PUBLIC

SAP HANA Platform 2.0 SPS 00
Document Version: 1.0 – 2016-11-30

SAP HANA XS Advanced Migration Guide

Content

1 Introduction

Migrate an SAP HANA XS Classic application to SAP HANA XS Advanced platform instead of developing a new application.

SAP HANA extended application services, advanced model, (XS advanced) provide a comprehensive platform for the development and execution of native data-intensive applications that run efficiently in SAP HANA, taking advantage of its in-memory architecture and parallel execution capabilities. Structured accordingly, applications can profit from the increased performance provided by SAP HANA due to the integration with the data source. XS advanced is a polyglot application platform that supports several programming languages and execution environments, for example, Java and Node.js. The classic XS JavaScript (XSJS) is supported by a framework running in the Node.js run time.

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

To migrate an XS classic application to XS advanced using the XS Advanced Migration Assistant, you perform the following high-level steps:

1. Prepare for migration any XS classic artifacts that cannot be automatically migrated (for example, attribute and analytic views).

   > **Note**
   >
   > To complete the tasks described in this step, you need access to tools that are only available in the XS classic SAP HANA studio.

2. Run the Migration Assistant.
3. Read the report generated by the Migration Assistant and fix all the problems listed in the report.
4. Upload the successfully migrated application to XS advanced and use the provided XS advanced tools to build and refine the application in the new run-time environment.
5. Deploy and test the migrated application to the XS advanced run-time environment.

> **Restriction**
>
> The target XS advanced system where you plan to deploy the migrated application must be running HANA 2.0. You cannot use the XS Migration Assistant to help you migrate your XS classic application to an SAP HANA 1.0 system, for example, SPS 11 or SPS 12.

Related Information

Migrating an XS Application [page 6]

Comparing XS Classic to XS Advanced [page 21]
SAP HANA Developer Guide for XS Advanced

2 Migrating an XS Application

Use the XS Advanced Migration Assistant to migrate an XS classic application to the XS advanced run-time environment.

The following table lists the main steps required to complete the migration of you XS classic application to the XS advanced run time:

Table 1: Application Migration Steps

Steps	Action	Comments
1	Prepare for the migration operation	It is necessary to convert manually all deprecated object types used in the migrated XS classic application to the new XS advanced object types, for example: scripted calculation views, attribute views, Application Function Library (AFL) models, and Decision Tables
2	Prepare the SAP HANA source systems and required users	The migration requires the two SAP HANA systems: source (XS classic) and target (XS advanced). In this step you configure the source system (where the legacy XS classic application artifacts are located). **Note** If your source system is older than SAP HANA SPS 11, you also need an *external parse system*. Your target XS advanced system can be configured to play the role of external parser.
3	Migrate the XS classic application	Install and run the migration assistant Start the migration process **Restriction** The target XS advanced system where you plan to deploy the migrated application must be running HANA 2.0. You cannot use the XS Migration Assistant to help you migrate your XS classic application to an SAP HANA 1.0 system, for example, SPS 11 or SPS 12.

Steps	Action	Comments
4	Locate and read the report generated by the migration assistant.	The report generated by the migration assistant lists the problems found during the migration process; the problems must be fixed before running the migration again, for example: • Deprecated attribute views • Incorrectly formatted calculation views • Security artifacts (`.hdbroles` and `.xspriviliges`) • Database artifacts (`.hdbtable`, `.hdbview`, etc.) • XS JavaScript artifacts (`.xsjs`/`.xsjslib`)
5	Deploy the migrated application to XS advanced	Upload the successfully migrated application to XS advanced Build the new XS advanced application Set up the target system (including any required user-provided services) Deploy the migrated application to the XS advanced run time

Related Information

Prepare XS Classic Artifacts for Migration [page 7]
Prepare the Source System for the XS Application Migration [page 9]
Migrate Your XS Classic Application to XS Advanced [page 12]
Read the Migration Report [page 16]
Deploy the Migrated Application to XS Advanced [page 18]

2.1 Prepare XS Classic Artifacts for Migration

Manually convert any XS classic artifacts that cannot be migrated automatically by the XS Advanced Migration Assistant.

Prerequisites

This step is required if your XS classic application uses any of the following XS classic artifacts:

- Scripted (text-based) calculation views (`.calculationview`)
- Attribute views (`.attributeview`)
- Analytic views (`.analyticview`)

- Application Function Library (AFL) models (`.aflpmml`)
- Decision tables (`.hdbruldec`)
- XML-based analytic privileges (`.analyticprivilege`)

> **Tip**
>
> To help you migrate these modeling artifacts, use the view-migration tool in the *SAP HANA Modeler* perspective in SAP HANA studio; the migration tools is available at *SAP H ANA Modeler* > *Migrate*.

Context

Some XS classic artifacts cannot be migrated automatically by the XS Migration Assistant tool; these artifacts must be manually prepared for migration using XS classic tools, for example, the *SAP HANA Modeler* perspective in SAP HANA studio.

Scripted calculation views, attribute views, or analytic views must be converted to **graphical** calculation views before you start the application migration process. The same applies to any other objects that are no longer supported (deprecated), for example, Application Function Library (AFL) models (`.aflpmml`) and Decision Tables (`.hdbruldec`). The conversion steps are described in the *SAP HANA Administration Guide for SAP HANA Studio* (AFL) and the *SAP HANA Modeling Guide for SAP HANA Studio* (calculation views and decision tables).

> **Note**
>
> If you run the Migration Assistant without first manually converting any deprecated artifact types, the resulting report will most likely not be very usable; it will contain a long list of problems to fix. For this reason, make sure you perform the manual steps listed here **before** running the Migration Assistant.
>
> For more information about **how** to migrate the XS classic objects, see *Related Links*.

Procedure

1. Manually convert to a graphical calculation view any XS classic scripted (text-based) calculation views.

   The XS Advanced Migration Assistant cannot automatically convert scripted **text-based** calculation-view artifacts (`.calculationview`) to the format required by XS advanced; you must use the tools included in SAP HANA studio to convert the scripted calculation view to a **graphical** calculation view (`.calculationview`) and a set of table functions by hand **before** you start the XS Advanced Migration Assistant.
2. Convert to a graphical calculation view any XS classic **attribute** views.

   The XS advanced Migration Assistant cannot automatically convert the XS classic attribute view artifact (`.attributeview`); you must use the tools included in SAP HANA studio to convert the attribute view to a **graphical** calculation view (`.calculationview`) by hand **before** you start the migration assistant.
3. Convert to a graphical calculation view any XS classic **analytic** views.

The XS advanced Migration Assistant cannot automatically convert the XS classic analytic view artifact (`.attributeview`); you must use the tools included in SAP HANA studio to convert the analytic view to a **graphical** calculation view (`.calculationview`) by hand **before** you start the migration assistant.

4. Convert any XS classic Application Function Library (AFL) models.

   The XS advanced Migration Assistant cannot automatically convert the XS classic AFL model artifact (`.aflpmml`) to the format required by XS advanced; you must use the tools included in SAP HANA studio to convert the AFL model by hand **before** you start the migration assistant.

5. Convert any decision tables.

   The XS advanced Migration Assistant cannot automatically convert XS classic decision tables to the format required by XS advanced; you must use the tools included in SAP HANA studio to convert the decision tables (`.hdbruldec`) by hand **before** you start the migration assistant.

6. Convert any XML-based analytic privileges to SQL analytic privileges.

   The XS advanced Migration Assistant cannot automatically convert XS classic XML-based analytic privileges (`.analyticprivilege`) to the format required by XS advanced; you must use the tools included in SAP HANA studio to convert the XML-based analytic privileges by hand **before** you start the migration assistant. For more information about **how** to perform this manual migration, see the *SAP HANA Modeler Guide for SAP HANA Studio* in the *Related Links*.

Related Information

SAP HANA Administration Guide
SAP HANA Modeling Guide for SAP HANA Studio (Migrating an Object Type)
SAP Note 2325817 (Migrating Analytic/Attribute Views to Calculation Views)
Security Models: XS Classic & XS Advanced [page 25]

2.2 Prepare the Source System for the XS Application Migration

Set up the XS classic and XS advanced systems to be used for the XS application migration.

Prerequisites

Before preparing source and target systems for using the XS Advanced Migration Assistant, bear in mind the following points:

- You must already have converted manually any XS classic artifacts that cannot be migrated automatically by the XS Advanced Migration Assistant.

Context

To set up the source and target systems for the XS Advanced Migration Assistant, perform the following steps:

Procedure

1. Prepare the source SAP HANA systems you plan to use for the migration operation.

   The following alternatives are supported:

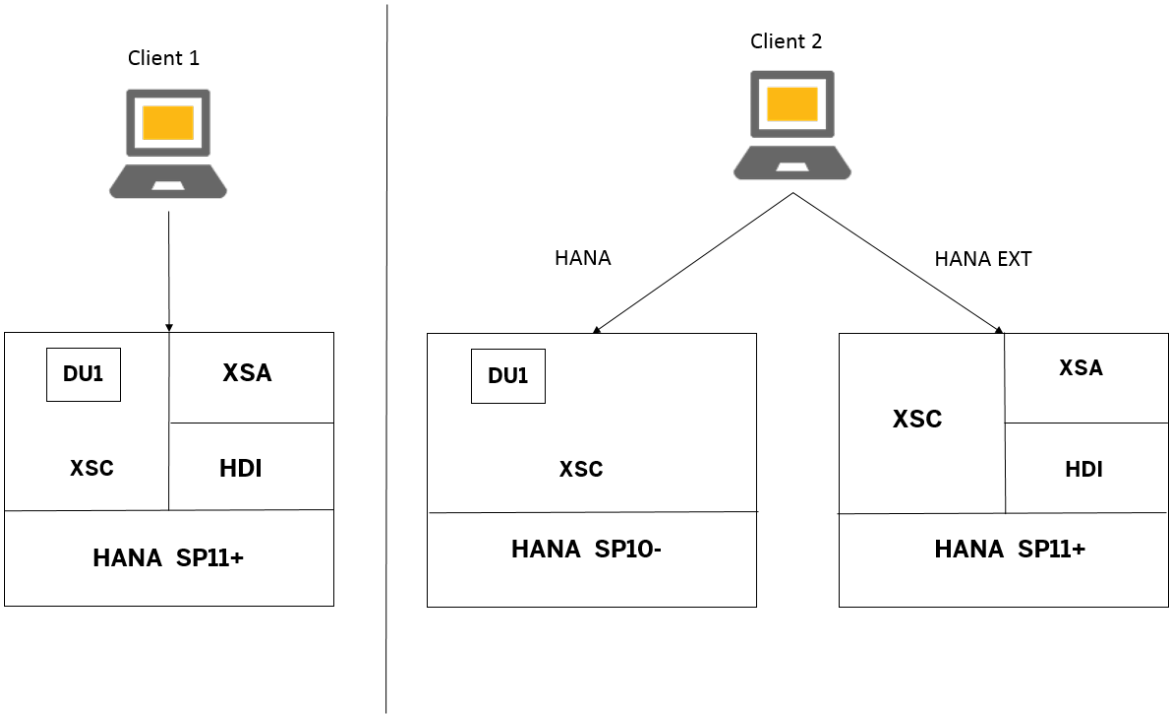
   - XS Classic System (SAP HANA SPS 09 or higher)
     If the source system (XS classic) is running SAP HANA SPS10 or lower, it does not support the procedure `SYS.GET_OBJECTS_IN_DDL_STATEMENT`. In that case, a SAP HANA SPS11 (or higher) system must be used as the external parse system.

     > **Note**
     >
     > Typically the XS Advanced target system plays this role.

   - XS Classic System (SAP HANA SPS 11)
     If the XS classic source system where your application project Delivery Units (DU) are installed is already on SAP HANA SPS 11 (or later), you do not need an additional system for parsing.

   > **Note**
   >
   > Since the XS Advanced Migration Assistant does not modify any application content or design time objects during the migration process, you can use any test or development XS classic system for the migration.

The following diagram shows the different setups supported by the XS Advanced Migration Assistant and the components required:



2. Configure the SAP HANA users required for the migration operation and assign them the necessary permissions.

   In order to read the design-time objects of your application from the system where your XS classic project is installed, a user is required on the **XS classic system** with the following XS classic and SAP HANA roles and privileges:

   - `sap.hana.xs.lm.roles::Administrator` role
   - `_SYS_REPO.ACTIVE_CONTENT_TEXT` object privilege
   - `_SYS_REPO.ACTIVE_CONTENT_TEXT_CONTENT` object privilege
   - `_SYS_REPO.ACTIVE_OBJECT_TEXT` object privilege
   - `_SYS_REPO.ACTIVE_OBJECT_TEXT_CONTENT` object privilege
   - `_SYS_BI.M_SCHEMA_MAPPING` object privilege

   If you use an additional system for XS advanced, you must also make sure that you have a database user on the XS advanced system that has the following roles and permissions assigned:

   - `sap.hana.xs.lm.roles::Administrator` role
   - The user needs to have `EXECUTE` permission on the `SYS.GET_OBJECT_DEFINITION` object.
     To do this, run the following SQL statement in the SQL console:
     `GRANT EXECUTE ON "SYS"."GET_OBJECT_DEFINITION" TO "<<user>>"`
   - The user needs to have `EXECUTE` permission on the `SYS.GET_OBJECTS_IN_DDL_STATEMENT` object.
     To do this, run the following SQL statement in the SQL console:`GRANT EXECUTE ON "SYS"."GET_OBJECTS_IN_DDL_STATEMENT" TO "<<user>>"`

- For security reasons, `GET_OBJECTS_IN_DDL_STATEMENT` is disabled by default; you must enable it, for example, with the following SQL statement:

```
ALTER SYSTEM ALTER CONFIGURATION ('indexserver.ini', 'system') SET
('sqlscript', 'enable_builtin_procedure_get_objects_in_ddl_statement') =
'true' WITH RECONFIGURE
```

- Alternatively, you can use *SAP HANA Administration Console* in SAP HANA Studio. Open the *Configuration* dialog in *indexserver.ini* > *sqlscript* and add a new system parameter with the "key" "`enable_builtin_procedure_get_objects_in_ddl_statement`" and the value "`true`".

Related Information

Prepare XS Classic Artifacts for Migration [page 7]
Migrate Your XS Classic Application to XS Advanced [page 12]
Migrating an XS Application [page 6]

2.3 Migrate Your XS Classic Application to XS Advanced

Run the XS Migration Assistant to convert your XS classic application to XS advanced.

Prerequisites

Before migrating your XS classic application to XS advanced using the XS Migration Assistant, bear in mind the following points:

- You must already have converted manually any XS classic artifacts that cannot be migrated automatically by the XS Migration Assistant.
- You must already have set up the source (XS classic) system for the migration
- You have set up the required users on the SAP HANA systems and assigned them the required access permissions.
- The XS Migration Assistant makes use of the SAP Java Virtual Machine (JVM), so make sure the correct C ++ library is installed on Windows machines, for example, the x64 variant of the Microsoft C++ run time. For more information, see *Related Links*.

> ⚠ Restriction
>
> The target XS advanced system where you plan to deploy the migrated application must be running HANA 2.0. You cannot use the XS Migration Assistant to help you migrate your XS classic application to an SAP HANA 1.0 system, for example, SPS 11 or SPS 12.

Context

To migrate an XS classic application to XS advanced using the XS Migration Assistant, perform the following steps:

Procedure

1. Install the Migration Software.
   a. Download the latest XS Migration Assistant package.
   b. Unpack the ZIP archive to any directory.

   > **Note**
   >
   > The name of the download directory must not contain any spaces and, to avoid problems with long file names in Windows, it must be close to the root directory (for example, `C:\`.

2. Configure the connection to the XS classic and XS advanced systems.

   > **Tip**
   >
   > You can define the connection settings in two ways: on the command line or using a `.env` file.

   You need to provide environment settings for the XS classic system where your XS classic application is running.

   a. Define environment settings for the XS classic system where the application project Delivery Units (DU) are installed.

      For Windows systems:

      - `set HANA_HOST=<HANA hostname>`
      - `set HANA_PORT=<XS-Engine port>` (for example, 8000)
      - `set HANA_SQL_PORT=<HANA SQL port>` (for example, 30015
      - `set HANA_USER=<HANA user>`
      - `set HANA_PASSWD=<HANA password>`
      - `set HANA_PROTOCOL=<[http | https]>`
      - `set HANA_CERTIFICATE=</path/to/HTTPS/certificate/file>`

      > **Note**
      >
      > For Linux System, use the **export** command instead of **set**.

   b. Configure environment settings for external parsing (optional).

      If the source database does not support the procedure `SYS.GET_OBJECTS_IN_DDL_STATEMENT`, an external parse system can be specified with the following environment variables

      For Windows systems:

      - `set HANAEXT_HOST=<XSA hostname>`

- `set HANAEXT_PORT=<XS-Engine port>` (for example, 8000)
- `set HANAEXT_SQL_PORT=<HANA SQL port>` (for example, 30015)
- `set HANAEXT_USER=<HANA username>`
- `set HANAEXT_PASSWD=<HANA password>`
- `set HANAEXT_PROTOCOL=<[http | https]>`
- `set HANAEXT_CERTIFICATE=</path/to/HTTPS/certificate/file>`

> **Note**
>
> For Linux System, use **export** instead of **set**.

c. Configure environment settings for "dotenv" support.

> **Note**
>
> This step is optional; if you define the settings in the `.env` file, you do not need to set the environment variables on the command line as specified in the previous steps.

The XS Migration Assistant supports `dotenv`, a Node.js function that enables you to load environment variables defined in a `.env` file. The XS Migration Assistant supports the following parameters:

- `[HANA | HANAEXT]_HOST=<HANA hostname>`
- `[HANA | HANAEXT]_PORT=<XS-Engine port>` (for example, 8000
- `[HANA | HANAEXT]_SQL_PORT=<HANA SQL port>` (for example, 30015)
- `[HANA | HANAEXT]_USER=<HANA username>`
- `[HANA | HANAEXT]_PASSWD=<HANA password>`
- `[HANA | HANAEXT]_PROTOCOL=<[HTTP | HTTPS]>`
- `[HANA | HANAEXT]_CERTIFICATE=</path/to/HTTPS/certificate.file>`

> **Tip**
>
> For more information about "dotenv" (.ENV), see *Related Links*.

3. Run the XS Migration Assistant.

The following simple example shows how to use the XS Migration Assistant:

> **Sample Code**

```
xs-migration  --target-dir /root/tmp/test_migration/result DU1,sap.com
```

The command shown in the example above takes the content from `DU1` from vendor "sap.com", generates a template of an XS advanced folder structure, and places the migrated objects in the folder `/root/tmp/test_migration/result/xsa-app`. The migration assistant also generates a report with a list of the problems detected; you must read the report and solve any problems before you can run the migration assistant again and use the application in XS advanced.

a. In the command shell (or at the "system" level), set the path to the root folder of the XS Migration Assistant (`xs-migration`).

The root folder contains the `package.json` and the `xs-migration` executable (`xs-migration.bat` for Windows or `xs-migration` for Linux).

b. Run the `xs-migration` or `xs-migration.bat` command with the desired options.

The `xs-migration` tool supports the following options:

- `<DU names>[,<vendor name>]`
  For example, `DU1,acme.com DU2,acme.com` or `DU1 DU2`.

  > **Note**
  > More than one Delivery Unit (DU) can be specified in a space-separated list; no comma is required to separate the DU names.

- `[--name <your project name>]`
  The name of the XS classic application project to migrate to XS advanced; the default is the name of the specified DUt
- `[--version <project version>]`
  The version of the XS classic application project to migrate to XS advanced; ; the default is the version of the specified DU.
- `--target-dir </path/to/generated/migration/archive>`
  The root directory where the result of the XS application migration is located. The target directory must not exist.
- `--zip`
  Add the migration result to a Zip-compressed folder for easy import into SAP WebIDE for SAP HANA
- `--generate-manifests`
  The XS Migration Assistant generates the manifest files (`manifest.yml` and `manifest-op.yml`) required to push the migrated application modules to the XS advanced run time. With these manifest files, you can use the XS command-line client to "`push`" individual modules of the generated MTA project to the XS advanced run-time environment.
- `--exclude-packages <[pkg],...>`
  A list of the packages to exclude from the XS migration, where `<[pkg],...>` is `<package-name>[:exclude-subpackages]` and "`exclude-subpackages`" is either "`true`" or "`false`" (default is "`false`", for example:

  ```
  --exclude-packages com.sap.db:true, <package-name>
  ```

- If you run the `xs-migration` or `xs-migration.bat` commands without any options, the migration does not start; instead, it displays details of the options supported by the migration tool.

Related Information

Prepare the Source System for the XS Application Migration [page 9]
Read the Migration Report [page 16]
Migrating an XS Application [page 6]
SAP JVM requires Microsoft Visual C++ 2010 runtime
dotenv (.ENV)

2.4 Read the Migration Report

The XS Advanced Migration Assistant generates a report, which lists details of the migration operation including problems that need to be fixed.

Prerequisites

Before reading the report generated by the XS Advanced Migration Assistant, bear in mind the following points:

- You must already have converted manually any XS classic artifacts that cannot be migrated automatically by the XS Advanced Migration Assistant.
- You must already have set up the source (XS classic) system for the migration

  > **Note**
  >
  > This does not have to be two separate systems; it could be the same SAP HANA system.

- You have set up the required users on the SAP HANA systems and assigned them the required access permissions.
- You have already installed, configured, and run the XS Advanced Migration Assistant.

Context

The migration report is created in the folder specified in the options included when you run the Migration Assistant (`xs-migration` or (`xs-migration.bat`). The report is in the form of an HTML file, and includes a migration summary and a list of any problems that occurred during the migration. The report summary includes details such as file statistics, a list of the migrated files, and so on; the migration summary contains a list of errors, warnings, and recommendations, which is split into convenient categories, for example:

- Preparation
- Security
- Unsupported Features
- Database Artifacts
- XS JavaScript Migration
- Translation
- Unknown File Types or Content
- Objects not Migrated

The objects from the provided delivery units are exported from the source XS classic system, analyzed, migrated, and then written into the folder structure required for an XS Advanced application. The format of the information in the report matches the phases the migration process goes through.

To understand and respond to the information and suggestions included in the report generated by the XS Advanced Migration Assistant, proceed as follows:

Procedure

1. Look at the information included in the migration-report section “Preparation Phase”.

   Before commencing with the migration of the application you must manually migrate the artifacts listed in the “Preparation Phase” section, for example, using the tools provided in SAP HANA Studio. This applies to scripted calculation views, attribute views, analytic views, application privileges, and files generated by the Application Function Modeler (`.aflpmml`) files. After you have completed the manual migration of the listed artifacts, run the XS Migration Assistant again.

   The Errors contains list of details of objects that must be migrated manually to the SAP HANA Studio.
2. Look at the information included in the migration-report section “Migration of Security Concept Required”.

   Since the XSA security concept is incompatible with that of XS Classic, manual migration steps are required in order to complete the migration of an XS classic application's security artifacts to XS Advanced.

   The information in the “Warnings” section includes a list of the roles and access privileges that must be checked using `$.hdbgetConnections`.

   > **Note**
   >
   > Due to the change in security concepts between XS classic and XS advanced, not all the features configured in the XS classic application's XS-access file (`.xsaccess`) can be migrated to the XS advanced application router descriptor (`xs-app.json`).
3. Look at the information included in the migration-report section “Currently Unsupported or Discontinued Features”.

   This section contains a list of unsupported and discontinued objects, which the migration assistant cannot migrate automatically; you must fix the issues listed in this section by hand.
4. Look at the information included in the migration-report section “Database Artifacts”.

   This section of the migration report contains information about design-time database artifacts which created warnings during the migration process.

   > **Note**
   >
   > The technical HDI user in XS advanced must be granted permission for access to the SAP HANA database to be able to read and retrieve the XS classic database artifacts, for example, tables, views, procedures, and so on.
5. Look at the information included in the migration-report section “XSJS JavaScript Migration Required”.

   This section of the migration report contains a list of the statements in XS JavaScript code (for example, in the application's code `.xsjs` and libraries `.xsjslib` that cannot be automatically migrated; you must migrated this XS JavaScript code manually.

6. Look at the information included in the migration-report section “Translation”.

   This section of the migration report contains a list of the `.hdbtextbundle` files containing UI texts and possibly translations which were found as part of the migrated XS classic application.
7. Look at the information included in the report section “Unknown File Types or File Content”.

   This section of the migration report includes a list of the components that the XS Advanced Migration Assistant was not able to identify and specifies the file types. You must decide whether these objects are relevant, and copy them into the appropriate container directory if needed.
8. Look at the information included in the migration-report section “Objects Which Have Not Been Migrated”.

   This section of the migration report contains details of the objects which it was not possible to migrate, for example, because the objects are either not relevant or have been successfully migrated to another object type.

Related Information

Prepare the Source System for the XS Application Migration [page 9]
Migrate Your XS Classic Application to XS Advanced [page 12]
Deploy the Migrated Application to XS Advanced [page 18]

2.5 Deploy the Migrated Application to XS Advanced

Deploy the successfully migrated XS classic application to the XS advanced run-time environment.

Prerequisites

Before reading the report generated by the XS Advanced Migration Assistant, bear in mind the following points:

- You must already have converted manually any XS classic artifacts that cannot be migrated automatically by the XS Advanced Migration Assistant.
- You must already have set up the source (XS classic) system for the migration
- You have set up the required users on the SAP HANA systems and assigned them the required access permissions.
- You have already installed, configured, and run the XS Advanced Migration Assistant.
- You have run the migration assistant, fixed any flagged problems, and produced a successfully migrated package.
- You have already installed your target system with HANA 2.0

> ⚠ Restriction
>
> The target XS advanced system where you plan to deploy the migrated application must be running HANA 2.0. You cannot use the XS Migration Assistant to help you migrate your XS classic application to an SAP HANA 1.0 system, for example, SPS 11 or SPS 12.

- You have installed the latest versions of the following components on your target system:
  - XS advanced run-time environment
  - SAP Web IDE for SAP HANA

Context

To deploy your migrated application project to the XS advanced run time, perform the following steps:

Procedure

1. Create user provided services for `synonym-grantor-service` as expected by generated `mta.yaml` and `db/cfg/grantor.hdbgrants` files:

   ```
   xs cups synonym-grantor-service -p "{\"host\":\"<hana_host>\",\"port\":
   \"<sql_port>\",\"user\":\"<hana_user>\",\"password\":\"<hana_passwd>\", \"tags
   \":[\"hana\"]}"
   ```

   You need to provide the credentials of a SAP HANA user that can access the physical source schema as specified in the generated `db/cfg/grantor.hdbgrants` file. Note that these specified privileges must be given with the grant option, because during deployment HDI will generate technical users and uses the provided credentials to grant appropriate privileges to these users.
2. Compress the contents of the migration target folder into a ZIP archive.

   > ➔ Tip
   >
   > If you specified the --zip option when running the XS Advanced Migration Tool (`xs-migration`) you can skip this step.

3. Import the ZIP archive into SAP WebIDE for SAP HANA using the *Workspace* context menu.

   Remember to assign the correct space to the imported project, for example, in the project settings, as described in the *SAP HANA Developer Guide for SAP XS Advanced*.
4. Build the database component (HDB module) in the new XS advanced application's MTA project.

   In SAP Web IDE for SAP HANA, right-click the `hdb/` folder of your imported Multi-Target Application in the *Project* pane, and in the context menu, choose *build*.
5. Check the build errors and fix any problems.

   You can process the build errors provided in the console at the bottom of the SAP Web IDE for SAP HANA window or use the generated `report.html` for a list of open problems in the individual MTA modules.

Alternatively, you can browse through the generated HDI container with the SAP HANA Runtime Tools (*Tools* > *SAP HANA Database Explorer*).

Related Information

Prepare XS Classic Artifacts for Migration [page 7]
Prepare the Source System for the XS Application Migration [page 9]
Migrate Your XS Classic Application to XS Advanced [page 12]
Read the Migration Report [page 16]

3 Comparing XS Classic to XS Advanced

List and compare the differences between the underlying concepts of XS classic and XS advanced.

To decide if it makes sense to migrate your XS classic application to XS advanced, you need to review the design of your XS classic application and compare it with the design of the XS advanced platform. The alternative is to rewrite the application from scratch taking advantage of all the new features provided by the XS advanced run-time environment.

The migration tool is able to convert HDB-related objects from the XS classic repository to an HDI-compatible format that is suitable for XS advanced. You can reuse the result even if you decide to develop from scratch. We recommend that you run the XS advanced Migration Assistant once and read the report to see what, if any, problems need to be addressed to complete the migration successfully. The information in this topic aims to help you understand the conceptual difference between XS advanced and XS classic.

Security in XS Classic and XS Advanced

The security concept underpinning XS advanced is different to the one used in XS classic. However, even though this increases the effort for you to migrate the application from XS classic to XS advanced, you can still use the XS advanced Migration Assistant to help migrate your application.

If you decide to migrate your XS classic application to XS advanced, bear in mind that you must adapt it to the XS advanced security concepts discussed in this section, which includes information about the following areas:.

- New Security Concepts in XS Advanced [page 21]
- Controlling Business Users' Access to Database Content [page 22]
- Migrating Security Components with the XS Advanced Migration Assistant [page 22]

New Security Concepts in XS Advanced

Business users are no longer bound to the `.hdbrole` role-definition file. For more information, see *Related Links* below.

In XS classic, business users were also database users, and it was possible to assign database privileges to each business user to protect database access. In XS advanced, business users are not linked to database access privileges. This means it is not possible to differentiate between users by assigning different privileges to database-related objects or schemas.

Authorization scopes are defined in the application layer.

The concept of the authorization “scope” in XS advanced is similar to the application-privileges descriptor (`.xsprivileges`) in XS classic. For more information, see *Related Links* below.

The XS advanced security migration requires a scope check on each XS JavaScript database request. In XS advanced, scopes are assigned to roles, and roles are assigned to role collections. Role collections can be assigned to business users.

The HDI container can get business user attributes from the request session by using `SESSION_CONTEXT('XS_<attribute name>')..`

This provides additional information to differentiate business user groups, for example,. `select * from TB1 where SESSION_CONTEXT('XS_COSTCENTER') = "010234"`.

Attributes should be requested by adding them to the role template in the XS advanced security descriptor (`xs-security.json`) in the JavaScript (`xsjs/`) container.

All request connections between the JavaScript (`xsjs/`) and database `db/` containers are triggered with the same technical user.

The technical user can have multiple `hdbroles` assigned for database access.

> **Tip**
>
> You can define a default `hdbrole` (called "`default_access_role`") for all required accesses in your XS advanced application. You can also prepare some `hdbroles` to provide limited access for foreign technical users from other applications. For more information about "`default_access_role`" for HDI containers, see the section describing the HDI Deployer in the *SAP HANA Developer Guide for XS Advanced* or *Related Links*.

Controlling Business Users' Access to Database Content

You must design (or reuse) the authorization scope and user roles defined in the application layer, and check on each database request.

As discussed in the security-migration topic, the migration tool generates a security "template" for you based on what security components you build into the XS classic application, for example, user roles (`.hdbrole` and application privileges (`.xsprivileges`)

You must review the security template generated by the XS Advanced Migration Assistant and carefully check if you have already defined enough authorization scopes and user roles to differentiate business users and protect database access in the JavaScript layer.

You must implement the attributes to differentiate user groups.

Migrating Security Components with the XS Advanced Migration Assistant

The migration tool detects `db.getConnection` or `hdb.getConnections` in your code to warn you that you need a scope check for database requests.

The migration tool generates a template `xs-security.json` file based on your `.hdbrole` and `.xsprivileges` design, which is a good point for you to start the security migration without testing the valid file format.

The migration tool generates a `.hdbrole` file for you called "default_access_role".

Container Isolation Concept

If your application has more than one Delivery Unit (DU), and the multiple DUs are working closely with each other, the information in this section is important.

XS advanced introduced the concept of containers, which includes a high level of isolation between the different application-development units. No easy access is allowed between containers that are not in the same application or under the same schema.

Possible solutions for cross-reference between containers include the following components:

- Public Synonyms
- Calls to the REST API
- Synonyms for Cross-HDI container data access
  Synonyms are required for foreign database container access. This means if the table design in your XS classic application follows the object-oriented discipline, you might need to develop your database structure from scratch with a different approach in the database module your XS advanced application.
- JavaScript cross-container access:
  - JavaScript containers in XS advanced cannot call the `.xsjs` files or `.xsjslib` libraries directly from another JavaScript container
  - Cross-container calls are possible with the REST HTTP, but the extra effort involved to set it up must be evaluated.

Run-time Authoring

- HDI provides APIs to generate new HDB objects during run time. DROP is strictly forbidden.
- The XS classic concepts of XS Job files (`.xsjob`) and secure-store definitions (`secureStore`) are still valid in XS advanced. You need to create and bind user-provided services in XS advanced to feed the data for the service.
- The JavaScript (`js/`) and Web (`web/`) containers do not allow the generation of new files at run time.
  If you have functions that generate XS JavaScript or UI code, and activate the code using XS classic repository APIs, bear in mind that this feature is not supported in XS advanced. You can choose not to use the code/function in your migrated XS advanced application or try to integrate it into the XS advanced application, for example, using the tools provides with SAP Web IDE for SAP HANA.
  If neither suggestion is valid for your application, it is not possible to migrate your application to XS advanced.

Related Information

Programming Models: XS Classic & XS Advanced [page 24]
Security Models: XS Classic & XS Advanced [page 25]
Translation Texts in XS Classic & XS Advanced [page 30]
SAP HANA Developer Guide for XS Advanced (Security Artifacts)
SAP HANA Developer Guide for SAP HANA Studio (Application-Privileges File)
SAP HANA Security Guide (Application Privileges)

3.1 Programming Models: XS Classic & XS Advanced

SAP HANA Extended Application Services advanced model (XS advanced) adds an application platform to the SAP HANA in-memory database. In the Cloud, this platform is provided by Cloud Foundry

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

Run-Time Authoring or Offline Checkin and Checkout

To prepare for an application migration, it is important to understand the fundamental differences in the development model employed in SAP HANA XS classic and SAP HANA XS advanced model. Unlike in XS classic, where design-time artifacts are stored in the SAP HANA Repository, the code base for an XS advanced application is stored in an external Git repository. Applications are deployed ("pushed") from the Git repository to the target platform and run in separate processes that can be scaled independently. This means the whole development paradigm changes between XS Classic and XS Advanced from an ABAP-like "share-everything" approach to a more Java-like "share-nothing" approach.

In XS advanced, every project has its own Git repository. Every change from any developer will be treated in isolation as a separate branch in the corresponding project's Git repository, which can be deliberately merged into central 'master' to synchronize with other changes, provided that conflicts were resolved or not created. To qualify changes for merge to master, typically a continuous integration pipeline is built up, which runs automated regression tests on every change. At run time, HDI containers provide a similar isolation on database level. The concept of 'Spaces' in XS Advanced can be used to achieve the same isolation also for running applications. To summarize, instead of building up dedicated machines for development, test, consolidation, and production, XS Advanced allows to host every stage – even for multiple projects – on a single, sufficiently dimensioned machine.

Related Information

SAP HANA Developer Guide for XS Advanced

3.2 Security Models: XS Classic & XS Advanced

How you can migrate your XS classic application's security concepts to XS advanced.

The information in this topic explains how to build up the security concept in software development, in particular what you need to do to adapt the security-related objects used in your XS classic applications to suit the security framework used in XS Advanced.

> ⚠ Restriction
>
> In this topic, “Security” is limited to SAP HANA authorizations as they relate to software development and software migration. The information included here does not address the following areas: system-security administration, network-security configuration, detailed network communication of authorization methods, or storage-security.

XS Classic Security-Relevant Objects

The information in this section describes the security-related XS classic objects and indicates which XS classic objects are supported in XS advanced. For more information about the individual objects described in this section, see *Related Links* below.

- Application Access
  The features configures in the XS classic application-access file (`.xsaccess`) are mostly controlled by the XS advanced application router (`approuter.js`), which is configured using the XS advanced application descriptor (`xs-app.json`).
  For more information about the XS classic application-descriptor (`.xsaccess`) and the XS advanced application descriptor (`xs-app.json`), see the respective developer guide as listed in *Related Links* below.
  - `"exposed": ["true | false"]`
    This XS classic feature is not supported in XS advanced. JavaScript (`xsjs/`) packages are “exposed” to client requests by default,. However, HDI packages are not exposed to client requests, for example, from a Web browser.
  - `"authentication":`
    The `authentication` property in XS classic supports a wide variety of authentication mechanisms. These authentication mechanisms have been moved to the SAP HANA XS User Account and Authentication service (UAA), and currently there are only two options for XS advanced: “`route`” and “`none`”. The migration tool automatically sets the authentication method to “`route`”. Note that the UAA service must already be configured for the specified authentication mechanism.
  - `"anonymous_connection"` and `"default_connection"`
    These features are related to SQL connection configuration files (`.xssqlcc`) which are neither required nor used in XS advanced. However the sqlcc concept is still valid but implemented in different way, for example, by using user-defined services.
    The example below shows how to control access to an XS advanced application in the application (router) descriptor `xs-app.json`:

    ```
    {
    ```

```
    "routes": [
        {
            "source": "^/sap/ui5/1(.*)$",
            "target": "$1",
            "destination": "ui5",
            "scope": ["viewer", "manager"]
        }
    ]
}
```

- The `.xsprivileges` and `.hdbrole` files:
  Developers use the `.xsprivilege` files to define their application privileges. Application privileges are assigned to roles which can then be bound to users. The same concept exists in XS Advanced. Developers can define roles and scopes in `xs-security.json` and bind them to users via groups. Scopes are almost the same as XS classic application privileges, and roles have a similar concept as XS classic `.hdbroles`, but both are only effective in the `xsjs` container and the app-router. All the HDI-related privileges stay with the `.hdbrole` files as HDI container objects. But the `.hdbrole` files are not designed for user access control, but for binding services access control. To make it short, the role-based privilege concept stays the same in the application layer (`node/xsjs/java` container and `app-router`) but differs in HDI containers and there are many required manual changes for this concept update. The detail security concepts will be described below in the *SAP HANA XS advanced Security Concept* section and how to update your code in the *Migration Plan* section.
- Analytic Privilege:
  There are two parts to this concept, `.analyticprivilege` files and the Data Control Language (DCL) in CDS.
  - DCL was introduced with HANA XS SP9. The XS Advanced Migration Assistant will use the CDS migration plug-in to convert the grammar of CDS artifacts including DCL parts.
  - The `.analyticprivilege` files are converted by the XS Advanced Migration Assistant in a two-step approach
    1. Migrate XML-based analytic privileges to SQL analytic privileges manually using tools in SAP HANA studio
    2. Migrate the SQL analytic privileges using the XS Advanced Migration Assistant to an XS advanced-compatible format.

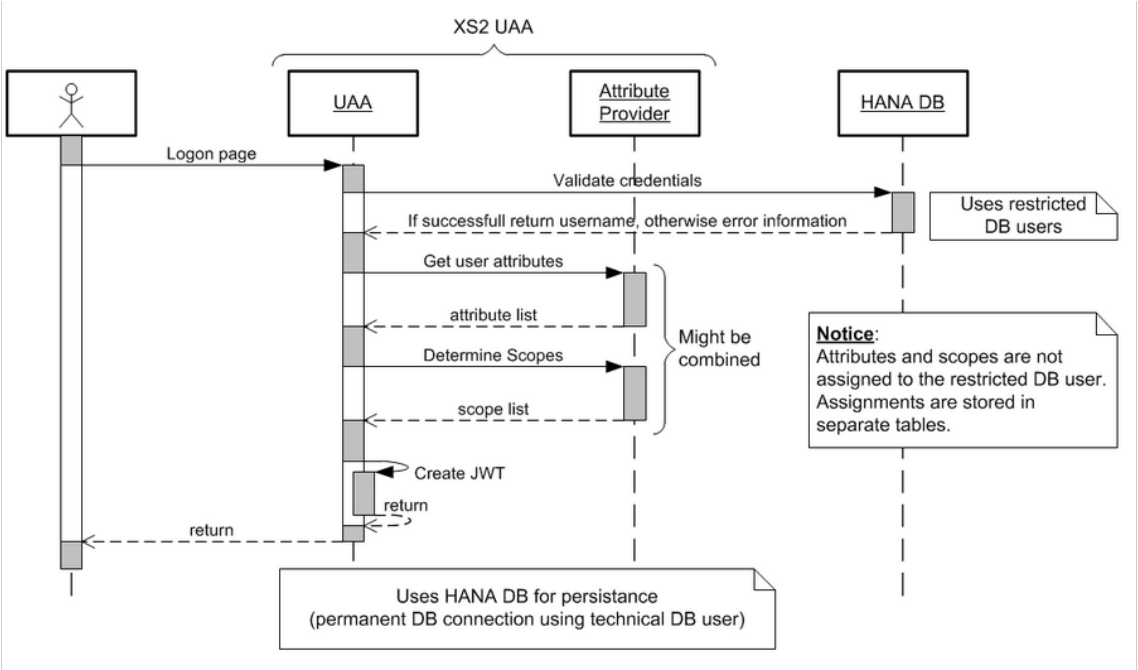
  Since `.analyticprivilege` files do not directly refer to any table or static value (for example, `SESSION_USER`) apart from `$$client$$` and `$$language$$`, the converted result is usually still valid. It can only be functional if the related objects (calculation views and `hdbprocedures`) are also correctly adjusted.
  Since users in XS advanced are no longer SAP HANA database users, an attribute concept was introduced in XS advanced. Attributes are stored in identity providers (IDPs) and it is recommended that developers request attributes that will be stored in a user session. The request is also defined in the XS advanced security descriptor (`xs-security.json`.

SAP HANA XS Advanced Security Concept

Business User Authentication method

Based on the network security configurations, you can define the business users' logon method either as `xsuaa` or `basic` in XS advanced.

For the standard UAA SAML authentication work flow, refer to the chart below:



When performing an authentication other than SAML2, the UAA only performs the required HTTP communication (for example, by providing a FORM-based password logon page, triggers the SPNego challenge, etc.). The received credentials are not validated by the UAA but they are passed to the HANA database for validation. Attributes cannot be extracted from the credentials but will be retrieved from a local persistence. For more information, see *Related Links* below.

Application Privileges in XS Advanced

In the user management, you define your business users and assign permissions to users by adding groups or role collections. A group or a role collection contains roles that are defined in the application (`xs-security.json`). And each role contains scopes, which are the privileges that you can check in the `xsjs/node/java` etc. coding and protect your exposed URL access.

Check the following example of a `xs-security.json` file.

```
{
    "xsappname": "sample-leave-request-app",
    "scopes":
    [
        {
            "name": "$XSAPPNAME.createLR",
            "description": "create leave requests"
        },
        {
            "name": "$XSAPPNAME.approveLR",
            "description": "approve leave requests"
            "granted-apps": ["MobileApprovals"]
        }
    ],
    "attributes":
    [
        {
            "name": "costcenter",
            "description": "costcenter",
            "valueType": "s"
```

```
        }
    ],
    "role-templates":
    [
        {
            "name": "employee",
            "description": "Role for creating leave requests",
            "scope-references":
[ "$XSAPPNAME.createLR","JobScheduler.scheduleJobs" ],
            "attribute-references": [ "costcenter"]
        },
        {
            "name": "manager",
            "description": "Role for creating and approving leave requests",
            "scope-references":
[ "$XSAPPNAME.createLR","$XSAPPNAME.approveLR","JobScheduler.scheduleJobs" ],
            "attribute-references": [ "costcenter" ]
        }
    ]
}
```

User-related check in XS Advanced's Deployment Infrastructure (HDI)

In XSA, business users are not bound to any HDI privileges or permissions. There used to be `.hdbrole` files in XS classic to bind HDB privileges to a business user role and further to a business user. This will not work in XS advanced. All users use the same service to connect to HDI using the same technical user. This requires the following manual actions on development side:

- Review HDI objects:
  - If you still need user attributes in your objects, region, cost-center, etc. and the user attributes are not maintained in the HDI container of your application, you need to request the user attributes in `xs-security.json` and list the proper attributes under different role definitions.
  - Remove unsupported referenced objects and find a workaround.
  - Replace `SESSION_USER` with `$USER` as now, in HDI, `SESSION_USER` points to a technical user.
- Change code in application layer:
  - You must use `$.session.hasAppPrivilege()` or `$.session.assertAppPrivilege()` before sending a request to HDI.
  - Check the `odata` calls which go directly from browser via app-router to HDI, by adding routes with proper privileges. This is necessary because the `odata` calls are not going through your application code in `xsjs/node.js/java`.
  - Review your privilege design to find out if it fits the new access control concept.

Other System Access Posibilities

- `.xssqlcc`
  Since the `.xssqlcc` file is no longer supported a similar concept is used to support `sqlcc`.
  Developers must create their own *user-provided-service instance* to feed the information required for `sqlcc`.
- HTTP destination
  Similar to the `.xssqlcc` solution, developers must create their own *user-provided-service instance* to feed the information required for the HTTP destination.

Migration Plan

Bear in mind the following points:

1. Application Layer - Define Scopes and Role Templates
   What the Migration Assistant does:
   - It converts the XS classic `.hdbrole` names to the new role templates for XS advanced.
   - It converted the XS classic application “privileges” to authorization scopes in XS advanced.
   - It assigned the application privileges to the user roles according to your XS classic application.

   Items that must be reviewed before migration:
   - Do the old application privileges and scopes you used in XS classic still the design you want to use in your Cloud-compliant XS advance application ?
   - For any application privileges defined for external applications still available or usable?

   Items that must be migrated manually:
   - Add the requested attributes to `xs-security.json` and properly link them to roles.
   - Add routes with scope checks to `xs-app.json` for all `xsodata`-like objects (objects that go through the application router to the HDI, without the possibility to check this in `xsjs` code)
2. Application Layer – Check Scopes in xsjs Coding
   What the migration tool did:
   - It reports some statements that you need to check in xsjs coding (`$.session.hasApplicationPrivilege`)
   - It reports some HDB connections that you can use for further review
   - It reports some statements that are no longer supported and require a new design (`$.repo.XXX`)

   What you must do:
   - You must use `$.session.hasAppPrivilege()` or `$.session.assertAppPrivilege()` before sending requests to HDI.
     If you use a different code flavor, there are similar functions. Check them in detail. Example code above is only valid for the XS advanced xsjs coding.
   - Prepare the user-provided service to feed your `sqlcc` or HTTP destination calls. The `xsjs` code mostly remains the same.
3. Update Objects and References in HDI Objects
   After defining the attributes in the role templates, the attributes are accessible within CDS by using `SESSION_CONTEXT('XS_<PROP>`.
   Check the content of the HDI objects and make sure that all the used objects are still valid in the XS advanced platform. Some objects that are no longer supported are already reported in the migration tool, in the HDI sections.
4. HDI – Understand and Update SAP HANA Database Roles (hdbroles)
   Check whether the generated technical user `.hdbrole` file contains all required permissions and whether all the objects used are still valid in the XS advanced platform.
   Create other `.hdbrole` files to enable foreign application access with the proper permissions defined.

Related Information

https://help.sap.com/hana/SAP_HANA_Developer_Guide_for_SAP_HANA_Studio_en.pdf
SAP HANA Developer Guide for XS Advanced

http://help.sap.com/saphelp_hanaplatform/helpdata/en/c3/d9889e3c9843bdb834e9eb56f1b041/content.htm?current_toc=/en/de/ec02ebbb57101483bdf3194c301d2e/plain.htm&show_children=true
Cloud Foundry (User-Provided Services)

3.3 Translation Texts in XS Classic & XS Advanced

Understand the differences in the way translation texts are handled in XS classic and XS advanced.

This section explains the how translation text stored in specially formatted (and named) files is handled in XS classic and XS advanced. The information in this section is split into the following sections:

- Translation Texts in XS Classic [page 30]
- Translation Texts in XS Advanced [page 31]

Translation Texts in XS Classic Applications

The XS classic Repository for design-time artifacts uses so-called `.hdbtextbundle` files to define any text that needs to be translated, for example, the names of fields and tab titles in the user interface.
These `.hdbtextbundle` files are similar both in content and format to `.properties` files and contain placeholders and metadata that is used to describe the lengths and types of string marked for translation and, in addition, an initial translation (often referred to as "developer text"). Like all other design-time artifacts, `hdbtextbundle` files are transported between XS classic systems by means of a Delivery Unit (DU). The translation texts themselves are transported separately using a different format coming from a central translation system.

Persistence

The content is persisted in central database tables located in the `_SYS_REPO` schema:

- `ACTIVE_CONTENT_TEXT`
- `ACTIVE_CONTENT_TEXT_CONTENT`
- `ACTIVE_OBJECT_TEXT`
- `ACTIVE_OBJECT_TEXT_CONTENT`

Usage

The translated texts can be used on different levels of an applications. Database objects can use them by joining the respective tables while XSJS and UI can access the `hdbtextbundle` files via HTTP. In this case a special functionality compiles the `hdbtextbundle` files and exchanges the translation according to the one defined in the current user session. If no translation is available, the file will be delivered with the developer texts defined in the original design-time file.

Translation Texts in XS Advanced Applications

Since XS advanced makes a clear separation between the different application layers, translation entities must be defined on the same layer as they are used. Although the format and content of the files that contain the text marked for translation in XS classic and XS advanced is the same, the suffix for files containing translation texts in XS advanced "`.properties`"; in XS classic the file suffix for translatable content was "`.hdbtextbundle`".

Persistence

The type of persistence depends on the application layer where the translation entity is used. For UI and application logic (independent of the specific run time), the `.properties` files are stored unchanged alongside the other source files.

For database usage, HDI defines a special header for the `.properties` files, which informs the HDI deployment logic to transfer the file content to a defined table. If the database artifacts using the translation texts are calculation views, the target table is central and the access is transparent to the user (`_SYS_BI.BIMC_DESCRIPTIONS`). For other database artifacts, the target table needs to be passed to HDI as a design-time artifact. The table holding the translation texts will then be container-local.

Sample Code

```
#!tabledata
#{
#  "target_table"   : "TEXT_TABLE",
#  "column_mappings" : {
#    "OBJECT"   : { "type"  : "constant",
#                   "value" : "name.space::VIEW" },
#    "LANGUAGE" : { "type"  : "function",
#                   "name"  : "extractLanguageCodeFromFileName",
#                   "parameters": { "file_name": "OBJECT_en_US.properties"}
#                 },
#    "KEY"      : 1,
#    "VALUE"    : 2
#  }
#}
# Note: The table data header ends on the first empty line or earlier on the
#       second occurrence of the !tabledata marker
# Note: Format of .properties entries:
#
#   # <metadata for translation process>
#   <key>=<value>
#
```

Usage

Depending on the layer, the usage differs. Database artifacts can use the translation texts by joining the container-local translation table. For UI and application logic, it is necessary to handle the selection of the appropriate `.properties` file individually. In contrast to the XS classic Repository, there is no longer any need for any compilation logic.

Migration

The information in this section explains the naming conventions used for translation entities in XS advanced and how this is handled when translation text bundles are migrated from XS classic to XS advanced. The name and suffix of files containing translation text in XS advanced use the following format::

Sample Code

```
<object-name>_<language_extension>.properties
```

The `<language_extension>` part of the file name is either `_<lang>` (for example, "`_de_DE`") for the translated texts or "empty" for the developer language. It is important to understand that `.hdbtextbundle` files are transformed to `.properties` files without a language extension (to represent the developer language). The corresponding translated texts are extracted from the `_SYS_REPO` database tables and written to `.properties` files with the respective appropriate language extension.

The same rules apply for calculation views, where an empty language column in the database table identifies the developer language.

The `.properties` files for the calculation view are prefixed with the specified header and moved to the `db/src/` folder of the generated Multi-Target Application (MTA) project in XS advanced. The `.properties` files succeeding `.hdbtextbundle` files are moved to a separate folder, and a corresponding message is added to the migration report. This is necessary, because the XS Advanced Migration Assistant cannot determine whether the files are used in database artifacts, in the application logic, or on the client side.

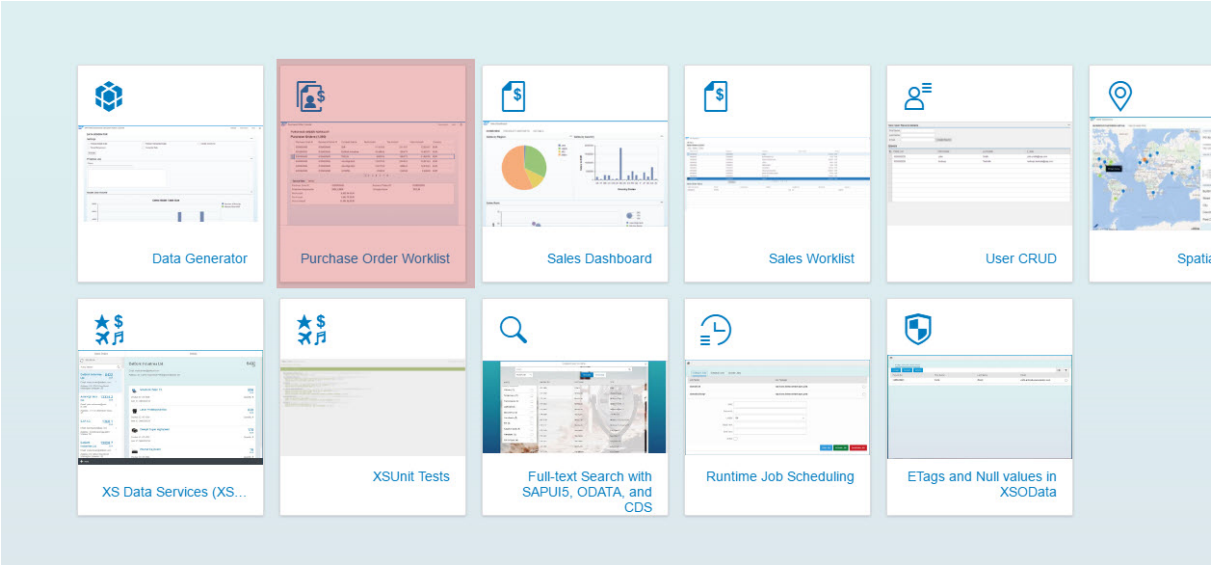
Related Information

Programming Models: XS Classic & XS Advanced [page 24]
Security Models: XS Classic & XS Advanced [page 25]
Migrating an XS Application [page 6]

4 XS Advanced Migration Demo - SHINE

A real-life demonstration of the XS Migration Assistant using the SAP HANA Interactive Education (SHINE) demo application.

The information in this section shows how to use the XS Migration Assistant to migrate a real application, the SAP HANA Interactive Education (SHINE) demo application from XS classic to XS advanced. To demonstrate how to migrate as many of the most common differences between XS classic and XS advanced as possible, the migration described in this section does not convert all elements of the XS classic SHINE application; instead, it focuses primarily on migrating the *Purchase Order Worklist* application included in the XS classic SHINE example, as illustrated in the following example.



> **Note**
>
> Although an XS advanced version of SHINE is already available, in order to illustrate and explain the typical challenges you are likely to encounter when migrating XS classic applications, this section deliberately starts the migration of the original XS classic version of SHINE from scratch. It is important to understand that SHINE represents the "worst case" migration scenario; its stated aim is to demonstrate almost every existing feature of XS Classic, and, due to conceptual changes, some of these features cannot be migrated without manual effort.

Related Information

Migrating an XS Application [page 6]
http://help.sap.com/hana/SAP_HANA_Interactive_Education_SHINE_en.pdf

4.1 Prepare the XS Classic Source System

Import the SHINE Delivery Unit (DU) into the source system and prepare all necessary files for migration.

Prerequisites

Before preparing source and target systems for the migration with the XS Advanced Migration Assistant, bear in mind the following points:

- You must already have converted manually any XS classic artifacts that cannot be migrated automatically by the XS Advanced Migration Assistant, for example, analytic privileges or decision tables, and so on.

Context

To set up the source and target systems for the XS Advanced Migration Assistant, perform the following steps:

Procedure

1. Download the XS classic SHINE DU.

   The SHINE documentation on the SAP Help portal explains where to find the SHINE DU. However, you can also export the SHINE DU from an existing SAP HANA XS classic development system and import it into an SPS 11 system. For more information, see *Related Links*

   In order to read the design-time objects included in the SHINE application, a user with the Adminstrator role *sap.hana.xs.lm.roles::Administrator* and a `$HANA_USER` used by `xs-migration` to access the migration source system.
2. Migrate files in the source system.

   The SHINE application contains the decision tables and analytic privileges listed below; these objects must be converted to graphical calculation views **before** you start the migration to XS advanced. For more information, see *Related Links*.

   Decision tables:
   - `sap.hana.democontent.epm.models::DT_CUSTOMER_DISCOUNT_CALCULATION.hdbruldec`
   - `sap.hana.democontent.epm.models::DT_TAX_CALCULATION.hdbruldec`

   Analytic privileges:
   - `sap.hana.democontent.epm.models::AP_PURCHASE_ORDER_PROD_CAT.analyticprivilege`
   - `sap.hana.democontent.epm.models::AP_PURCHASE_ORDER.analyticprivilege`
   - `sap.hana.democontent.epm.models::AP_SALES_ORDER.analyticprivilege`

Related Information

Prepare XS Classic Artifacts for Migration [page 7]
SAP HANA Interactive Education (SHINE)

4.2 Run Migration and Evaluate Migration Report

Run the XS Advanced Migration Assistant, evaluate the generated migration report, and edit the application sources.

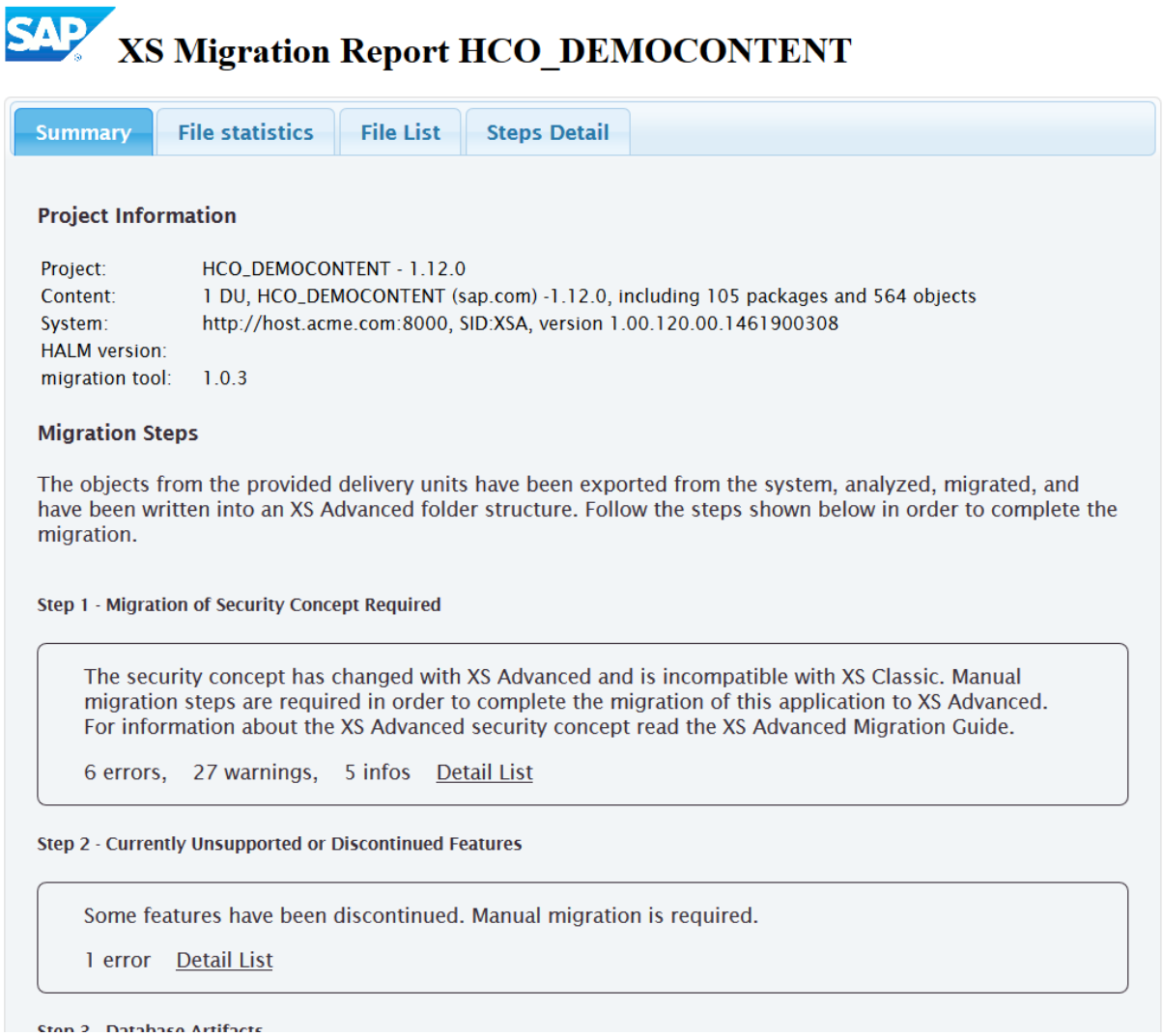
Prerequisites

Before running the migration for SHINE, you must perform the following steps:

- Prepare the XS classic source system
- Manually convert any unsupported artifacts, for example: analytic, attribute, and scripted calculation views.

Context

The XS Advanced Migration Assistant generates a report that looks like the following example:



To migrate the SHINE application to XS advanced using the XS Migration Assistant, perform the following steps:

Procedure

1. Run the XS Advanced Migration assistant.
   a. To display the options permitted by the XS Advanced Migration Assistant, open a command shell, and call the `xs-migration` command without any parameters.
   b. The server-access data can be supplied by means of environment variable or as a list of variables defined in a `.env` file.

The command used to start the migration of the XS classic SHINE application should look like the following example:

```
xs-migration --target-dir /root/tmp/migration-results/shine HCO_DEMOCONTENT
```

> **Note**
>
> If no target directory is specified, a “results” folder is created in the installation directory of the migration assistant.

2. Evaluate the migration report.

   The migration report lists all items migrated for the complete SHINE application; the action items are grouped according to technical area. In this example, action is required in the following areas:

   - Step 1: Migration of Security Concept Required
   - Step 4: XSJS JavaScript Migration Required
   - Step 5: Translation

   The follow-up action indicated in the migration report covers the following areas:

   a. Adapt security artifacts.

      The security section of the migration report lists the migrated security-related artifacts which require further attention.

   b. Adapt XS JavaScript artifacts.

      The following table lists the migrated artifacts of category `XSJS` which require further attention.

      Table 2: XSJS-related Issues in the Migration Report

Type	Details	Affected Files
ERROR	Use of `sap.hana.xs.libs.dbutils`	`xsjs/lib/sap/hana/democontent/epm/services/lazyPost.xsjslib` `xsjs/lib/sap/hana/democontent/epm/services/session.xsjslib`
ERROR	Use of `$.util.Zip();`	`xsjs/lib/sap/hana/democontent/epm/services/poWorklistQuery.xsjs`
WARNING	Database statements containing schema	`xsjs/lib/sap/hana/democontent/epm/services/poWorklistUpdate.xsjs` `xsjs/lib/sap/hana/democontent/epm/services/LoadCurr.xsjs`
WARNING	Use of `xsodata` with `xml` format	`web/resources/sap/hana/democontent/epm/ui/poworklist/view/Detail.controller.js`

   c. Adapt translation-related artifacts.

The following table lists the migrated artifacts of category `TRANSLATION` which require further attention.

Table 3: Translation-related Issues in the Migration Report

Type	Details	Affected Files

INFO	Use of `.hdbtextbundle`	`migration/todo/src/sap/hana/democontent/epm/ui/launchpad/i18n/messagebundle.properties`
		`migration/todo/src/sap/hana/democontent/epm/ui/poworklist/i18n/messagebundle.properties`
		`web/resources/sap/hana/democontent/epm/ui/NewLaunchpad.htm`
		`web/resources/sap/hana/democontent/epm/ui/poworklist/Component.js`

3. Edit the application sources.

   To make the necessary adaptations to the application, you can either do it in **either** of the following ways:

   - Directly on the generated migration result with tools of your choice; you must upload the application to and deploy the application in XS advanced.
   - Upload the migrated sources to SAP Web IDE for SAP HANA and make the necessary changes using the available Web-based editors. Then, you can deploy the adapted application from the Web IDE. For more information, see *Related Links*.

Related Information

Prepare XS Classic Artifacts for Migration [page 7]
Prepare the Source System for the XS Application Migration [page 9]
Migrate Your XS Classic Application to XS Advanced [page 12]
Read the Migration Report [page 16]

4.2.1 Concept Check

It is necessary to go through an evaluation of the XS advanced concepts to check whether the concept fits the design of your application.

After running the XS Migration Assistant, a migration folder is generated containing the migrated elements of your XS classic application.

Concept Check for Purchase Order Worklist

Although the *Purchase Order Worklist* application element does **not** need cross-schema access, other SHINE application elements do; it is important in the context of XS advanced. For this reason, it is recommend not to access any system objects, for example, tables, or views.

> **Note**
>
> For an application that will be migrated to a Cloud application, you will need to check again with the XS advanced experts and, where necessary, reconsider your solution for this part.

Related Information

XS Advanced Migration Demo - SHINE [page 33]
Migrating an XS Application [page 6]

4.3 Security-Related Concept Check

Check the authorization scope of the XS advanced user to ensure that all database calls are protected.

Context

The XS classic SHINE application contains two roles, "*User*" and "*Admin*". These XS classic roles contain database privileges that cannot be migrated to XS advanced automatically; the XS Advanced Migration Assistant only considers the application privileges. It is essential to ensure that the XS advanced scopes are sufficient to distinguish users.

Procedure

1. Check the authorization scopes defined in the generated XS advanced security descriptor `xs-security.json`:

   The migrated XS classic user privileges can be found as “scopes” in the generated XS advanced security descriptor `xs-security.json` as illustrated in the following example:

   > **Sample Code**
   >
   ```
   {
     "xsappname": "HCO_DEMOCONTENT",
   ```

```
  "scopes": [
    {
      "name": "$XSAPPNAME.sap.hana.democontent.epm.Basic",
      "description": "Basic usage privilege"
    },
    {
      "name": "$XSAPPNAME.sap.hana.democontent.epm.Admin",
      "description": "Administration privilege"
    }
  ],
  "role-templates": [
    {
      "name": "sap_hana_democontent_epm_roles_Admin",
      "scope-references": [
        "$XSAPPNAME.sap.hana.democontent.epm.Basic",
        "$XSAPPNAME.sap.hana.democontent.epm.Admin"
      ]
    },
    {
      "name": "sap_hana_democontent_epm_roles_User",
      "scope-references": [
        "$XSAPPNAME.sap.hana.democontent.epm.Basic"
      ]
    }
  ]
}
```

2. These scopes are then referred to in the `xs-app.json` file of the web module so that the URLs are protected:

   These scopes are then referred to in the `xs-app.json` file of the web module so that the URLs are protected:

Sample Code

```
{
  "welcomeFile": "index.html",
  "authenticationMethod": "route",
  "routes": [
    {
      "source": ".*xsjs.*",
      "destination": "xsjs"
    },
    {
      "source": ".*\\.xsodata.*",
      "destination": "xsjs"
    },
    {
        "source": "^/sap/hana/democontent/epm/(.*)$",
        "scope": [
            "$XSAPPNAME.sap.hana.democontent.epm.Basic"
        ],
        "localDir": "resources"
    },
    {
        "source": "^/sap/hana/democontent/epm/admin/(.*)$",
        "scope": [
            "$XSAPPNAME.sap.hana.democontent.epm.Admin"
        ],
        "localDir": "resources"
    }
  ]
}
```

3. Check and adapt any references to the old XS classic analytic privileges.

   The original XS classic SHINE application includes roles that contain references to analytic privileges. To ensure that the migrated application can run in XS advanced run-time environment, the XS Advanced Migration Assistant collects all XS classic privileges in the default XS Advanced role for the `HDI` container, with the result that access to the database is not protected. For productive applications, the roles and privileges must be adapted to meet the requirements of the XS advanced security concept. For more information, see *Related Links*.

Related Information

SAP HANA Developer Guide for SAP HANA XS Advanced Model
XS Advanced Migration Demo - SHINE [page 33]
Migrating an XS Application [page 6]

4.4 Create the Required XS Advanced Services and Feed Values

To build, deploy and run the migrated SHINE application in XS advanced, it is necessary to create some XS advanced services manually.

Context

The SAP Web IDE for SAP HANA automatically generates some XS advanced services. However, this automation does not include the User Account and Authorization (`UAA`) service or the synonym grantor service. You must create these services by hand, for example, using the XS CLI (`xs create-service` command).

Procedure

1. Create the UAA services.

   Make sure the generated `xs-security.json` file can be found; the file is required by the `xs` as illustrated in the following example:

   Sample Code

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

2. Create the synonym granter service and feed system and user

   In the following example, cups is an abbreviation of the `create-user-provided-service`.

   Sample Code

   ```
   xs cups synonym-grantor-service -p "{\"host\",
   \"port\",\"user\",\"password\",\"driver\",\"tags\"}"
   ```

Related Information

Migrating an XS Application [page 6]
XS Advanced Migration Demo - SHINE [page 33]

4.5 Adjust the Generated Template

When using the SAP Web IDE for SAP HANA for the application deployment, there is no manual adaptation of the generated templates that are required to get the *Purchase Order Worklist* application running.

Context

The `xs-security.json` security descriptor contains some references to the Fiori Launchpad example. Since the Fiori Launchpad example is not supported in XS advanced, these references have to be deleted.

Procedure

Check the contents of the security descriptor (`./xs-security.json`) for references to "`FioriLaunchPad`", as illustrated in the following example:

Sample Code

```
"$FOREIGNXSAPPNAME.sap.hana.democontent.epm.ui.uis.FioriLaunchPad.WidgetAccess:
FioriShineCatalog",
```

Related Information

Migrating an XS Application [page 6]
XS Advanced Migration Demo - SHINE [page 33]

4.6 Making Required Code Updates

Make any updates to the code for the XS advanced application which are listed in the migration report.

The information in this section describes the code updates that are required to ensure that the *Purchase Order Worklist* application can run in the XS advanced run-time environment. The code updates reflect the instructions listed in the report generated by the XS Advanced Migration Assistant. It is necessary to read the migration report and check the content in each section (reflecting the XSA containers) and make the required changes. The following containers are covered by the information in this section:

- The Database (DB) container
- The JavaScript (XSJS/Node.js) container
- The Web (UI) container

Related Information

The Database (DB) Container [page 43]
The JavaScript (XSJS) Container [page 45]
The Web Container [page 48]

4.6.1 The Database (DB) Container

The migrated SHINE application requires additional privileges for HDI objects in the XS advanced database (DB) container.

Context

To permit the deployment of the database module to the XS advanced run-time environment, it is necessary to add the synonyms for currency conversion.

Procedure

1. Check the contents of the synonym configuration file `cfg/synonymconfig.hdbsynonymconfig`.

   Sample Code

   ```
   "TCURF": {
      "target": {
           "object": "sap.hana.democontent.epm.data::Conversions.TCURF"
      }
   },
   "TCURN": {
       "target": {
           "object": "sap.hana.democontent.epm.data::Conversions.TCURN"
       }
   },
   "TCURR": {
       "target": {
           "object": "sap.hana.democontent.epm.data::Conversions.TCURR"
       }
   },
   "TCURV": {
       "target": {
           "object": "sap.hana.democontent.epm.data::Conversions.TCURV"
       }
   },
   "TCURX": {
       "target": {
           "object": "sap.hana.democontent.epm.data::Conversions.TCURX"
       }
   }
   ```

2. Check the contents of the synonyms definition file `src/synonyms.hdbsynonym`.

   Sample Code

   ```
   "TCURF": {}
   "TCURN": {}
   "TCURR": {}
   "TCURV": {}
   "TCURX": {}
   ```

Related Information

Migrating an XS Application [page 6]
XS Advanced Migration Demo - SHINE [page 33]

4.6.2 The JavaScript (XSJS) Container

The migrated SHINE application requires modifications to objects in the XS advanced database JavaScript container.

Prerequisites

Since it is not possible to guarantee that the XS Advanced Migration Assistant detects all problems in the XS classic XS Javascript `xsjs` code scan, you must review all issues that are listed in the XSJS section of the migration report gererated by the XS Advanced Migration Assistant, as shown in the following list:

Procedure

1. Check the JavaScript library `lazyPost.xsjslib`.
   a. Locate the code shown in the following example:

Sample Code

```
$.import("sap.hana.xs.libs.dbutils", "xsds");
var XSDS = $.sap.hana.xs.libs.dbutils.xsds;
var metadata = {
  "entityName":"sap.hana.democontent.epm.data::Util.Messages",
  "tableName":"\"SAP_HANA_DEMO\".
\"sap.hana.democontent.epm.data::Util.Messages\"",
  "schemaName":"SAP_HANA_DEMO",
  "revMapping":{
    "TEXT":"TEXT","DESCRIPTION":"DESCRIPTION","LANGUAGE":"LANGUAGE",
    "MESSAGENUMBER":"MESSAGENUMBER","MESSAGECLASS":"MESSAGECLASS"},
  "keyFields":{"LANGUAGE":{"$seq":true,"$type":11},"MESSAGENUMBER":
{"$seq":true,"$type":11},
  "MESSAGECLASS":{"$seq":true,"$type":11}},"secondaryIndexes":
[{"LANGUAGE":{"$seq":true,"$type":11},
  "MESSAGENUMBER":{"$seq":true,"$type":11},"MESSAGECLASS":
{"$seq":true,"$type":11}}],
  "sqlMetadata":{"TEXT":{"$type":26,"$size":0},"DESCRIPTION":{"$type":
11,"$size":0},
  "LANGUAGE":{"$type":11,"$size":0,"$key":true},"MESSAGENUMBER":{"$type":
11,"$size":0,"$key":true},
  "MESSAGECLASS":{"$type":11,"$size":0,"$key":true}},
  "cdsMetadata":
{},"isCdsEntity":true,"isUnmanaged":false,"isFlexible":false,"isAuto":fal
se,"entityId":1};
var mapping = {
  "TEXT":{"$column":"TEXT"},"DESCRIPTION":
{"$column":"DESCRIPTION"},"LANGUAGE":{"$column":"LANGUAGE","$key":true},
  "MESSAGENUMBER":{"$column":"MESSAGENUMBER","$key":true},"MESSAGECLASS":
{"$column":"MESSAGECLASS","$key":true}};
var entity =
XSDS.Entities.makeEntity('sap.hana.democontent.epm.data::Util.Messages',
'"SAP_HANA_DEMO"."sap.hana.democontent.epm.data::Util.Messages"',
mapping, metadata, { $noimport: true });
```

b. Replace it with the following code:

Sample Code

```
var conn = $.db.getConnection();
var XSDS = $.require("sap-cds").xsjs(conn); // "sap-cds" refers to node-
cds
var entity =
XSDS.importEntity('sap.hana.democontent.epm.data','Util.Messages');
```

2. Check the JavaScript library `session.xsjslib`.

a. Locate the code shown in the following example:

Sample Code

```
var XSProc = $.import("sap.hana.xs.libs.dbutils", "procedures");
var proc = XSProc.allProcedures("SAP_HANA_DEMO",
"sap.hana.democontent.epm.procedures");
```

b. Replace it with the following code:

Sample Code

```
var connection = $.hdb.getConnection();
var proc = {
    get_session_variable:
connection.loadProcedure("sap.hana.democontent.epm.Procedures::get_sessio
n_variable"),
    set_session_variable:
connection.loadProcedure("sap.hana.democontent.epm.Procedures::set_sessio
n_variable")
    get_application_variable:
connection.loadProcedure("sap.hana.democontent.epm.Procedures::get_applic
ation_variable"),
    set_application_variable:
connection.loadProcedure("sap.hana.democontent.epm.Procedures::set_applic
ation_variable")
};
```

3. Remove references to specific XS Classic schema in the JavaScript code `poWorklistUpdate.xsjs`:

You must remove all references to the schema "`SAP_HANA_DEMO`", as explained below.

Tip

In XS classic, URIs are always case sensitive.

Sample Code

Replace

a. Locate the code shown in the following example:

Sample Code

```
query = 'SELECT LIFECYCLESTATUS, APPROVALSTATUS, CONFIRMSTATUS,
ORDERINGSTATUS, INVOICINGSTATUS ' + 'from
```

```
"SAP_HANA_DEMO"."sap.hana.democontent.epm.data::PO.Header" where
PURCHASEORDERID = ?';
```

b. Replace it with the following code:

Sample Code

```
query = 'SELECT LIFECYCLESTATUS, APPROVALSTATUS, CONFIRMSTATUS,
ORDERINGSTATUS, INVOICINGSTATUS ' + 'from
"sap.hana.democontent.epm.data::PO.Header" where PURCHASEORDERID = ?';
```

4. Remove references to specific XS Classic schema in the JavaScript code `LoadCurr.xsjs`:

   Remove schema part from the following query and variable:

Sample Code

```
query = 'alter sequence
"SAP_HANA_DEMO"."sap.hana.democontent.epm.data::seriesData" RESTART WITH 1';
```

Sample Code

```
var procLoad = conn2.loadProcedure('SAP_HANA_DEMO',
'sap.hana.democontent.epm.Procedures::seriesData');
```

5. Remove references to specific XS Classic schema in the JavaScript code `poWorklistQuery.xsjs`.

   Remove all schema references from the SQL statements. This enables the Excel export of the *Purchase Order Worklist* application. As mentioned in the migration report, the export to Zip does not work as the API call used (`$.util.Zip()`) is not available in the `xsjs` compatibility library for XS advanced.

Tip

There are plenty of Open-Source alternatives available to help you complete this task. For more information, see *Related Links*.

6. Modify any upper-case values defined in request parameters requests in the JavaScript code in `poWorklistUpdate.xsjs`:

   In XS advanced, the request parameters are case sensitive, which might require some adaptations. Replace all occurrences of request parameters, for example, "`$.request.parameters.get('PURCHASEORDERID')`", as described below:

   a. Locate the code shown in the following example:

Sample Code

```
var purchaseOrderID = $.request.parameters.get('PURCHASEORDERID');
```

   b. Replace the upper-case parameter value `'PURCHASEORDERID'` as shown in the following code:

Sample Code

```
var purchaseOrderID = $.request.parameters.get('PurchaseOrderId');
```

Related Information

Making Required Code Updates [page 43]
Migrating an XS Application [page 6]
XS Advanced Migration Demo - SHINE [page 33]

4.6.3 The Web Container

Since XS advanced only supports the JSON format of OData results, the client code has to be adapted accordingly.

Procedure

1. Modify the OData-related code in the JavaScript file `Detail.controller.js` as described in the following example.
   a. Locate the code shown in the following example:

      Sample Code

      ```
      var oModel = new sap.ui.model.odata.ODataModel("/sap/hana/
      democontent/epm/services/poWorklistJoin.xsodata/", false);
      ```

   b. Replace the parameter value `false` with `true` in `poWorklistJoin.xsodata` as shown in the following code:

      Sample Code

      ```
      var oModel = new sap.ui.model.odata.ODataModel("/sap/hana/
      democontent/epm/services/poWorklistJoin.xsodata/", true);
      ```

2. Search for (and replace) the same pattern in other SAPUI5 files.

   It is recommended to search for the above pattern in the sources and replace all occurrences of similar XML OData calls.

Related Information

Making Required Code Updates [page 43]
Migrating an XS Application [page 6]
XS Advanced Migration Demo - SHINE [page 33]

4.7 Translation-Related Artifacts

Clean up any migrated translation-related artifacts.

Context

Since the XS Advanced Migration Assistant is not sure what to do with XS classic text bundles (`.hdbtextbundle`), it stores them in the `todo/` folder; you must copy these text bundles (now `.properties` files) manually to the desired destination.

Procedure

1. Check the `todo/` folder created by the XS Advanced Migration Assistant, locate the translation `.properties` files for the `Purchase Order Worklist` application, and copy them to the target locations specified in the following table:

Source (migration/todo/...)	Target (web/resources...)
`migration/todo/src/sap/hana/democontent/epm/ui/launchpad/i18n/messagebundle.properties`	`web/resources/sap/hana/democontent/epm/ui/launchpad/i18n/messagebundle.properties`
`migration/todo/src/sap/hana/democontent/epm/ui/poworklist/i18n/messagebundle.properties`	`web/resources/sap/hana/democontent/epm/ui/poworklist/i18n/messagebundle.properties`

2. Correct the code for the loaded text-bundle, if you used files with extensions other than `.properties`.

You must update the code in the `Web/` container. For the `Purchase Order Worklist` application, locate and change the indicated code in the following files:

File	Code

`web/resources/sap/hana/democontent/epm/ui/NewLaunchpad.htm`	Line 29
	`...`
	`// Internationalization`
	`// create global i18n resource bundle for texts in application U`
	`sap.app.i18n = jQuery.sap.resources({`
	`url : "./launchpad/i18n/messagebundle.properties",`
	`locale : sap.ui.getCore().getConfiguration().getLanguage()`
	`});`
	`sap.ui.getCore().setModel(sap.app.i18n, "i18n");`
	`...`

`web/resources/sap/hana/democontent/epm/ui/poworklist/Component.js`	Line 14
	`...`
	`config : {`
	`resourceBundle : "i18n/messagebundle.properties"`
	`serviceConfig : {`
	`serviceUrl : "/sap/hana/democontent/epm/services/poWorklist.xsodata/"`
	`}`
	`}`
	`...`

Related Information

Migrating an XS Application [page 6]
XS Advanced Migration Demo - SHINE [page 33]

4.8 Deploy and Run the Migrated Application

Deploy and run the migrated application.

Procedure

1. Prepare and assign XS advanced role collections that enable access to the migrated XS advanced application.
   - It is assumed that the UAA is configured as identity provider (not SAP HANA); this is the default settting in XS advanced.
   - Since the XS advanced application is protected by scopes defined in the application router configuration, the corresponding route configuration can be found in the application descriptor (`xs-app.json`), which is located in the *web/* module of the XS advanced Multi-target application (MTA).
   - The `xs-security.json` file defines the scopes and assigns the scopes to role templates. In order to enable access to the XS advanced application, you must assign the template-based roles to one or more role collections, which you then assign to the users who need access to the application.
   - The assignment tasks can be performed using the XS Advanced Administration Tools UI.
2. Deploy and run the XS advanced application in the XS advanced run-time environment.
   - To deploy the migrated SHINE application from SAP Web IDE for SAP HANA, use the built-in tools to perform the following actions:
     - Build the *db* module
     - Run the *xsjs* and *web* modules.
   - To access the application, use the link displayed in the SAP Web IDE for SAP HANA *run console* when running the *web/* module. The link will open the launchpad of the SHINE application where you will find the `Purchase Order Worklist` tile that the previous sections of the documentation were focused on. With the described adaptations, only the`Purchase Order Worklist` application part will work properly. You will have to make further changes to run the other application parts as well.
   - If you want to hide the tiles not handled in this example, you can modify the file `web/resources/sap/hana/democontent/epm/ui/launchpad/main.view.js` and remove the respective tiles:

```
...
var items = [adminTile, poTile, soTile, sowTile,
             userTile, spatialGoldTile, salesMobileTile,
             fioriTile, xsdsTile,xsUnitTile, ui5SearchTile,
             jobSchedulingTile, eTagsTil
         ]
...
```

Related Information

SAP HANA Developer Guide for SAP HANA XS Advanced
Migrating an XS Application [page 6]

XS Advanced Migration Demo - SHINE [page 33]
Security-Related Concept Check [page 39]

5 Verifying Object Migration Details

Check that the migration process has migrated all objects successfully.

During the migration, SAP HANA XS classic Repository objects are migrated to their equivalent counterparts in XS advanced their respective containers: *web*, *xsjs*, and *db*. The information in this section describes in detail the migration of the various objects and shows where to look for the equivalent migrated objects in XS advanced, for example, in the containers (*db*, *web*, and *xsjs*).

The migration tool puts objects whose destination cannot be automatically established into the *migration* folder; for these objects, you need to check and decide how you want to process them. Objects in the *migration* folder need to be copied manually by a developer into the appropriate container. This includes but is not limited to the following file types:

- `.properties`:
  This is generally a translation text file but the XS Advanced Migration Assistant cannot decide whether it really is a translation text file. It also cannot decide whether to copy this into the *web*, *xsjs*, or *db* container.
- `.xml`:
  The XS Advanced Migration Assistant checks for known structures in those files and copies them to the appropriate container. If it cannot determine the type, it will copy it to the *migration* container from where it must be moved manually to the appropriate container.
- `.json`:
  The XS Advanced Migration Assistant checks for known structures in those files and copies them to the appropriate container. If it cannot determine the type, it will copy it to the *migration* container from where it must be moved manually to the appropriate container.

Related Information

XS Migration: DB Container Objects [page 53]
XS Migration: JavaScript (JS) Container Objects [page 69]
XS Migration: Web Container Objects [page 72]
XS Migration: Deprecated Object Types [page 79]
XS Migration: Moved Files [page 80]

5.1 XS Migration: DB Container Objects

A list of the database objects migrated to the XS advanced application's database container during migration.

The information in this section lists the database- and HDI-related design time object types of the old SAP HANA classic Repository and shows what the equivalent artifact type is after migration to the XS advanced HDI.

Overview

The migration of database artifacts concerns all artifacts which are handled by using HDI.

Removal of Schema References

Schema references are automatically removed from all artifacts. If referenced database artifacts are not part of the migrated delivery unit, a synonym is generated. If artifacts are used which have no meaning in XS Advanced, the XS Migration Assistant generates messages in the report and the code must altered accordingly.

Design-Time Object Types

The database- and HDI-related design time object types of the old SAP HANA classic Repository are migrated, as far as this is possible, to the equivalent artifact type required by the XS advanced SAP HANA Deployment Infrastructure (HDI).

Migrated Database & HDI-Related Design-Time Object Types

The following table lists the database- and HDI-related design time object types of the old SAP HANA classic Repository and shows what the equivalent artifact type is after migration to the XS advanced HDI.

Table 4: Database/HDI-Related Design-Time Object Types

Object Type	Description	Supported in SAP HANA Repository	Supported in HDI	Same syntax	Migratable
`hdbtable`	Table definition for Database table	✓	✓	x	✓
`hdbschema`	Schema definition for Database schema	✓	x	x	x
`hdbsequence`	Sequence definition	✓	✓	x	✓
`hdbview`	View definition	✓	✓	x	✓
`hdbstructure`	Table template definition	✓	x	x	x
`hdbtabletype`	User-defined type	x	✓	x	x
`hdbprocedure`	SQL Script Procedure definition	✓	✓	✓	✓
`hdbtextbundle`	Text bundle for translation	✓	x	x	✓

Object Type	Description	Supported in SAP HANA Repository	Supported in HDI	Same syntax	Migratable
hdbrole	Database role definition	✓	✓	x	✓
analyticprivilege	Analytic Privilege definition	✓	✓	x	✓
analyticview	Analytic view definition	✓	x	x	✓
attributeview	Attribute view definition	✓	x	x	✓
calculationview	Calculation view definition	✓	✓	✓	✓

Related Information

Verifying Object Migration Details [page 53]
XS Migration: JavaScript (JS) Container Objects [page 69]
XS Migration: Web Container Objects [page 72]
XS Migration: Non-Container-Specific Objects [page 76]

5.1.1 hdbtable

This object type represents a database table.

Repository

Notation

- Full hdbtable specification

```
struct TableDefinition {
    string SchemaName;
    optional bool temporary;
    optional TableType tableType;
    optional bool public;
    optional TableLoggingType loggingType;
    list<ColumnDefinition> columns;
    optional list<IndexDefinition> indexes;
    optional PrimaryKeyDefinition primaryKey;
    optional string description
```

```
};
```

- Temporary
  Determines whether the table is session-specific or not. If true, the table exists only for the duration of a session and only the session owner can `INSERT/READ/TRUNCATE` data.
- TableType
  Can be either `COLUMNSTORE` or `ROWSTORE`.
- LoggingType
  Enables/disables logging. Value must be either `LOGGING` or `NOLOGGING`.
- Column Definition
  List with the columns. Each column can be specified according to the structure below.

```
struct ColumnDefinition {
    string name;
    SqlDataType sqlType;
    optional bool nullable;
    optional bool unique;
    optional int32 length;
    optional int32 scale;
    optional int32 precision;
    optional string defaultValue;
    optional string comment;
};
```

- SqlDataType

```
enum SqlDataType {
    DATE; TIME; TIMESTAMP; SECONDDATE;
    INTEGER; TINYINT; SMALLINT; BIGINT;
    REAL; DOUBLE; FLOAT; SMALLDECIMAL; DECIMAL;
    VARCHAR; NVARCHAR; CHAR; NCHAR;
    CLOB; NCLOB; ALPHANUM;
    TEXT; SHORTTEXT;
    BLOB; VARBINARY;
};
```

- Primary Key Definition
  Specification:

```
struct PrimaryKeyDefinition {
    list<string> pkcolumns;
    optional IndexType indexType;
};
```

- Table Index Definition
  Table indexes can be defined in the following structure:

```
struct IndexDefinition {
    string name;
    boolean unique;
    TableIndexOrder order;
    list<string> indexColumns;
    IndexType indexType;
}
```

  The indexType can be either `B_TREE` or `CPB_TREE`.

Table 5:

Type	Description
`B_TREE`	Specifies an index tree of type B+ which maintains sorted data that performs the insertion, deletion and searches of records.
`CBE_TREE`	Compressed Prefix `B_TREE`. Very small index which uses a partial key (a key that is only part of a full key in index nodes).

- Table Index Order
  Can be either ASC or DSC.

```
table.schemaName = "MYSCHEMA";
table.tableType = COLUMNSTORE;
table.columns = [
{name = "Col1"; sqlType = VARCHAR; nullable = false; length = 20; comment =
"dummy comment";},
{name = "Col2"; sqlType = INTEGER; nullable = false;},
{name = "Col3"; sqlType = NVARCHAR; nullable = true; length = 20;
defaultValue = "Defaultvalue";},
{name = "Col4"; sqlType = DECIMAL; nullable = false; precision = 2; scale =
3;}];
table.indexes = [{name = "MYINDEX1"; unique = true; indexColumns = ["Col2"];},
                 {name = "MYINDEX2"; unique = true; indexColumns = ["Col1",
"Col4"];}];
table.primaryKey.pkcolumns = ["Col1", "Col2"];
```

HDI

Notation

The file format uses a `DDL-style` syntax which is equivalent to the corresponding `CREATE TABLE SQL` syntax without the leading `CREATE`.

- Type is limited to `ROW` and `COLUMN` (Type specification is mandatory). Flexible and History not supported.
- Must be self contained (no 'A like B' or 'A as select ...').
- Indexes are separate objects and therefore need to be defined in a separate file.

Transformation

The table is transformed to DDL using the HANA interface procedure `SYS.GET_OBJECT_DEFINITION`. Therefore only active objects can be migrated. During the transformation, the active object will not be modified or touched.

After this first step, the DDL is further processed to meet the HDI specifications. In detail the following steps are executed:

1. The schema name is removed.

2. The definition is checked for occurrences of indexes or constraints. These are handled separately.
3. The result is written to a `.hdbtable` file in the db/src directory. The filename is equal to the old name. Also the namespace is preserved.

Indexes

Because HDI is based on an "one-object-per-file" architecture, indexes need to be defined in a separate `.hdbindex` file. During the transformation of a table, the result is scanned for index definitions. The found definitions are then extracted to separate files. During this process the schema references inside the index definitions are removed. The resulting file has the same filename as the indexname.

- Fulltextindex
  The same procedure is used for fulltextindexes which have no corresponding design-time representation in the old Repository, but had to be created using SQL. If such a fulltextindex has been defined it is extracted to a `.hdbfulltextindex` file.
- Namespace
  Although indexes were defined alongside of tables in the old Repository, the activated index did not inherit the namespace of the table. This means that the index name had to be unique inside of a schema. This behavior is respected by the Migration Assistant. The generated index files are located directly inside of db/src and not parallel to the tables in db/src/table/namespace.

5.1.2 hdbschema

This object type represents a database schema. A schema divides the database into logical parts.

Repository

- Notation

```
schema_name=”MYSCHEMA”;
```

- Characteristics
  Schema name must be identical to the name of the repository object.

HDI

HDI does not use schemas to divide the database. Instead containers are used.

Transformation

All `.hdbschema` files are removed and not included in the generated XS advance-app skelleton. The active objects are not affected by this and remain untouched on the database.

5.1.3 hdbsequence

This object type represents a sequence.

Repository

- Notation

```
string schema;
int32 increment_by(default=1);
int32 start_with(default=-1);
optional int32 maxvalue;
bool nomaxvalue(default=false);
optional int32 minvalue;
bool nominvalue(default=false);
optional bool cycles;
optional string reset_by;
bool public(default=false);
optional string depends_on_table;
optional string depends_on_view;
optional list<string> depends_on;
```

- Reset by query and dependencies
  The sequence can be reseted with a specified `reset_by` query. Any dependency that exists due to the usage of such a query must be explicitly declared using `depends_on(/_table/_view)`. In addition to that any other dependency can be modeled using these keywords. If the specified tables/views do not exist during the deployment, the activation of the sequence fails.
  During the restart of the database, the `reset_by` query is executed and the sequence is reseted to the resulting value. If no `reset_by` query has been specified, the sequence value is persistent and remains the same, even after a restart.

HDI

- Notation
  The file format uses a DDL-style syntax which is equivalent to the corresponding `CREATE_SEQUENCE SQL` syntax, without the leading `CREATE`.
- Reset by query
  During the deployment, the sequence is set to its start value which is determined by the `RESET_BY` query. If such a query has not been specified, the default start value is used.
- Dependencies

Any dependency that emerges due to the usage of a `RESET_BY` query is handled automatically during the deployment of the artifact. There is no need to model these dependencies manually anymore.

Transformation

Sequences are transformed to the DDL-based HDI syntax by the HANA built-in procedure `SYS.GET_OBJECT_DEFINITION`. The resulting DDL is then cleaned from schema references and written to an .hdbsequence file with the same filename and namespace as the old one.

5.1.4 hdbview

This object type represents a database view.

Repository

- Notation

```
string schema;
string query;
bool public(default=true);
optional list<string> depends_on;
```

- Query
  SQL `SELECT` query is used to define the view.

```
query="SELECT * FROM \"<MY_SCHEMA_NAME>\".
\"<repository.package.path>::<MY_TABLE_NAME>\"";
```

- Public
  State whether the view is publicly available. Default is true.
- Dependencies
  Comma separated list of tables/views that the defined view depends on. Specified objects need to be mentioned in the `SELECT` query of the view definition. These dependencies are not checked on the initial activation but only on reactivation.

```
depends_on=
["acme.com.test.tables::MY_TABLE1","acme.com.test.views::MY_VIEW1"];
```

> **Note**
>
> There is no consistency check based on the `query`, dependencies need not be explicitly specified.

- Example

Sample Code

```
schema="SIMPLE_SCHEMA";
query="SELECT o.\"NR\", o.\"Customer\", p.\"Price\" FROM \"SIMPLE_SCHEMA\".
\"com.sap.migration.testing.db::orders\" AS o INNER JOIN
              \"SIMPLE_SCHEMA\".\"com.sap.migration.testing.db::product\"
AS p ON p.\"ID\" = o.\"ProductID\"";
public=false;
```

HDI

- Notation
  DDL based syntax including support for column views of `TYPE HIERARCHY`.
- Example

  Sample Code

  ```
  VIEW "SIMPLE_SCHEMA"."com.sap.migration.testing.db::order_with_price"
  ( "NR", "Customer", "Price" ) AS SELECT o."NR", o."Customer", p."Price"
  FROM "SIMPLE_SCHEMA"."com.sap.migration.testing.db::orders" AS o INNER JOIN
               "SIMPLE_SCHEMA"."com.sap.migration.testing.db::product" AS p
  ON p."ID" = o."ProductID" WITH READ ONLY
  ```

- Specialities
  If the view uses an object outside the current container (or a synonym that links to such an object), the container object owner needs privileges for this object (usually `with_grant_option`).
  Usage of analytic/structured privileges must be declared by `WITH STRUCTURED PRIVILEGE CHECK` clause.
  Usage of wildcard-operator (*) can lead to non-deterministic order of the columns.
  Dependencies are verified implicitly, no explicit modeling is required anymore.

Transformation

Views are transformed to the DDL-based HDI syntax by the HANA built-in procedure `SYS.GET_OBJECT_DEFINITION`. The resulting DDL is then cleaned from schema references and written to an `.hdbview` file with the same filename and namespace as the old one. If the view contains references to objects not included in the migrated delivery unit, synonyms are generated.

5.1.5 hdbstructure

This object type represents a table-structure definition which can be reused as a template for creating new tables of the same type and structure.

Repository

- Notation
  The notation is a subset to the one of a `hdbtable` [page 55]:

```
struct TableDefinition {
    string SchemaName;
    optional bool public;
    list<ColumnDefinition> columns;
    optional PrimaryKeyDefinition primaryKey;
};
```

- Usage
  Use the `CREATE TABLE` statement to create a new table based on a table structure:

```
CREATE TABLE "MySchema"."MyTypeTable" like
"MySchema"."Structures::TableStructure"
```

HDI

The artifact type is replaced by `hdbtabletype` [page 63].

Transformation

The transformation process invokes the HANA build-in procedure `SYS.GET_OBJECT_DEFINITION` to retrieve the DDL-representation of the active version of a hdbstructure. The result is cleaned from schema references and written to a newly created `.hdbtabletype` file. The filename and namespace remain the same.

5.1.6 hdbtabletype

This object type represents a user-defined type in the structure of a table that can be used in procedure and function definitions.

Repository

This artifact type is not available in the Repository.

HDI

Design-time representation of a table type database object.

- Notation
  The file format uses a DDL-style syntax which is equivalent to the corresponding `CREATE TYPE AS TABLE SQL` syntax, without the leading `CREATE`.

Transformation

Since this artifact type was not available in the HANA Repository, there is no need to migrate such files.

5.1.7 hdbprocedure

This object type represents a SQL Script stored procedure.

Repository

- Notation
  Stored Procedures are written in SQL Script. The notation is equivalent to the `CREATE_PROCEDURE` statement without the leading `CREATE`.

HDI

- Notation

The notation is basically the same as the Repository one without any schema references. This also includes the `DEFAULT_SCHEMA` clause which is not supported anymore. If procedure needs to access database objects outside the HDI container it is deployed in, synonyms need to be used.

Transformation

Schema names are removed from the procedure name, all parameters and inside of the procedure body. If an object outside of the current HDI container is accessed, a synonym is generated for this object. The synonym has the same name as the original object. Because of this approach, the usage of synonyms is completely transparent for the procedure logic. If a `DEFAULT_SCHEMA` clause has been used, it is removed.

5.1.8 hdbtextbundle

This object type represents a collection of text strings that are displayed in the user interface. The content is used in the context of internationalization.

Repository

- Notation
  The notation is based on the syntax of a Java Properties file.

```
# TRANSLATE
# XBUT,20
BUTTON_SAVE=Save
# XBUT,20
BUTTON_CANCEL=Don't save
# XMSG,40
MSG_SUCCESS=File has been saved.
```

  `# TRANSLATE` identifies the content as translatable.
  The file contains the placeholders (e.g. `BUTTON_SAVE` in the above example), the metadata describing the length and type and an initial translation, which is used if no language specific translation is maintained.

HDI

The object type is replaced by the new XS advanced translation concept: Refer to related links.

Related Information

Translation Texts in XS Classic & XS Advanced [page 30]

5.1.9 hdbrole

This object type represents a database role.

Repository

Database roles were part of the security concept of XS classic Model. With database users also used for application access they were used to identify different access levels on the database layer.

Roles are defined using the following structure:

```
role Roles::example_role
    extends role sap.example::role1
    extends catalog role "CATROLE1", "CATROLE2"
{
    system privilege: BACKUP ADMIN, USER ADMIN;
    catalog sql object "SYSTEM"."TABLE2": SELECT;
    catalog sql object "SYSTEM"."TABLE2": INSERT, UPDATE, DELETE;
    sql object sap.example:VIEW1.attributeview, sap.example:PROC1.hdbprocedure,
sap.example:SEQ1.sequence: SELECT, DROP;
    sql object sap.example:MY_VIEW.attributeview: DROP;
    catalog schema "SYSTEM": SELECT;
    schema sap.example:app1.hdbschema: INSERT, UPDATE, DELETE;
    analytic privilege: sap.example:sp1.analyticprivilege,
sap.example:AP2.analyticprivilege;
    catalog analytic privilege: "sp3";
    package sap.example: REPO.EDIT_NATIVE_OBJECTS;
    package sap.example, sap.co: REPO.READ;
    application privilege: sap.example::Execute, sap.example::Save;
}
```

HDI

With XS advanced Model the security model changed. Application access is given to business users, the authentication and authorization is done independent from the database. The database artifacts are owned by a single technical user obviating user differentiation on database level.

However it is still possible to define roles using a design-time representation. The syntax is now JSON based and looks like the following example:

```
{
  "role":{
    "name":"RoleX",
    "global_roles":[
      "MODELING",
      "DATA ADMIN"
    ],
    "schema_roles":[
      {
        "schema_reference":"RoleSchemaRef1",
        "names": ["RoleA", "RoleB"]
      },
      {
        "schema_reference":"RoleSchemaRef2",
```

```
        "names": ["RoleB", "RoleC"]
      }
    ],
    "system_privileges":[
      "BACKUP ADMIN",
      "USER ADMIN"
    ],
    "schema_privileges":[
      {
        "reference":"Ref1",
        "privileges":[ "SELECT" ],
        "privileges_with_grant_option":[ "UPDATE" ]
      }
    ],
    "object_privileges":[
      {
        "name":"Table1",
        "type":"TABLE",
        "privileges":[ "SELECT" ],
        "privileges_with_grant_option":[ "UPDATE" ]
      }
    ],
    "schema_analytic_privileges":[
      {
        "schema_reference":"APSchemaRef1",
        "privileges": ["AP1", "AP2"],
        "privileges_with_grant_option": ["AP4"]
      },
      {
        "schema_reference":"APSchemaRef2",
        "privileges": ["AP1", "AP3"]
      }
    ]
  }
}
```

Transformation

Roles are migrated to the new format one by one preserving the namespace and the location in the project. For convenience all found and transformed roles are in the end referenced in the `default_access_role`, which is used by the deployer application to grant database privileges to the HDI Object Owner.

The single properties are transformed according the following structure:

- extends catalog role: If the mentioned role is part of the migration context, it is referenced in `schema_roles`, otherwise it is referenced in `global_roles` (these global roles are not part of the migration and need to be present on the target system).
- system privilege: Moved to corresponding `system_privileges` list.
- catalog schema: Moved to `schema_privileges`.
- schema: Moved to `schema_privileges`.
- catalog sql object: Moved to `object_privileges`.
- sql object: Moved to `object_privileges`.
- application privilege: Translated to scope and added to `xs-security.json`.
- catalog analytic privilege: If the referenced analytic privilege is part of the migration context, it is referenced in `schema_analytic_privileges`, otherwise an error message is added to the migration report.

- package: Since the repository is not used anymore, these privileges are deprecated and therefor not translated.

Synonyms:

If a privilege pointed to an object in a schema that is not part of the migrated delivery unit, a synonym is generated for this object.

Schema privileges:

If the referenced schema is part of the migration context, the privilege is redirected to the current HDI container. Otherwise a `.hdbroleconfig` file with the role name as filename is generated in the `/db/cfg` folder of the generated MTA-project. In addition to that an entry to the grantor `.hdbgrants` file is added, which is used to grant the respective database privileges of the external schema to the HDI Object Owner during the deployment.

5.1.10 Modeler Artifacts (Attribute, Analytic, Calcview etc.)

A list of the modeler-related database objects migrated to the XS advanced application's database container during migration.

The information in this topic describes the handling of all modeler-related BI artifacts during the application-migration process. The information covers the following artifacts:

- Attribute views (`.attributeview`)
- Analytic views (`.analyticview`)
- Calculation views (`.calculationview`) both graphical and scripted
- Analytic privileges (`.analyticprivileges`)

The migration process is divided into the following distinct steps:

1. Conversion of the XS classic view to the 1-View concept
   Use the *Migrate* tool in SAP HANA Studio's *Modeler* perspective to convert the following view types to “1-View” before running the XS Migration Assistant:
   - Convert attribute views and analytic views to **graphical** calculation views
   - Convert script-based calculation views to graphical calculation views and table functions
   - Convert classical XML-based analytic privileges to SQL analytic privileges
2. Migration of the converted view to HDI
   Newly created (or existing) “1-View”-compatible artifacts are processed to make them compatible with the SAP HANA Deployment Infrastructure's (HDI) deployer. This step is done by either by a plug-in or by the XS Migration Assistant. To prevent ambiguity in XS advanced, the “`hdb*`” prefix is added to all file extensions, for example, `hdbanalyticview`.

   > **Note**
   >
   > The migration process removes a number of deprecated features.

Deprecated and Obsolete Features

The following features are deprecated during the migration to HDI in XS advanced:

- Translation relevance
  Not relevant anymore in HDI; use `.property` files instead.
- Relational Optimization
  Used for analytic views and not supported by calculation views in HDI
- Execution Semantic
  Not supported since SP06; removed during migration.
- Generate Concat Attributes
  Not supported for calculation views; removed during migration.
- Default Schema
  Not supported in HDI; removed during migration.

Adjustments for HDI

Adjustments are made to the following elements during migration to HDI:

- Currency and Unit Conversion
  - Currency column
    No automatic generation of currency column. All currency and unit columns must be explicitly modeled.
  - Currency conversion schema
    In XS Classic it was only necessary to specify the schema in which the engine would look up the conversion tables, for example, `TCUR`, `T006`. At run time these tables are accessed by the conversion engine with “invoker” privileges. In XS advanced (HDI). all table references have to be made explicitly. For this reason, references are added to the conversion tables in the conversion model. These model features (XML attributes) have to be filled with the names of synonyms pointing to the actual conversion tables. The current run-time HDI container will always be passed as conversion schema passed to the calculation and conversion engine.
- Derived Parameters
  The model for derived parameters has been streamlined for HDI.
- Pruning
  References to pruning tables are now modeled as design time links to tables that have to exist.
- Privileges
  - HDI supports only SQL-based analytic privileges that can be modeled using the `applyPrivilegeType` attribute. If the `applyPrivilegeType` is either missing or empty, **no** privileges are applied.
  - The old XML attribute `checkAnalyticPrivileges` is ignored.

Related Information

XS Migration: DB Container Objects [page 53]

5.2 XS Migration: JavaScript (JS) Container Objects

A list of objects which are migrated to the `xsjs` compatibility layer for XS classic applications to run on the `Node.js`-based JavaScript runtime environment in XS advanced.

The following file types will be migrated into the `xsjs` compatibility layer:

1. XS Classic Javascript Files (.xsjs, and .xsjslib) [page 69]
2. XS Classic HTTP Destination Files (.xshttpdest) [page 70]
3. XS Classic Job Files (.xsjob) [page 71]
4. XS Odata Files (.xsodata) [page 71]
5. XS Classic SQL Files (.xssqlcc) [page 72]

> **Note**
>
> In addition, it might be necessary to move certain files from the *todo* container into the *xsjs* container; namely the `.properties` files (translation text files). For more information, see *Related Links*.

XS classic Javascript files (.xsjs and .xsjslib)

XS classic Javascript files have one of two file extensions: XS JavaScript application code "`.xsjs`" or XS JavaScript libraries "`.xsjslib`".

The migration tool does not attempt to make any changes to these files and merely copies them to the *xsjs* container into the given directory which reflects the package hierarchy in the XS classic runtime. The migration tool however runs several source code scans in order to scan for statements that require manual attention. This is mostly related to security-relevant statements, but not limited to it.

The migration tool parses all JavaScript files, checks for the usage of the following APIs, and generates messages if they are used:

- `$.repo` APIs: For the APIs under `$.repo` it is officially documented that they will not be supported in XS advanced. Code that uses those APIs must be adapted.
- `$.session.hasAppPrivilege` and `$.session.assertAppPrivilege`: Application privileges were replaced by the *scopes* concept. The privilege names must be aligned with new scope names defined in the `xs-security.json` file.
- `$.session.hasSystemPrivilege` and `$.session.assertSystemPrivilege`: These APIs are not supported in XS advanced and code needs to be changed.
- `$.config`: This API which has never been officially documented is not supported and the code must be changed

In addition to the API checks the migration tool searches for the usage of the following strings which point to the usage of database tables which no longer have a meaning in XS advanced:

- `_SYS_BIC`: All views and procedures were migrated to the local schema. The schema prefix is no longer needed.
- `_SYS_REPO`: This schema no longer has a meaning for XS advanced applications and the code accessing this schema must be adapted.

Required Migration Tasks in `xsjs` files

The following migration tasks need to be carried out by the development team:

- Security-related migration as described in the security migration section. This mainly means to modify code to do scope checks instead of invoking `hasAppPrivilege` or `assertAppPrivilege`.
- Code which uses the APIs as outlined in the report must be changed to use the new APIs.
- All schema references must be removed from SQL statements. For references that point to objects that have not been migrated (e.g. to a different schema) synonyms must be created.
- Code which uses database artifacts from other schemas, especially from `SYS` and `_SYS_REPO` must be changed.

Code replacement is also required

Use CDS libs in `xsjs`

In XS classic code:

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

In XS advanced code:

```
var conn  = $.db.getConnection();
                    var XSDS = $.require(" sap-cds ").xsjs(conn);  // “cds”
refers to node-cds
```

Use dbutils

In XS classic code:

```
var XSProc  = $.import(" sap.hana.xs.libs.dbutils ", " procedures ");
                    var proc  = XSProc.allProcedures(" SHINE_LIGHT ", "
shineLight.procedures ");
```

In XS advanced code:

```
var connection  = $.hdb.getConnection();
                    var proc  = {
                    get_session_variable : connection.loadProcedure("
shineLight.Procedures::get_session_variable "),
                    set_session_variable : connection.loadProcedure("
shineLight.Procedures::set_session_variable "),
                    get_application_variable : connection.loadProcedure("
shineLight.Procedures::get_application_variable "),
                    set_application_variable : connection.loadProcedure("
shineLight.Procedures::set_application_variable ")
                    };
```

XS classic HTTP Destination files (.xshttpdest)

HTTP destination files define connection details for remote systems that provide services required by an XS classic application; HTTP destination files have the file extension “`.xshttpdest`”.

`xshttpdest` objects are supported in XS advanced. Connection information should be provided using the user-provided service.

XS Classic Job Files (.xsjob)

XS classic Job files define job schedules running in the XS classic run-time environment; they have the file extension “`.xsjob`”.

`xsjob` files are supported in XS advanced without modification. There have been slight changes in how schedules are defined in XS advanced but the job scheduler run-time detects old schedules and handles them accordingly.

XS Odata Files (.xsodata)

XS OData Files (`.xsodata`)

`.xsodata` files are migrated by the XS Migration Assistant.

> **Note**
>
> The xsodata on XS advanced implements the OData standard more strictly in the sense that batch requests to such service endpoints must use ‘`\r\n`’ line endings instead of ‘`\n`’.

The following changes are applied to `.xsodata` files:

- Schema references are removed as per the following example:

  ```
  "SAP_HANA_DBCC"."sap.hana.dbcc.data::RESOURCE_URLS" as "DBCCResourceURLs"
                                  create forbidden
                                  update forbidden
                                  delete forbidden;
  ```

  is automatically migrated to

  ```
  "sap.hana.dbcc.data::RESOURCE_URLS" as "DBCCResourceURLs"
                                  create forbidden
                                  update forbidden
                                  delete forbidden;
  ```

- In some instances "/" has been used as a separator between package name and object name. The XS Migration Assistant will replace the "/" with a "::" and remove the file extension. This is an example snippet from a calculation view

  ```
  "sap.ead.fnd.views/CA_DASHBOARDS.calculationview" as "Dashboards"
  keys("DashboardID")
                                  create forbidden
                                  update forbidden
                                  delete forbidden;
  ```

  which is migrated to

  ```
  "sap.ead.fnd.views::CA_DASHBOARDS" as "Dashboards"
                              keys("DashboardID")
                              create forbidden
                              update forbidden
                              delete forbidden;
  ```

XS Classic SQL Files (.xssqlcc)

XS classic SQL files

`xssqlcc` files in XS classic enable the execution of SQL statements by server-side JavaScript code with different credentials. In XS classic all database operations are executed by default with the access privileges by the requesting user. This could be changed with the `xssqlcc` mechanism where the credentials configured for the `xssqlcc` connection were used.

In some instances this mechanism may no longer be required. It was, however, ported to XS advanced, but the `xssqlcc` files are no longer needed and are not migrated.

The `xssqlcc` connection property can be configured by the following code:

```
options.hana.sqlcc = xsenv.getServices({
    'com.sap.my.sqlcc_config': 'SQLCC__NAME',
    'com.sap.my.other_sqlcc_config': 'OTHER_SQLCC_UPS_NAME'
});
```

and used later in `xsjs`code like:

```
var connection = $.db.getConnection('com.sap.my.sqlcc_config');
```

Related Information

XS Migration: DB Container Objects [page 53]
XS Migration: Web Container Objects [page 72]
Verifying Object Migration Details [page 53]

5.3 XS Migration: Web Container Objects

A list of the objects that are copied to the “web” run-time container during migration to XS advanced.

This objects copied to the application project's “web” container includes (but is not limited to) the following file types:

- `.html` and `.css`
- plain text files (`.txt`)
- JavaScript executed in the browser
- Images, such as `.gif`, `.png`, or `.jpeg` files
- Scalable Vector Graphics files (SVG)

- SAP UI 5 artifacts, such as `.control` or `.library` files.
  For more information , see *Related Links*.

It is important to review the list of files copied to the web container since in some rare cases this might lead to the exposure of files to external users which are not meant to be exposed because they are used by the xsjs runtime. Check the XS classic `.xsaccess` files in each web-related package and make sure that all protected XS classic files will also be protected in XS advanced.

It might be necessary to copy translation text files (`.propertiesfiles`) from the *todo* folder to the *web* folder. The XS Migration Assistant cannot determine if a translation file covers translation for texts in the Web browser, server-side xsjs code, or whether it is required for database artifacts. The XS Advanced Migration Assistant does not make any changes to any of the files. For this reason, in some cases, manual migration might be required.

The following manual changes are required for translation text bundles for SAP UI 5 applications:

1. OData requests must be in JSON format, as illustrated in the following example:

   ```
   var oModel = new sap.ui.model.odata.ODataModel(serviceUrl, true);
   ```

2. Check the UI5 loading path in HTML (in `sap-ui-bootstrap`)
   The XS Advanced Migration Assistant replaces your library location to `sapui5.hana.ondemand` as illustrated below; you might need to change it.

   ```
   <script src="https://sapui5.hana.ondemand.com/resources/sap-ui-core.js"
   id="sap-ui-bootstrap"
                           ....></script>
   ```

Migrated Web & UI-Related Design-Time Object Types

The following table lists the Web- and UI-related design time object types of the old SAP HANA classic Repository and their shows what the equivalent artifact type is after migration to the XS advanced HDI.

Table 6: Web/UI-Related Design-Time Object Types

Object Type	Description	Supported in SAP HANA Repository	Supported in HDI	Same syntax	Migratable
`xsaccess`	XS Access Configuration	✓	x	x	✓

Related Information

Verifying Object Migration Details [page 53]
XS Migration: Moved Files [page 80]

5.3.1 xsaccess

The `.xsaccess` file used for application-access control in XS classic is migrated to the application descriptor (`xs-app.json`) in XS advanced.

Discovery

In XS advanced, the application-router (approuter) configuration defined in the `xs-app.json` is located in one central file in the application router's current working directory. To facilitate the migration process, collect all `xsaccess` files for an XS classic project and aggregate them into one file.

Analysis

In this step, the properties defined in the old `xsaccess` files are transformed into the corresponding parameters of the approuter.

The following table lists the properties allowed in the XS classic application-access descriptor (`.xsaccess`), shows if it can be migrated to XS advanced, and if it can, what the migrated property is.

> **Note**
>
> Not all the properties are transformed; some XS classic properties have no counterpart in XS advanced.

Table 7: Transformation of Application-Access Properties

XS Classic Property (`xsaccess`)	XS advanced Property (`xs-app.json`)	Comments
`exposed`	-	Not required in XS advanced due to a different security concept
`authentication`	-	Ignored: the MTA project generated in XS advanced has authentication enabled by default
`authorization`	✓	The privileges are transformed to XSA scopes and routes protected by theses scopes.
`anonymous_connection`	-	Not required in XS advanced due to a different concept for database access
`default_connection`	-	Not required in XS advanced due to a different concept for database access
`cache_controll`	X	No corresponding feature in XS advanced
`cors`	X	No corresponding feature in XS advanced

XS Classic Property (`xsaccess`)	XS advanced Property (`xs-app.json`)	Comments
`default_file`	(✓)	**Partially** replaced by "`welcomeFile`" (which is a singleton for an application)
`enable_etags`	X	No corresponding feature in XS advanced
`force_ssl`	✓	"`forceSSL`" can be enabled for destinations defined by an environment variable or in the application's manifest file
`mime_mapping`	X	No corresponding feature in XS advanced
`prevent_xsrf`	✓	"`csrfProtection`"can be enabled for individual routes; enabled by default
`rewrite_rules`	-	-
`headers`	(✓)	Feature only possible via application environment settings

Generation of the xs-app.json File

The following basic template is used for the application descriptor (the approuter configuration file).

Sample Code

```
{
    "welcomeFile" : "index.html",
    "authenticationMethod": "route",
    "routes" : [
    {
        "source": ".*xsjs.*",
        "destination": "xsjs"
    },
    {
        "source": ".*xsodata.*",
        "destination": "xsjs"
    }]
};
```

welcomeFile

If an `.xsaccess` file with the `default_file` parameter was found at the application root of the XS classic project (the package where the `.xsapp` file is located), the value of this parameter is used (translated to the new file path).

routes

Since privileges are translated to scopes, the privilege configuration by the `authorization` parameter is translated to separate routes with the location of the old `.xsaccess` file as a target.

Related Information

XS Migration: Web Container Objects [page 72]
Verifying Object Migration Details [page 53]
The `.xsaccess` file (XS classic)

5.4 XS Migration: Non-Container-Specific Objects

A list of the design-time objects migrated to one or more of the XS advanced application's containers during migration.

The information in this section lists the design time object types of the old SAP HANA classic Repository and shows what the equivalent artifact type is after migration to XS advanced. The objects listed here are not tied to a specific XS advanced container, for example, database (`hdb/`) or Web (`ui/`).

Migrated Non Container-Specific Design Time Object Types

The following table lists the design time object types of the old SAP HANA classic Repository which are not specific to a particular XS advanced container type and shows what the equivalent artifact type is after migration to the XS advanced.

Table 8: Container-Overlapping Design Time Object Types

Object Type	Description	Supported in SAP HANA Repository	Supported in HDI	Same syntax	Migratable
`xml`	Various usage	✓	✓	✓	✓
`json`	Various usage	✓	✓	✓	✓
`properties`	Mostly translation or UI5 related	✓	✓	✓	✓

Related Information

Verifying Object Migration Details [page 53]
XS Migration: DB Container Objects [page 53]
XS Migration: JavaScript (JS) Container Objects [page 69]
XS Migration: Web Container Objects [page 72]

5.4.1 xml

Information about how `.xml` files are processed in the context of an application migration to XS advanced.

Container assignment strategy

If the file is of one of the following types, it is copied to the Web container (typically named `web/`) of the generated MTA-project.

Table 9: .xml files

Type	Identified by element
`View`	`sap.ui.core.mvc`
`UI Lib`	`sap.hana.admin.cockpit.uilib`
`Core View`	`xmlns:core="sap.ui.core"`
`UI5 Viz`	`xmlns:viz="sap.viz.ui5"`
`UI5 Layout`	`xmlns:layout="sap.ui.commons.layout"`
`UI5 Form`	`xmlns:form="sap.ui.commons.form"`
`UI5 site.xml`	`site.xml`

If the file cannot be recognized as one of the mentioned types, it is moved to the `todo` folder and a respective message is added to the report.

Related Information

XS Migration: Non-Container-Specific Objects [page 76]
Verifying Object Migration Details [page 53]

5.4.2 json

This section describes how `.json` files are processed in the context of a migration.

Container assignment strategy

If the file is of one of the following types, it is copied to the web container of the generated mta-project.

Table 10: .json files

Type	Identified by
View	`controllerName`
Library Dependencies File	`library.dependencies.json` or `library.parameters.json`
UI5 File	`library-parameters.json` or `library-preload.json`

If the file cannot be recognized as one of the mentioned types, it is moved to the `todo` folder and a respective message is added to the report.

Related Information

XS Migration: Non-Container-Specific Objects [page 76]
Verifying Object Migration Details [page 53]

5.4.3 properties

This section describes how `.properties` files are processed in the context of a migration

Container assignment strategy

If the filename is `library.properties`, the originates from UI5 and is moved to the web folder of the generated MTA-project.

Otherwise the file could have been used for translation. It is moved to the `todo` folder and a respective message is added to the report.

Related Information

XS Migration: Non-Container-Specific Objects [page 76]
Verifying Object Migration Details [page 53]

5.5 XS Migration: Deprecated Object Types

Some object types cannot be migrated from XS classic to XS advanced.

The files listed in the following table cannot be migrated to XS advanced and are not moved to the generated MTA-project.

> **Note**
>
> Strictly speaking, not all of the object types listed in the following table are XS classic objects and, for this reason, cannot be "deprecated", for example: Eclipse-related files such as `.project` and `.classpath` or the Windows `.bat` file, and Python `.pyc` files. The one thing in common that **all** the files in this table share is that they are no longer relevant for XS advanced applications, and are not migrated by the XS Advanced Migration Assistant.

Table 11: Non-Migratable Object Types

Object Type	Description
*.xsapp	An application-specific file in an XS classic repository package that defines the root folder of a native SAP HANA application. All files in that package (and any subpackages) are available to be called via URL.
*.project	An Eclipse project descriptor used in SAP HANA studio for . The `.project` file is a design-time artifact that is stored in the SAP HANA XS classic repository.
*.pyc	Compiled Python files
*.regiignore	A file containing ignore of exclude patterns for XS Classic repository used by HANA Studio Repository Team Provider
*.DS_Store	MacOS Desktop Services Store used for custom file attributes, icons, etc. Should not be committed to any project repository.
*.bat	MS DOS Batch files
*.classpath	Eclipse JDT project Classpath configuration
*.jdtscope	Eclipse JDT project configuration
*.gitignore	A file containing ignore (exclude) patterns for Git client
*.db	A database file used by Sybase SQL Anywhere, SQLite, and others

Object Type	Description
*.xsprivileges	A file that defines a privilege that can be assigned to an SAP HANA Extended Application Services application, for example, the right to start or administer the application.
*.xssecurestore	The design-time file that creates an application-specific secure store; the store is used by the application to store data safely and securely in name-value form.
*.xssqlcc	A file that enables execution of SQL statements from inside server-side JavaScript code with credentials that are different to those of the requesting user
*.hdbrole	A file containing a design-time definition of an SAP HANA user role for XS classic.

Related Information

XS Migration: DB Container Objects [page 53]

5.6 XS Migration: Moved Files

A lists of the different file types that are moved to the new MTA-project during the application migration.

The following table lists the files moved during the migration of an XS classic application and shows to which XS advanced application container the file is moved.

Table 12: Moved Files

Containers	Files
XSJS runtime container	• xsjs • xsjslib • xsjob • mustache • xshttpdest

Containers	Files
Web container	• html • htm • txt • js • jpg • JPG • gif • jpeg • png • PNG • css • ico • svg • less • type • control • library • theming • WOFF • woff • woff2 • ttf • eot • jar • apk • rtf • doc • docx • pdf • cur • swf • otf

Containers	Files
DB container	• hdbtable • hdbprocedure • hdbscalarfunction • hdbtablefunction • hdbsequence • hdbview • hdbsynonym • calculationview • hdbstructure • csv • hdbafllangprocedure • analyticprivilege

Related Information

Verifying Object Migration Details [page 53]

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).

go.sap.com/registration/contact.html