, you have to manually call `commit()` on `jasmine.dbConnection`.
2.10.3.2 XSUnit Test Tools Syntax

Example syntax for the functions, assertions, and parameters required by the SAP HANA XSUnit test tools.

The following code example lists the most commonly used functions and assertions used in the XSUnit syntax. For more information about the assertions used, for example, `toBe`, `toBeTruthy`, or `toBeFalsy`, see Assertions.

```javascript
/*global jasmine, describe, beforeOnce, beforeEach, it, xit, expect*/
describe("My First Test Suite using Jasmine", function() {
  beforeOnce(function() {
    // beforeOnce is called only one time for all specs
  });
  beforeEach(function() {
    // beforeEach is called before each specs
  });
  // it = test case specification it("should show an assertion that passes", function() {
  var array = [{foo: 'bar', baz: 'quux'}, {bar: 'foo', quux: 'baz'}];
  expect(1).toBeTruthy();
  expect(12).toBe(jasmine.any(Number));
  expect(array).toContain(jasmine.objectContaining({foo: 'bar'}));
  });
  it("should show an negative assertion", function() {
    expect(true).not.toBe(false);
  });
});
```
it("should throw an expected error", function() {
  expect(function() {
    throw new Error("expected error");
  }).toThrowError("expected error");
});
// xit = this test case is excluded
xit("should show an assertion that fails", function() {
  expect(1).toBe(2);
});

XSUnit Assertions and Parameters

The following code example lists the most commonly used assertions, shows the required syntax, and the expected parameters.

```
expect(actual).toBe(expected);
expect(actual).toBeFalsy();
expect(actual).toBeTruthy();
expect(actual).toEqual(expected);
expect(actualArray).toContain(expectedItem);
expect(actual).toBeNull();
expect(actualNumber).toBeCloseTo(expectedNumber, precision);
expect(actual).toBeDefined();
expect(actual).toBeUndefined();
expect(actualString).toMatch(regExpression);
expect(actualFunction).toThrowError(expectedErrorMessage);
expect(actualFunction).toThrowError(expectedErrorType, expectedErrorMessage);
expect(actualTableDataSet).toMatchData(expected, keyFields);
expect(actual).toBeLessThan(expected);
expect(actual).toBeGreaterThan(expected);
```

2.10.3.3 XSUnit Test Run Options

The XSUnit tool suite includes a generic tool that you can use to run tests.

You can start the XSUnit test-running tool (TestRunner.xsjs) by entering the following URL in a Web Browser:

```
```

The following table lists the parameters that you can use to control the behavior of test-runner tool. If you execute the test runner without specifying the pattern parameter, only the tests in *Test.jslib files are discovered (and run) within the package hierarchy.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>coverage</td>
<td>enables or disables code coverage reporting.</td>
</tr>
<tr>
<td>exclude</td>
<td>specifies a comma-separated list of folders to</td>
</tr>
<tr>
<td></td>
<td>exclude from the test coverage.</td>
</tr>
<tr>
<td>pattern</td>
<td>specifies the file pattern to include in tests.</td>
</tr>
<tr>
<td>tests</td>
<td>specifies a comma-separated list of tests to run</td>
</tr>
<tr>
<td>testRunner</td>
<td>specifies the test runner tool.</td>
</tr>
</tbody>
</table>

**Note**

You can specify multiple parameters by separating each parameter=value pair with the ampersand character (&), for example: `coverage=true&exclude=sap.hana.tests`
<table>
<thead>
<tr>
<th>Name</th>
<th>Mandatory</th>
<th>Description</th>
</tr>
</thead>
</table>
| package  | yes       | Package that acts as starting point for discovering the tests. If not otherwise specified by parameter “pattern” all .xsjslib files in this package and its sub-packages conforming to the naming pattern “*Test” will be assumed to contain tests and will be executed.  

package=sap.hana.testtools.demo |
| pattern  | no        | Naming pattern that identifies the .xsjslib files that contain the tests. If not specified, the pattern “*Test” is applied. You can use question mark (?) and asterisk (*) as wildcards to match a single or multiple arbitrary characters, respectively. To match all ”Suite.xsjslib” files, use the following code:  

pattern=Suite |
| format   | no        | Specifies the output format the test runner uses to report test results. By default, the results will be reported as HTML document. This parameter has no effect if a custom reporter is provided via parameter “reporter”. To display outputs results using the JSON format, use the following code:  

format=json |
| reporter | no        | Complete path to module that provides an implementation of the Jasmine reporter interface. With this parameter a custom reporter can be passed to publish the test results in an application specific format. To specify the reporter interface, use the following code:  

reporter=sap.hana.testtools.unit.jasminexs.reporter.db.dbReporter  

| Note     |            | i Note  

format=db produces the same result |
| tags     | no        | Comma-separated list of tags which is used to define the tests to be executed.  

tags=integration,long_running |
| profile  | no        | Name of a “profile” defined in the test which filters the tests to be executed on the basis of tags.  

profile=end2end |
| coverage | no        | Activate code coverage measurement for all server-side (XS) JavaScript code that is executed by the tests or which is in the scope of a specified package.  

coverage=true  

coverage=sap.hana.testtools.mockstar  

coverage=true&exclude=sap.hana.testtools.mockstar.tests |
2.10.3.4 XSUnit Test Examples

XSUnit includes a selection of test packages that demonstrate the scope of tests you can perform on an SAP HANA XS application.

The following table lists the test packages included in the XSUnit test framework. The table also indicates the name of the test file and provides a quick overview of the scope of the test.

### Note

If you want to have a look at the code in the tests, checkout the package `sap.hana.testtools.demo` as an XS project to your local workspace.

<table>
<thead>
<tr>
<th>Package Name</th>
<th>Test Name (.xsjslib)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tests.getting_started</td>
<td>myFirstTest</td>
<td>Shows the usage of some basic Jasmine matchers as well as the usage of custom matchers <code>toMatchData</code> and <code>toEqualObject</code> that are supported by the extended Jasmine version.</td>
</tr>
<tr>
<td>tests.attribute_view_1</td>
<td>AT_PRODUCTS_Test</td>
<td>Shows how to configure mockstar in order to replace a CDS entity with a test table. Be aware that this test does not make sense, as this attribute test tests nothing at all - no logic, no joins,...</td>
</tr>
<tr>
<td>tests.graphic_calcview_1</td>
<td>CA_ORDERS_Test</td>
<td>Tests a copy of the graphical calculation view where the direct dependent tables are replaced by test tables.</td>
</tr>
<tr>
<td>tests.graphic_calcview_3</td>
<td>CA_OPEN_AMOUNT_Test</td>
<td>Tests the integration with the analytic view but replaces the dependencies to the tables with test tables. This example test shows how to upload data from a comma-separated-values (CSV) file into the test tables</td>
</tr>
<tr>
<td>tests.hdbprocedure_with_cds</td>
<td>CreateProductTest</td>
<td>Tests a non-read-only HDBProcedure with table in/out parameters while replacing the underlying Core Data Services (CDS) entities with test tables.</td>
</tr>
<tr>
<td>tests.hdbprocedure_with_hdbview</td>
<td>GetInvoicesTest</td>
<td>Tests an HDBProcedure with scalar in and view out parameters while replacing a dependent hdbview with a test table.</td>
</tr>
<tr>
<td>tests.hdbprocedure_with_hierarchy-view</td>
<td>HierarchyProcedureTest</td>
<td>Tests an HDBProcedure that includes a hierarchy view while replacing all underlying CDS entities with test tables.</td>
</tr>
<tr>
<td>tests.hdbprocedure_with_hdbprocedure</td>
<td>CreateProductTest</td>
<td>Tests an HDBProcedure while replacing a dependent hdbprocedure with an hdbprocedure that was created for testing.</td>
</tr>
<tr>
<td>Package Name</td>
<td>Test Name (.xsjslib)</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>tests.http_service</td>
<td>whoAmIServiceTestE2E</td>
<td>Tests an http service and checks if it returns the expected value. This test is not automatically executed since the SAP HANA instance needs to be maintained by the system administrator.</td>
</tr>
<tr>
<td>tests.procedure_1</td>
<td>PR_OPEN_AMOUNT_Test</td>
<td>Tests a copy of the stored procedure where the directly dependent tables are replaced with test tables.</td>
</tr>
<tr>
<td>tests.scripted_calcview_1</td>
<td>CA_ABC_PRODUCTS_Test</td>
<td>Tests a copy of the scripted calculation view where the directly dependent analytic view is replaced with a test table.</td>
</tr>
<tr>
<td>tests.scripted_calcview_2</td>
<td>CA_OPEN_AMOUNT_SCRIPTED_W_PROCEDURE_Test</td>
<td>Tests the integration with the called stored procedures but replaces the dependencies to the tables with test tables.</td>
</tr>
<tr>
<td>apps.rating.tests</td>
<td>validatorTest</td>
<td>Tests a simple server-side (XS) JavaScript.</td>
</tr>
<tr>
<td></td>
<td>dataAccessorTest</td>
<td>Tests the database layer of server-side (XS) JavaScript using Jasmine spyOn() for testing in isolation.</td>
</tr>
<tr>
<td></td>
<td>oDataTestE2E</td>
<td>Checks the accessibility of an OData service and tests an OData service without dependencies using mockstar.</td>
</tr>
<tr>
<td></td>
<td>ratingServicesTestE2E</td>
<td>Tests an XS JavaScript service (end-to-end scenario test).</td>
</tr>
<tr>
<td>tests</td>
<td>myMockstarEnvironment</td>
<td>Shows how to enhance the mockstarEnvironment library to add further reuse functions or change the behaviour slightly to suit the context.</td>
</tr>
</tbody>
</table>

2.10.3.5 The Mockstar Test Environment

Mockstar is a tool that is designed to enable you to isolate SAP HANA content in tests run by an automated test suite.

To write self-contained unit tests that are executable in any system, it is essential to be able to test the selected SAP HANA objects in isolation. For a typical unit test using the XSUnit tools, you need to be able to change any direct dependencies between the tested objects and other views or tables with references to simple tables. For integration tests, rather than change the direct dependencies to a view or a table, you might need to change dependencies between the dependent views (deeper in the dependency hierarchy).

Mockstar is a tool that is specifically designed to enable you to isolate test objects, for example, a view or procedure. Mockstar allows you to create a copy of the tested view or procedure and substitute the dependency to a another view or table with a table that is stored in a test schema. It is strongly recommended to use a dedicated schema for the tests; in this test schema, you have write permissions and, as a result, full control over the data in the tables and views.
The Mockstart test-isolation tool provides the following features:

- Creates a copy of the SAP HANA object to test (for example, a view or database table); the copied object retains the same business logic as the original one object, but replaces some or all dependencies.
- Replaces the (static) dependencies to tables or views with temporary tables
- Supports deep dependency substitution
  Mockstar can determine dependencies deep within a hierarchy of dependencies and copy only the necessary parts of the hierarchy.

Mockstar tools are included in the delivery unit HANA_TEST_TOOLS, which you must install manually, for example, using the SAP HANA studio or the SAP HANA Application Lifecycle Management tool. After the installation completes, the Mockstar tools are available in the package sap.hana.testtools.mockstar.

**Note**

Importing a delivery unit into an SAP HANA system requires the REPO.IMPORT privilege, which is normally granted only to the system administrator.

### 2.10.3.6 Mockstar Environment Example Syntax

A basic example of the syntax required to set up the Mockstar test environment.

The following example shows a simple setup using standard locations.

**Note**

The names of schemas, tables, and views used in the following code example are intended to be for illustration purposes only.

```javascript
var mockstarEnvironment = $.import('sap.hana.testtools.mockstar',
'mockstarEnvironment');
describe('testSuiteDescription', function() {
  var testEnvironment = null;
  beforeOnce(function() {
    var definition = {
      schema : 'SCHEMA',
      model : {
        schema : '_SYS_BIC',
        name : 'modelName' //e.g. package/MODEL
      },
      substituteTables : {
        "table" : { name : 'package::TABLE' }
      },
      substituteViews : {
        "view" : {
          schema : '_SYS_BIC',
          name : 'package/VIEW'
        }
      }
    },
    testEnvironment = mockstarEnvironment.defineAndCreate(definition);
  });
});
```
2.10.3.7 XSUnit Troubleshooting Solutions

Use trace files and other tools to fix problems with test operations.

The Mockstar test-isolation tools write helpful information in the SAP HANA trace files. You can adapt the trace level, for example, to debug to ensure the right amount and type of information is written during the test run. Note that you need the corresponding administration role to be able to change the trace-level settings in SAP HANA. The trace files are written in the trace component xsa:sap.hana.testtools (truncated to “xsa:sap.hana.tes” in the trace files).

Tip

As an alternative to reading the trace files directly, you can also use the SQL console to select data from the table M_MERGED_TRACES.

This section contains information about the problems that developers frequently encounter during test runs:

- SAP HANA Test Tools Version [page 119]
- The Library is Not Part of an Application [page 119]
- Error for Cloned OData Service [page 120]
- Duplicate Entries When Inserting Test Data [page 120]
- Test Table Already Exists [page 120]
- Test Model Activation Fails [page 121]
- No Entries Returned From Copied Test Model [page 121]
- No Test Data Inserted into Test Table [page 121]
- TestRunner Tool Times Out [page 122]
- Test Model Creation is Aborted [page 123]
- Database Connections in XSUnit Test [page 123]

SAP HANA Test Tools Version

Which version of the SAP HANA test tools suite is installed?

1. Start SAP HANA studio
2. Open the SAP HANA Modeler perspective.
3. In the Quick Launch window, choose Delivery Units...
4. Choose HANA_TEST_TOOLS.

Import Error: The Library is Not Part of an Application

If the test runner tool shows the following error:

```
import: the library is not part of an application
```

The JavaScript library you want to test can only be loaded when there is an application descriptor (.xsapp file) defined within the package hierarchy. The application descriptor is the core file that you use to describe an
application’s framework within SAP HANA XS. If your tests are not part of your application package hierarchy, it is recommended you to create an .xsapp file in the context of the XS Project that contains the tests.

**Error for Cloned OData Service**

The following error message is displayed when testing access to an OData service in SAP HANA XS:

```
404 - Not found: Error for cloned OData Service (.xsodata)
```

Try the following solutions:
1. Try to access the generated service directly in a separate Web browser.
2. Check whether the file (xsodata service definition) exists, has been activated in the SAP HANA repository, and is in the expected target folder.
3. Ensure that the target folder or one of its parent folders contains the following activated artifacts:
   - .xsapp file
     Application descriptor file required by an SAP HANA XS application
   - .xsaccess file
     Application access file which enables access to an SAP HANA XS application

**Duplicate Entries When Inserting Test Data**

If you encounter problems concerning duplicate entries when running tests, try the following solutions:
1. When inserting records into a productive table, ensure that no jasmine.dbConnection.commit() call occurs during test execution.
2. When inserting records into a test table, ensure that the table entries are deleted (dropped) before they are (re)created.

```javascript
var tableUtils = new TableUtils(jasmine.dbConnection);
tableUtils.clearTableInUserSchema(invoicesTestTable);
```

**Test Table Already Exists**

You encounter an error message that explains that a test table cannot be created during the test because the table already exists. You must ensure that the specified table is deleted before the test tries to create it during the test run.

```javascript
var sqlExecutor = new SqlExecutor(jasmine.dbConnection);
var createTableString = 'CREATE COLUMN TABLE ' + <table name> + '...' ;
sqlExecutor.execSingleIgnoreFailing('drop table ' + <table name> );
sqlExecutor.execSingle(createTableString );
```
You can also use the functions provided by the table utilities library, which enables you to ensure that the table is dropped at the right time:

```javascript
var tableUtils = new TableUtils(jasmine.dbConnection);
testTable = tableUtils.copyIntoUserSchema(originSchema, originTable);
```

### Test Model Activation Fails

Your test produces an error relating to a failed activation:

```javascript
Error: Repository: Activation failed for at least one object [...]
identifier is too long:[...]
Maximum length is 127: ...
```

the name of the model is too long (including the package name). You can reduce the name by setting the `TruncOptions` option as shown in the following code snippet:

```javascript
var mockstar = $.sap.hana.testtools.mockstar;
testView = mockstar.apiFacade.createTestModel(originalModel,
targetPackage, dependencySubstitutions, mockstar.TruncOptions.FULL);
```

➤ Tip

It's a good idea to analyze the created model before it is activated.

To generate a detailed and structured error log, in the SAP HANA Systems view in the SAP HANA studio and locate the test package and activate it manually.

### No Entries Returned From Copied Test Model

1. Open the generated test model.
   The generated model is located in a package with the name `tmpunittest.<userName>.<originalPackage>`. If you have configured the `createTestModel()` function with the parameter `mockstar.TruncOptions.FULL`, the package name is `tmpunittest.<userName>`.
2. Ensure that the dependencies have been replaced as expected.
   To see if the tables are filled correctly by the test, see No Test Data Inserted into Test Table [page 121].
3. Check the test view itself.
   If the tested view returns no data, but data are expected, check if the data are removed by a filter during extraction from the underlying data source.

### No Test Data Inserted into Test Table

To test whether a test inserts data as expected into the created test table, implement a `jasmine.dbConnection.commit()` connection to ensure that the data created during the test is stored...
persistently. Without the jasmine.dbConnection.commit(), the test data is not persistent; the test deletes all test data when the database session is closed. Start the test again using the TestRunner tool. When the test completes, the test table should contain test data.

TestRunner Tool Times Out

The default timeout setting for the TestRunner tool is ten (10) minutes. If your test run for longer than ten minutes and cause a timeout, try splitting the test into smaller and shorter elements. If this is no possible, try running the test in three phases:

1. Prepare the test run.
   /sap/hana/testtools/unit/jasminexs/PrepareTestRun.xsjs
   This generates a new test-run ID; no test runs are executed.
   ○ Response:
     Returns the new test-run ID. If you request the answer in HTML format and provide all required parameters for the TestRunner tool, you receive the appropriate links you can use in the following steps (run the test and fetch the results).
   ○ Parameters:
     format (optional; default = "html")
     Set this parameter to receive the test-run ID in the desired format. You can use any of the formats supported by the TestRunner format parameter.

2. Run the tests.
   /sap/hana/testtools/unit/jasminexs/TestRunner.xsjs
   This step is almost identical to the usual test execution with the addition of parameter runid.
   ○ Response:
     If the tests finish within the configured time frame, you receive the test results as expected. If the test are too long,a timeout occurs.
   ○ Parameters:
     runid. Required for this kind of (manual) execution: This is the test-run ID generated in the previous step.

3. Fetch the test results (optional: only required if the test run causes a timeout).
   /sap/hana/testtools/unit/jasminexs/GetTestResults.xsjs
   Fetches the test results for a given test-run ID. You an run this service multiple times for each test.
   ○ Response:
     Returns the test results in the requested format. If the tests are not yet finished, you receive a status message (either "PREPARED" or "STARTED"). If the run ID provided does not exist, an error message is displayed.
   ○ Parameters:
     runid. Required for this kind of (manual) execution.
     format (optional; default = "html")
Test Model Creation is Aborted

This error sometimes occurs if you try to create a copy of the original view and replace some dependencies with test tables. The reason for the error is one of the following:

- You did not provide any dependency substitutions. For example, you passed an empty array as the third parameter of `mockstar.createTestModel()`.
- The view that you want to test does not depend on any of the original views specified in the dependency substitutions.
- For active schema mapping, you have written the dependencies with the `physical` schema whereas the view refers to the `authoring` schema. Provide the schema in the same way as it is written in the view (or stored procedure).

Database Connections in XSUnit Test

The XSUnit test framework provides a new “managed” database connection called `jasmine.dbConnection`, which is automatically opened and rolled back (and closed) after each test completes. You can use it in `beforeEach` or `afterEach` functions, in other functions defined in your test libraries, or even in imported libraries, in the event that you have moved test code into external libraries.

Related Information

Managed Database Connection Setup [page 111]

2.10.4 Testing JavaScript with XSUnit

Test an XS JavaScript using XSUnit test tools.

As the XSUnit test tools are based on a custom version of the JavaScript test framework Jasmine, you can use XSUnit to test JavaScript. XSUnit provides tools that enable you to create and install a test “double” for one or more object methods. In the Jasmine framework, a test double is known as a “spy”. A spy can be used not only to stub any function but also to track calls to it and all arguments, too.

Note

XSUnit includes special `matchers` that enable interaction with Jasmine spies.

The XSUnit test tools delivery unit (DU) includes a small XS JavaScript demo “Ratings” application which comprises an SAPUI5 client frontend on top of OData and XS JavaScript services; the Ratings application enables you to experiment with different test techniques. You can try out the application at the following URL:

```
http://<SAPHANA_host>:80<instancenumber>/sap/hana/testtools/demo/apps/rating/WebContent/
```
Related Information

XSUnit's Jasmine Spy Syntax [page 124]
Testing HTTP Services with XSUnit [page 125]

2.10.4.1 XSUnit's Jasmine Spy Syntax

A command “cheat sheet” for the Jasmine Spy syntax.

The following code example provides a quick overview of commonly used commands that enable the use of Jasmine Spies. You can see how to perform the following actions:

- Install a method double [page 124]
- Install an object double [page 124]
- Check a function call (and values) [page 125]

Installing a Method Double

The following code example shows how install a method double (simple example).

```javascript
spyOn(object, "method");
expect(object.method).toHaveBeenCalled();
```

The following code example shows how install a method double (variant).

```javascript
var spyMethod = spyOn(object, "method");
expect(spyMethod).toHaveBeenCalled();
```

The following code example shows how install a method double (custom action for double).

```javascript
spyOn(object, "method"); // delegates nowhere
spyOn(object, "method").and.returnValue(3); // returns constant value
spyOn(object, "method").and.callThrough(); // delegates to original function
spyOn(object, "method").and.callFake(fakeFunction); // delegates to other function
```

Installing an Object Double

The following code example shows how install an object double.

```javascript
var spyObject = jasmine.createSpyObj("spy name", [ "method1", "method2", "method3" ]); 
spyObject.method1.and.returnValue(3);
expect(spyObject.method1).toHaveBeenCalledWith();
```
Checking Function Calls (and Values)

The following code example shows how to check whether the function has been called as expected, and if so, if the the right values were used.

```javascript
expect(spyObject.method).toHaveBeenCalled();
expect(spyObject.method).toHaveBeenCalledWith(expArgValue1, expArgValue2);
expect(spyObject.method.calls.allArgs()).toContain([expArgValue1, expArgValue2]);
expect(spyObject.method.calls.mostRecent().args).toEqual([expArgVal1, expArgVal2]);
spyObject.method.calls.reset(); // reset all calls
```

2.10.4.2 Testing HTTP Services with XSUnit

XS JavaScript files that can be accessed by performing an HTTP call against the service defined in the XS JavaScript file.

You can use the TestRunner tool to call an XS JavaScript service. The TestRunner service is part of the test-tools package sap.hana.testtools.unit.jasminexs and has one mandatory parameter, namely package. Since TestRunner is an HTTP GET service, you can execute the service in the browser using the following URL:

```
http://<hostname>:80<instancenumber>/sap/hana/testtools/unit/jasminexs/
TestRunner.xsjs?package=<mypackage>
```

Since it is not possible to import XS Javascript files (.xsjs) files into a JavaScript library (.xsjslib), the functions you implement inside the XS JavaScript file cannot be tested within an XSUnit test. As a consequence, it is recommended to include only minimal logic within the XSJS files and delegate tasks to the functions implemented in corresponding JavaScript libraries; these libraries can be tested in isolation using XSUnit tools (for example, Mockstar).

Note

XSUnit enables you to perform an HTTP call to your XSJS services via HTTP. However, this is an end-to-end system test with no possibility to use test doubles during the test. These tests are not suitable for testing a JavaScript function.

Since you cannot insert test data into the test table during the test, the tests have no control over the data. This restriction reduces the scope of the tests you can perform for HTTP calls, for example, you can test the following scenarios:

- Service must return an error if mandatory parameters are missing
- Service must return an error if the chosen HTTP type is correct
- Service must return an error if the wrong input is provided
- End-to-end HTTP scenarios (CREATE, READ, UPDATE, and DELETE)

```javascript
describe("example for http tests", function() {
    it("should receive answer from service", function() {
        var requestBody = '{"param1":42,"param2":"xyz"}';
```
var headers = {
"Content-Type" : "application/json"
};

var response = jasmine.callHTTPService("/path/to/your/app/Service.xsjs",
$.net.http.POST, requestBody, headers);
expect(response.status).toBe($.net.http.OK);
var body = response.body ? response.body.asString() : "";
expect(body).toMatch(/regular expression that checks correct response/));

SAP HANA Database Logon for XSUnit

To ensure access to SAP HANA, you need to adapt the default HTTP destination file
(:localhost.xshttpdest) provided with the XSUnit test tools. The default HTTP destination configuration
file is located in sap.hana.testtools.unit.jasminexs.lib:localhost.xshttpdest to fit to your
HANA instance. To access an HTTP destination configuration, you need the permissions granted in the user role sap.hana.xs.admin.roles::HTTPDestAdministrator.

⚠️ Caution

To change the HTTP destination, create an HTTP extension* of your own; do not make any changes to the
file localhost.xshttpdest. Changes to localhost.xshttpdest are overwritten by updates to the
XSUnit test tools on your system.

2.10.4.3 Testing JavaScript Functions with XSUnit

Use XSUnit tools to test JavaScript code that depends on functions in your code, for example: dependencies
on functions, libraries, or to database tables.

In JavaScript it is possible to overwrite anything that is visible in a context, for example: public data, public
functions, or even the whole class. With XSUnit, you can make use of a simulation framework that is included
with Jasmine. The simulation framework provides a mechanism that enables you to create and install a test
double (so-called Jasmine “Spy”), which can help you to reduce some of the basic code and keep the code
more concise. Jasmine Spies should be created in the test setup, before you define any expectations. The
Spies can then be checked, using the standard Jasmine expectation syntax. You can check if a Spy is called (or
not) and find out what (if any) parameters were used in the call. Spies are removed at the end of every test
specification.

⚠️ Note

Each dependency increases the complexity of testing involved for a function or a component.

The Average Component Dependency (ACD) is the number of dependencies to other components, averaged
over all components; it indicates whether your system is loosely coupled. If you prefer to implement
JavaScript in an object-oriented way, you can apply dependency management aspects by following object-
oriented design principles (OOD).
The information in this topic covers the following test scenarios:

- Dependencies on Function Libraries [page 127]
- Dependency on Database Table [page 128]

**Dependencies on Function Libraries**

The following code snippet defines a controller that you want to test; the controller depends on a `Date` object. The accompanying code snippet shows how you can test this code.

```javascript
var Controller = null;
(function() {
  //constructor function
  Controller = function(dataModel) {
    this.model = dataModel;
  };
  function updateModelWithTimestamp(newData) {
    this.model.updateData(newData, this.getCurrentDate());
  }
  Controller.prototype.updateModel = function(newData) {
    //bind 'this' to the private function
    updateModelWithTimestamp.call(this, newData);
  };
  Controller.prototype.getCurrentDate = function() {
    return new Date(Date.now());
  };
})();

function DataModel() {
  var modifiedAt = null;
  var modifiedBy = null;
  var data = null;
  this.updateData = function(newData, modifiedAtDate) {
    data = newData;
    modifiedAt = modifiedAtDate;
    modifiedBy = $.session.getUsername();
  };
  this.getModificationDate = function() {
    return modifiedAt;
  };
}
```

The following code snippets shows an example of the test code you could run; the code uses a Jasmine Spy ensures the dependencies on the `Date` object are replaced and tested as expected.

```javascript
var Controller = $.import("sap.hana.testtools.demo.objects.xs_javascript", "javascript00").Controller;
var DataModel= $.import("sap.hana.testtools.demo.objects.xs_javascript", "javascript00").DataModel;
describe('Controller', function() {
  var controller = null;
  var model = null;
  var anyDate = new Date(2013, 8, 27, 11, 0, 0, 0);

  beforeEach(function() {
    model = new DataModel();
    controller = new Controller(model);
  });

  it('should set current date when data is modified (replace Date.now() using jasmine spies)', function() {
    spyOn(Date, 'now').and.returnValue(anyDate.getTime());
    controller.updateModel({data : [1,2,3]});
  });
});
```
Dependency on Database Table

It is important to try to avoid mixing business logic that is implemented in JavaScript with the data base interaction. We recommend moving the database persistency logic into a dedicated persistency class, so that just the business logic remains for testing. The goal of the test is to be able to test both normal and special cases without interacting with the data base at all.

To unit test the persistency class, you can parameterize the schema and use a schema for testing, for example, the user schema where you have all authorizations required to create, modify, and drop objects, and cannot mess things up with the test. Last of all, you can offer a small set of integration tests, that just ensure that the productive classes, the AnyService class, and the Persistency class, integrate well.

Note

For sake of conciseness, resource closing and error handling is missing from the following code example.

```javascript
function Persistency(dbConnection, schema) {
    var dbSchema = schema !== undefined ? schema : 'SAP_HANA_TEST_DEMO';
    this.existsEntry = function(key) {
        var pstmt = dbConnection.prepareStatement('SELECT key FROM "' + dbSchema + '"."Table" WHERE KEY=?');
        pstmt.setString(1, key);
        if (pstmt.executeQuery().next()) {
            return true;
        }
        return false;
    };
    this.insertEntry = function(newEntry) {
        var pstmt = dbConnection.prepareStatement('INSERT INTO "' + dbSchema + '"."Table" VALUES(?,?)');
        pstmt.setString(1, newEntry.Id);
        pstmt.setString(2, newEntry.Value);
        pstmt.execute();
    };
}
function AnyService(persistency) {
    this.execute = function(input) {
        //validate input
        if (!persistency.existsEntry(input.Id)) {
            //calculate newEntry
            persistency.insertEntry(newEntry);
        }
    };
}
```

The following code snippets shows an example of the test code you could run to test the dependencies.

```javascript
var Persistency = $.import("package.of.persistency", "persistency").Persistency;
describe('Persistency test', function() {
    var SqlExecutor = $.import('sap.hana.testtools.unit.util', 'sqlExecutor').SqlExecutor;
    var TableUtils = $.import('sap.hana.testtools.unit.util', 'tableUtils').TableUtils;
    var originTable = 'TableName';
```
Testing a Self-Contained JavaScript Function

The following code snippet show how to use XSUnit to test a self-contained JavaScript function (mathlib); a self-contained function has no dependencies to other JavaScript functions, database tables or session parameters.

```javascript
var mathlib = $.import("package.of.your.library", "math");
describe('The math XS JavaScript library', function() {
  it('should calculate "7" as maximum value of "3, 7"', function() {
    var maxValue = mathlib.max(3, 7);
    expect(maxValue).toBe(7);
  });

  it('should calculate "-10" as maximum value of "-10, -20"', function() {
    var maxValue = mathlib.max(-10, -20);
    expect(maxValue).toBe(-10);
  });
});
```
Important Disclaimer for Features in SAP HANA Platform, Options and Capabilities

SAP HANA server software and tools can be used for several SAP HANA platform and options scenarios as well as the respective capabilities used in these scenarios. The availability of these is based on the available SAP HANA licenses and the SAP HANA landscape, including the type and version of the back-end systems the SAP HANA administration and development tools are connected to. There are several types of licenses available for SAP HANA. Depending on your SAP HANA installation license type, some of the features and tools described in the SAP HANA platform documentation may only be available in the SAP HANA options and capabilities, which may be released independently of an SAP HANA Platform Support Package Stack (SPS). Although various features included in SAP HANA options and capabilities are cited in the SAP HANA platform documentation, each SAP HANA edition governs the options and capabilities available. Based on this, customers do not necessarily have the right to use features included in SAP HANA options and capabilities. For customers to whom these license restrictions apply, the use of features included in SAP HANA options and capabilities in a production system requires purchasing the corresponding software license(s) from SAP. The documentation for the SAP HANA optional components is available in SAP Help Portal at http://help.sap.com/hana_options. If you have additional questions about what your particular license provides, or wish to discuss licensing features available in SAP HANA options, please contact your SAP account team representative.
Important Disclaimers and Legal Information

Coding Samples

Any software coding and/or code lines / strings ("Code") included in this documentation are only examples and are not intended to be used in a productive system environment. The Code is only intended to better explain and visualize the syntax and phrasing rules of certain coding. SAP does not warrant the correctness and completeness of the Code given herein, and SAP shall not be liable for errors or damages caused by the usage of the Code, unless damages were caused by SAP intentionally or by SAP’s gross negligence.

Accessibility

The information contained in the SAP documentation represents SAP’s current view of accessibility criteria as of the date of publication; it is in no way intended to be a binding guideline on how to ensure accessibility of software products. SAP in particular disclaims any liability in relation to this document. This disclaimer, however, does not apply in cases of willful misconduct or gross negligence of SAP. Furthermore, this document does not result in any direct or indirect contractual obligations of SAP.

Gender-Neutral Language

As far as possible, SAP documentation is gender neutral. Depending on the context, the reader is addressed directly with "you", or a gender-neutral noun (such as "sales person" or "working days") is used. If when referring to members of both sexes, however, the third-person singular cannot be avoided or a gender-neutral noun does not exist, SAP reserves the right to use the masculine form of the noun and pronoun. This is to ensure that the documentation remains comprehensible.

Internet Hyperlinks

The SAP documentation may contain hyperlinks to the Internet. These hyperlinks are intended to serve as a hint about where to find related information. SAP does not warrant the availability and correctness of this related information or the ability of this information to serve a particular purpose. SAP shall not be liable for any damages caused by the use of related information unless damages have been caused by SAP’s gross negligence or willful misconduct. All links are categorized for transparency (see: http://help.sap.com/disclaimer).