

SAP HANA Platform 2.0 SPS 02  
Document Version: 1.0 – 2017-07-26

# SAP HANA Developer Guide

## For SAP HANA XS Advanced Model



# Content

<b>1</b>	<b>SAP HANA Developer Guide for XS Advanced Model</b>	<b>9</b>
<b>2</b>	<b>Introduction to Application Development and Deployment (XS Advanced Model)</b>	<b>10</b>
2.1	SAP HANA Extended Application Services, Advanced Model	11
2.2	SAP HANA XS Advanced Multi-Target Applications	12
2.3	The XS Advanced Programming Model	13
2.4	XS Advanced Application Development Tools	21
2.5	Developer Information Map for XS Advanced	23
<b>3</b>	<b>Getting Started with Application Development in XS Advanced</b>	<b>25</b>
3.1	Working with the SAP Web IDE for SAP HANA	26
	Create a Simple "Tiny-World" Application	27
	Extend the "Tiny World" Application to Use an OData Service and a Client UI	33
	Add Business Logic to the "Tiny World" Application	38
	Deploy the "Tiny World" Application	47
	Upgrade and Redeploy the "Tiny World" Application	52
	Add User Authentication to the "Tiny World" Application	59
	Add Authorization Checks to the "Tiny World" Application	63
	Create and Assign User Roles for the "Tiny World" Application	67
	Working with the SAP HANA Database Explorer	70
3.2	Working with the XS Advanced Command-Line Client	78
	Get Started with the XS CLI Client	79
	Deploy a Node.js Hello-World Application using the XS CLI	84
	Deploy a Node.js Application that Uses Database Artifacts and OData Services	89
3.3	Working with Gerrit Version Control in XS Advanced	98
	Set up Gerrit for XS Advanced Application Development	99
3.4	Set up an XS Advanced Application Project	104
	XS Advanced Application Resource Files	110
<b>4</b>	<b>Maintaining the XS Advanced Application Development &amp; Deployment Descriptors</b>	<b>112</b>
4.1	Create the MTA Description Files	113
	Multi-Target XS Advanced Applications	115
	The MTA Development Descriptor	117
	The MTA Deployment Descriptor	121
	The MTA Deployment Extension Description	164
4.2	Deploy a Multi-Target Application (with Command-Line Tools)	166
	The MTA Archive	170
	The MANIFEST.MF File	170

<b>5</b>	<b>Defining the Data Model in XS Advanced .....</b>	<b>172</b>
5.1	Maintaining HDI Containers .....	173
	Naming Conventions in HDI.....	175
	Set up an HDI Container.....	176
	Define HDI Run-Time Name-Space Rules.....	183
5.2	Configuring the HDI Deployer .....	188
	Integrate the HDI Deployer into an XS Advanced MTA Database Module.....	189
5.3	Creating the Data Persistence Artifacts in XS Advanced.....	199
	Create the Data Persistence Artifacts in XS Advanced.....	200
	Create a CDS Document (XS Advanced).....	208
	Create a CDS Entity in XS Advanced.....	243
	Create a CDS User-Defined Structure in XS Advanced.....	264
	Create a CDS Association in XS Advanced.....	278
	Create a CDS View in XS Advanced.....	293
	Create a CDS Extension.....	316
	Create a CDS Access-Policy Document (XS Advanced).....	332
	Create a CDS Aspect (XS Advanced).....	336
	Create a CDS Role (XS Advanced).....	337
5.4	Maintaining JSON Collections in the SAP HANA Document Store.....	342
	Create a JSON Document Store.....	343
	Add and Remove JSON Collections in the Document Store.....	345
	Import JSON Documents into a DocStore Collection.....	349
5.5	Creating Procedures and Functions in XS Advanced.....	352
	Create the Underlying Data Artifacts for a Procedures.....	353
	Create a Database Procedure in XS Advanced.....	363
	Create a Database Function in XS advanced.....	366
5.6	Using Synonyms to Access External Schemas and Objects in XS Advanced.....	369
	Enable Access to Objects in a Remote Classic Schema.....	370
	Enable Access to Objects in Another HDI Container.....	394
5.7	Setting Up the Analytic Model.....	400
	Create Graphical Calculation Views.....	400
	Preview Calculation View Output.....	404
	Defining Data Access Privileges.....	406
<b>6</b>	<b>Defining Web-based Data Access .....</b>	<b>407</b>
6.1	Maintaining OData Services in XS Advanced .....	407
	Defining OData v2 Services for XS Advanced JavaScript Applications.....	408
	Defining OData v4 Services for XS Advanced Java Applications.....	463
6.2	Data Access with XMLA in SAP HANA XS.....	495
	Define the Data to Expose with an XMLA Service in XS Advanced.....	496
	Set up and Use the XMLA Interface in XS Advanced.....	497

<b>7</b>	<b>Writing the XS Advanced Application Code.....</b>	<b>509</b>
7.1	The SAP HANA XS Advanced JavaScript Run Time. ....	510
	Secure Programming with JavaScript. ....	514
	Download and Consume SAP Node.js Packages. ....	516
	Download and Install JavaScript Data Services. ....	562
	Tutorial: Setting up your JavaScript Application in XS Advanced. ....	572
7.2	The SAP HANA XS Advanced Java Run Time. ....	589
	Download and Consume Java Libraries. ....	590
	Tutorials: Setting Up the Java Run-Time Environment in XS Advanced. ....	602
7.3	Custom Run-Time Environments in SAP HANA XS Advanced. ....	636
	Best Practices for Deploying Applications in a Custom Run-Time Environment. ....	637
	Create a Python Buildpack for XS Advanced. ....	640
	Create a PHP Buildpack for XS Advanced. ....	645
	Available Build Packs. ....	651
7.4	Create the XS Advanced Application Package Descriptor. ....	652
	The XS Advanced Application Package Descriptor. ....	653
	XS Advanced Application Package Descriptor Configuration Syntax. ....	656
<b>8</b>	<b>Maintaining Application Services in XS Advanced. ....</b>	<b>660</b>
8.1	Default Services in XS Advanced. ....	661
	File-System Storage Services in XS Advanced. ....	662
	Deployment-Infrastructure Services in XS Advanced. ....	664
	User Account and Authentication Services in XS Advanced. ....	668
	Audit Log Services in XS Advanced. ....	673
	Job Scheduler Services in XS Advanced. ....	675
8.2	Service Plans and Resources in XS Advanced. ....	678
8.3	Service Types in XS Advanced. ....	679
8.4	Create a Service Instance. ....	681
	The SAP HANA Service Broker. ....	682
	The SAP HANA Service-Broker API. ....	683
8.5	Update a Service Instance. ....	685
8.6	Scheduling Jobs in XS Advanced. ....	687
	Configure the Service Broker for Job Scheduler. ....	687
	Create a Job Scheduler Service Instance. ....	689
	Bind an Application to the Job Scheduler Service. ....	690
	Maintain Jobs and Job Schedules in XS Advanced. ....	691
<b>9</b>	<b>Creating the Client User Interface. ....</b>	<b>713</b>
9.1	Building User Interfaces with SAPUI5 for SAP HANA. ....	713
9.2	Consuming Data and Services with SAPUI5 for SAP HANA. ....	714
<b>10</b>	<b>Maintaining XS Advanced Application Routes and Destinations. ....</b>	<b>716</b>

10.1	Configure the XS Advanced Application Router.....	717
	XS Advanced Application-Router Resource Files.....	720
	The XS Advanced Application Descriptor.....	721
	XS Advanced Application Router Configuration Syntax.....	725
	XS Advanced Application Routes and Destinations.....	739
	XS Advanced Application Router Environment Variables.....	741
10.2	Extend the XS Advanced Application Router.....	749
	XS Advanced Application Router Extensions.....	751
	Custom Middleware Injection.....	753
	XS Advanced Application Router Extension API.....	754
<b>11</b>	<b>Setting Up Security Artifacts.....</b>	<b>757</b>
11.1	Maintaining Application Security in XS Advanced.....	758
	OAuth 2.0 in XS Advanced.....	760
	Building an Authorization Concept for Users of an XS Advanced Application Instance.....	760
	Example Scenario for Leave Request Authorizations.....	761
11.2	Create the Security Descriptor for Your XS Advanced Application.....	763
	The Application Security Descriptor.....	764
	Application Security Descriptor Configuration Syntax.....	767
11.3	Create an Instance of the XS UAA Service.....	773
	Modifications and Restrictions for XSUAA Service Updates.....	775
11.4	Bind the XS UAA Service Instance to the XS Advanced Application.....	776
11.5	Assign Functional Authorization Scopes for an XS Advanced Application.....	777
	Authorization in SAP HANA XS Advanced.....	778
11.6	Enable Applications to Perform Tasks with the Authority of Other Applications.....	781
	Authorization Scopes and Authorities.....	783
<b>12</b>	<b>HDI Artifact Types and Build Plug-ins Reference.....</b>	<b>785</b>
12.1	AFL Language Procedures (.hdbaflangprocedure).....	787
12.2	Analytic Privileges (.hdbanalyticprivilege).....	788
12.3	Calculation Views (.hdbcalculationview).....	789
12.4	Constraints (.hdbconstraint).....	790
12.5	Copy Only (.txt).....	791
12.6	Core Data Services (.hdbcds).....	792
12.7	Document Store Collections (.hdbcollection).....	793
12.8	Full-Text Indexes (.hdbfulltextindex).....	794
12.9	Functions (.hdbfunction).....	795
12.10	Graph Workspaces (.hdbgraphworkspace).....	796
12.11	Hadoop Map Reduce Jobs (.hdbmrjob or .jar).....	797
12.12	Indexes (.hdbindex).....	798
12.13	Libraries (.hdbllibrary).....	799
12.14	Logical Schema Definition (.hdblogicalschema).....	800

12.15	Procedures (.hdbprocedure) . . . . .	801
12.16	Projection Views (.hdbprojectionview) . . . . .	802
12.17	Public Synonym (.hdbpublicsynonym) . . . . .	804
12.18	Result Cache (.hdbresultcache) . . . . .	806
12.19	Roles (.hdbrole) . . . . .	807
12.20	Search Rule Set (.hdbsearchruleset) . . . . .	812
12.21	Sequence (.hdbsequence) . . . . .	814
12.22	SQL Views (.hdbview) . . . . .	815
12.23	Statistics (.hdbstatistics) . . . . .	816
12.24	Structured Privilege (.hdbstructuredprivilege) . . . . .	817
12.25	Synonyms (.hdbsynonym and .hdbsynonymconfig) . . . . .	818
12.26	_SYS_BIC Synonym (.hdbsysbicsynonym) . . . . .	821
12.27	Tables (.hdbtable and .hdbdropcreatetable) . . . . .	822
12.28	Table Data (.hdbtabledata) . . . . .	825
12.29	Table Data Properties (.properties) . . . . .	834
12.30	Table Type (.hdbtabletype) . . . . .	836
12.31	Text Analysis Configuration (.hdbtextconfig) . . . . .	837
12.32	Text Analysis Dictionaries (.hdbtextdict) . . . . .	838
12.33	Text Analysis Extraction Rules (.hdbtextrule) . . . . .	838
12.34	Text Analysis Extraction Rules Includes (.hdbtextinclude) . . . . .	839
12.35	Text Analysis Extraction Rules Lexicon (.hdbtextlexicon) . . . . .	840
12.36	Text Mining Configurations (.hdbtextminingconfig) . . . . .	841
12.37	Triggers (.hdbtrigger) . . . . .	841
12.38	Virtual Functions (.hdbvirtualfunction) . . . . .	842
12.39	Virtual Procedures (.hdbvirtualprocedure) . . . . .	844
12.40	Virtual Tables (.hdbvirtualtable) . . . . .	845
12.41	Virtual Packages (.hdbvirtualpackage) . . . . .	847
<b>13</b>	<b>The XS Command-Line Interface Reference . . . . .</b>	<b>848</b>
13.1	XS CLI: Logon and Setup . . . . .	850
13.2	XS CLI: Application Management . . . . .	852
13.3	XS CLI: Services Management . . . . .	874
13.4	XS CLI: Organizations . . . . .	890
13.5	XS CLI: Spaces . . . . .	892
13.6	XS CLI: Domains . . . . .	896
13.7	XS CLI: Certificates . . . . .	899
13.8	XS CLI: Routes . . . . .	901
13.9	XS CLI: Buildpacks . . . . .	904
13.10	XS CLI: Run-Time Environments . . . . .	908
13.11	XS CLI: Tasks . . . . .	915
13.12	XS CLI: User Administration . . . . .	917
13.13	XS CLI: Administration . . . . .	929

13.14	XS CLI: Configuration . . . . .	931
13.15	XS CLI: Blob Store . . . . .	934
13.16	XS CLI: Advanced . . . . .	936
13.17	XS CLI: Other Commands . . . . .	937
13.18	XS CLI: Plug-ins . . . . .	939
<b>14</b>	<b>SAP Web IDE for SAP HANA Reference . . . . .</b>	<b>959</b>
14.1	Product Overview . . . . .	959
14.2	Post-Installation Administration Tasks . . . . .	962
	Enabling Access to the SAP Web IDE Administration and Development Tools . . . . .	962
	Managing Spaces for Development . . . . .	965
	Managing SSL Certificates . . . . .	966
14.3	Getting Started . . . . .	966
	Navigating SAP Web IDE . . . . .	967
	Set User Preferences . . . . .	969
	Enabling Optional Features . . . . .	971
	Keyboard Shortcuts . . . . .	972
	Workspace Search Options . . . . .	975
	Working with Files and Folders . . . . .	980
	Resizing Panes . . . . .	982
14.4	Developing Multi-Target Applications . . . . .	983
	Inside an MTA Descriptor . . . . .	985
	Application Development Workflow . . . . .	989
	Setting Up Application Projects . . . . .	989
	Developing SAP HANA Database (HDB) Modules . . . . .	992
	Developing Node.js Modules . . . . .	1025
	Developing Java Modules . . . . .	1033
	Developing HTML5 Modules . . . . .	1039
	Developing SAP Fiori Launchpad Modules . . . . .	1080
	Packaging and Deploying Applications . . . . .	1092
14.5	Customizing Your Project . . . . .	1093
	Customizing JavaScript Beautifier Properties . . . . .	1093
	Customizing Code Checking Rules . . . . .	1094
	Configuring Mock Data Usage . . . . .	1095
14.6	Using Code Editors . . . . .	1096
	Configuring the Code Editor . . . . .	1096
	Working in the Code Editor . . . . .	1097
	Generating JSDoc Comment Snippets . . . . .	1101
	Using Code Completion . . . . .	1102
	Checking Code . . . . .	1105
	Locating Objects in Code . . . . .	1111
14.7	Using Source Control (Git) . . . . .	1113

Setting Up Git. . . . .	1117
Cloning Repositories. . . . .	1119
Initializing a Local Git Repository. . . . .	1120
Setting a Remote Repository. . . . .	1120
Fetching Changes. . . . .	1121
Rebasing Changes. . . . .	1122
Merging Changes. . . . .	1123
Pulling Changes. . . . .	1124
Staging Files. . . . .	1124
Committing Changes. . . . .	1126
Pushing Changes. . . . .	1127
Using Multiple Local Branches. . . . .	1127
Creating a Remote Branch. . . . .	1129
Viewing Git History. . . . .	1130
Setting Up Git to Work with Gerrit. . . . .	1131
14.8 Troubleshooting. . . . .	1133
Archive Import Troubleshooting. . . . .	1133
Git Troubleshooting. . . . .	1134

# 1 SAP HANA Developer Guide for XS Advanced Model

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

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

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

## Related Information

[Introduction to Application Development and Deployment \(XS Advanced Model\) \[page 10\]](#)

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

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

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

- Database developers

Often a business/data analyst or database expert, the database developer is concerned with the definition of the data model and schemas that will be used in SAP HANA XS advanced model, the specification and definition of tables, views, primary keys, indexes, partitions and other aspects of the layout and inter-relationship of the data in SAP HANA.

The database developer is also concerned with designing and defining authorization and access control, through the specification of privileges, roles, and users and, in addition, authorization scopes.

- Application programmers

The programmer is concerned with building SAP HANA XS advanced model applications: applications that are one part of so-called Multi-Target Applications (MTAs). Programmers develop the code for the business-logic component, for example, in JavaScript (Node.js or XS JavaScript) or Java.

- Client UI developers

The user-interface (UI) client developer designs and creates client applications which bind business logic (from the application developer) to controls, events, and views in the client application user interface. In this way, data exposed by the database developer can be viewed in the client application's UI.

### Caution

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

### Related Information

[SAP HANA Extended Application Services, Advanced Model \[page 11\]](#)

[SAP HANA XS Advanced Multi-Target Applications \[page 12\]](#)

[The XS Advanced Programming Model \[page 13\]](#)

[XS Advanced Application Development Tools \[page 21\]](#)

[Developer Information Map for XS Advanced \[page 23\]](#)

## 2.1 SAP HANA Extended Application Services, Advanced Model

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

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

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

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

### XS Advanced Application Development Scenarios

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

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

### Related Information

[Introduction to Application Development and Deployment \(XS Advanced Model\) \[page 10\]](#)

## 2.2 SAP HANA XS Advanced Multi-Target Applications

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

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

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

### ➔ Tip

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

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

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

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

### ➔ Tip

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

## Application Deployment

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

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

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

## Related Information

[The XS Advanced Programming Model \[page 13\]](#)

[SAP HANA Extended Application Services, Advanced Model \[page 11\]](#)

[XS CLI: Plug-ins \[page 939\]](#)

## 2.3 The XS Advanced Programming Model

Writing applications for deployment to SAP HANA XS advanced.

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

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

- [Cloud Foundry Concepts \[page 14\]](#)
- [System Architecture \[page 16\]](#)
- [Run-Time Platform \[page 17\]](#)
- [Authentication and Authorization \[page 17\]](#)

- [Component Model \[page 18\]](#)
- [Client User Interface \[page 18\]](#)
- [OData Services \[page 19\]](#)
- [SAP HANA Database \[page 19\]](#)

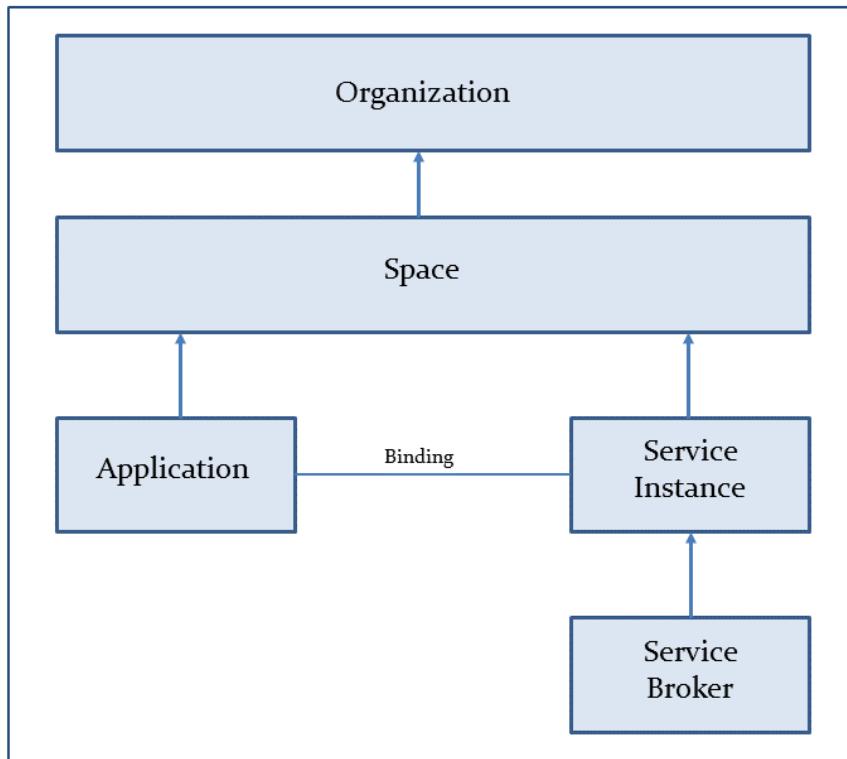
Includes information about the Service Broker, HDI containers, the HDI deployer, the SAP HANA Instance Broker, and database synonyms, as well as Core Data Services

## Cloud Foundry Concepts

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

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

Before a service can be used by an application, it is necessary to create a service “instance” to which the application must be bound. For each type of service there is a “service broker”; the service broker executes the actions required to create and delete service instances and bind application to services. The UAA service broker, for example, implements the API for managing UAA service instances. What a service instance means technically may differ between services and service plans. A service instance can be just a configuration for a shared implementation and a separate login and data area on a shared server. In certain other cases, each service instance can have its own server.



**Figure 1: Basic Cloud Foundry Concepts**

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

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

Applications are started as separate processes. At run time, the applications need the connection information for the services instances to which they are bound. The applications obtain this information from process-specific environment variables, which are resolved by the platform in a process known as “service wiring”. Bindings can only be created between applications and service instances in the same “space”. For scale-out scenarios, the platform can be instructed to start multiple instances of an application (several processes). XS advanced on-premise also implements the concepts of spaces, applications, services, buildpacks and droplets. Like Cloud Foundry applications, XS advanced applications are units that can be pushed, started, stopped, and bound to services. You can display a list of all the applications running in the current space with the `xs apps` command.

### **i** Note

Each XS advanced application runs in a specific run-time container.

## System Architecture

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

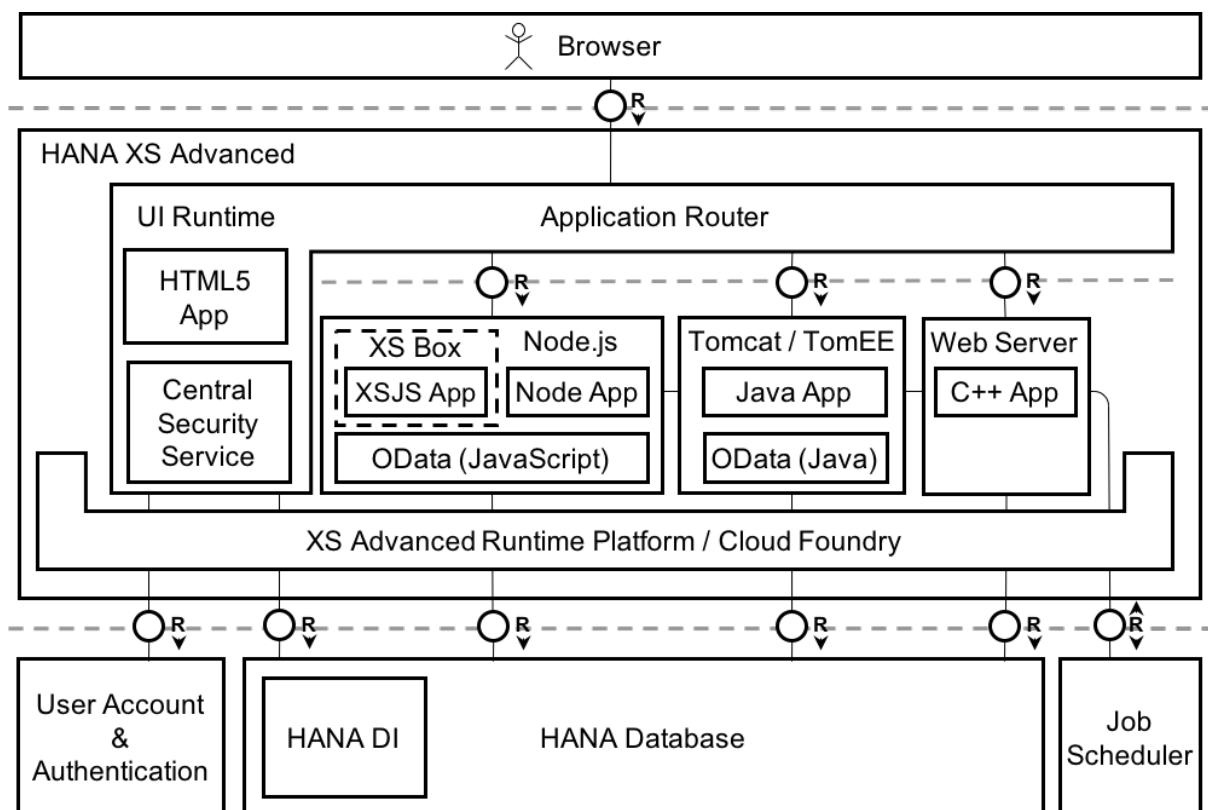


Figure 2: XS Advanced System Architecture

Typically, the following service layers are available:

- **Backing Services**  
Provide the technology on which XS advanced application are built, for example:
  - Persistence services (databases, key-value stores, ...)
  - Communication services ( message queues, email gateways, ...)
  - Authentication services (User Account and Authentication service (UAA))

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

## Run-Time Platform

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

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

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

## Authentication and Authorization

When a user sends an unauthenticated request to the central entry point of an XS advanced application such as the XS advanced Application Router (or any other mashup service), the user is redirected to the XS advanced User Account and Authentication (UAA) service. The UAA can handle different types of authentication methods, for example: basic (user name and password), SAML assertions, and X.509 certificates. The UAA does not store user credentials itself; relies on an external user store, for example: SAP Cloud Identity, an LDAP server or also the HANA database. Upon successful login, the UAA issues an OAuth token that can then be forwarded in all calls to further application services to authenticate that user. This way, application services only need to support one single authentication method: OAuth tokens. By contrast, authentication at backing services like databases is based on technical users that are authenticated with user credentials (name and password) provided to the application in its environment. The user's OAuth token can also be forwarded to a backing service such as the SAP HANA database for user-based authorizations.

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

## Component Model

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

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

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

## Client User Interface

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

The same reasons (of simplicity, implementation details, and central login services) also apply when deciding whether to use Cross-Origin Resource Sharing (CORS) in the browser as an alternative solution to the same-origin policy restriction. Fiori applications can be served by a central XS advanced Application Router that reads UI content and routing configuration from the layered repository (LRep) backing service. As a result, Fiori applications do not need to deploy a dedicated XS advanced Application Router; they just have to deploy their Fiori content to the shared LRep. By contrast, plain (non-Fiori) UI5 applications embed their UI content in

their own XS advanced Application Router which also acts as a Web server for content packaged with its own container; these UI5 applications do not rely on a backing service as an external content repository.

## OData Services

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

## SAP HANA Database

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

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

### SAP HANA Service Broker

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

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

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

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

## HDI Container

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

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

## HDI Deployer

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

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

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

## SAP HANA Instance Broker

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

## Synonyms

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

However, from a performance perspective and also from a data-consistency perspective, replication is not always the best answer. So, the SAP HANA database provides synonyms that enable an XS advanced application to perform a controlled “break out” from its own container to related containers. A database synonym is similar to a symbolic link; from an application's perspective, the access always happens within its own container, but the synonym may link to an object in another container. A container that exposes objects to other containers must explicitly grant access to consuming containers in order to create synonyms. The container exposing objects with synonyms has control over the set of objects which can be accessed. As a result, the exposing container has the means to define an application programming interface (API) at the database level, for example, by white-listing only public objects in a database role that is granted to the consuming container.

### **i** Note

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

## Core Data Services

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

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

## Related Information

[SAP HANA Extended Application Services, Advanced Model \[page 11\]](#)

[The XS Advanced Programming Model \[page 13\]](#)

## 2.4 XS Advanced Application Development Tools

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

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

- [Development Tools \[page 22\]](#)
- [Administration Tools \[page 23\]](#)

## XS Advanced Development Tools

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

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

## Gerrit for XS Advanced

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

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

### ➔ Tip

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

## Database Explorer

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

- Database browser  
Browse, view, run, and visualize the content of all types of catalog objects, for example: tables, views, stored procedures, functions, synonyms, and so on.
- SQL Console  
Create SQLScript, run SQL commands, visualize objects in text form (procedures, functions, and so on).

- SQL Debugger  
Debug procedures and views in HDI containers. View the call stack, set break points, view and evaluate expressions and variables, and so on.
- Job Log Viewer  
View the logs written by the XS jobs.

## XS Advanced Administration Tools

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

- *Application Monitor*  
Monitor the system usage of the applications running in the XS Advanced Model run-time
- *Organization and Space Management*  
Create, list, or delete user organizations and spaces in the XS Advanced Model run time.
- *Application Role Builder*  
Maintain and manage user roles and role collections in SAP HANA.
- *SAML Identity Providers Configuration*  
Configure SAML Identity providers (IDP) for SAP HANA XS advanced model applications that use SAML assertions as the logon authentication method.
- *User Management*  
Create and manage business users for SAP HANA XS advanced model applications.
- *SAP HANA Logical Database Configuration*  
Manage SAP HANA database instances for SAP HANA XS advanced model applications.
- *SAP HANA Service Brokers*  
Manage and monitor the SAP HANA service broker used by SAP HANA XS advanced model applications.
- *Job Scheduler Service Dashboard*  
Create, schedule, and manage long running operations jobs in the SAP HANA XS advanced model run-time environment.

## Related Information

[Introduction to Application Development and Deployment \(XS Advanced Model\) \[page 10\]](#)

[Developer Information Map for XS Advanced \[page 23\]](#)

## 2.5 Developer Information Map for XS Advanced

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

The development environment for SAP HANA supports a wide variety of application-development scenarios. For example, database developers need to be able to build a persistence model or design an analytic model;

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

➔ Tip

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

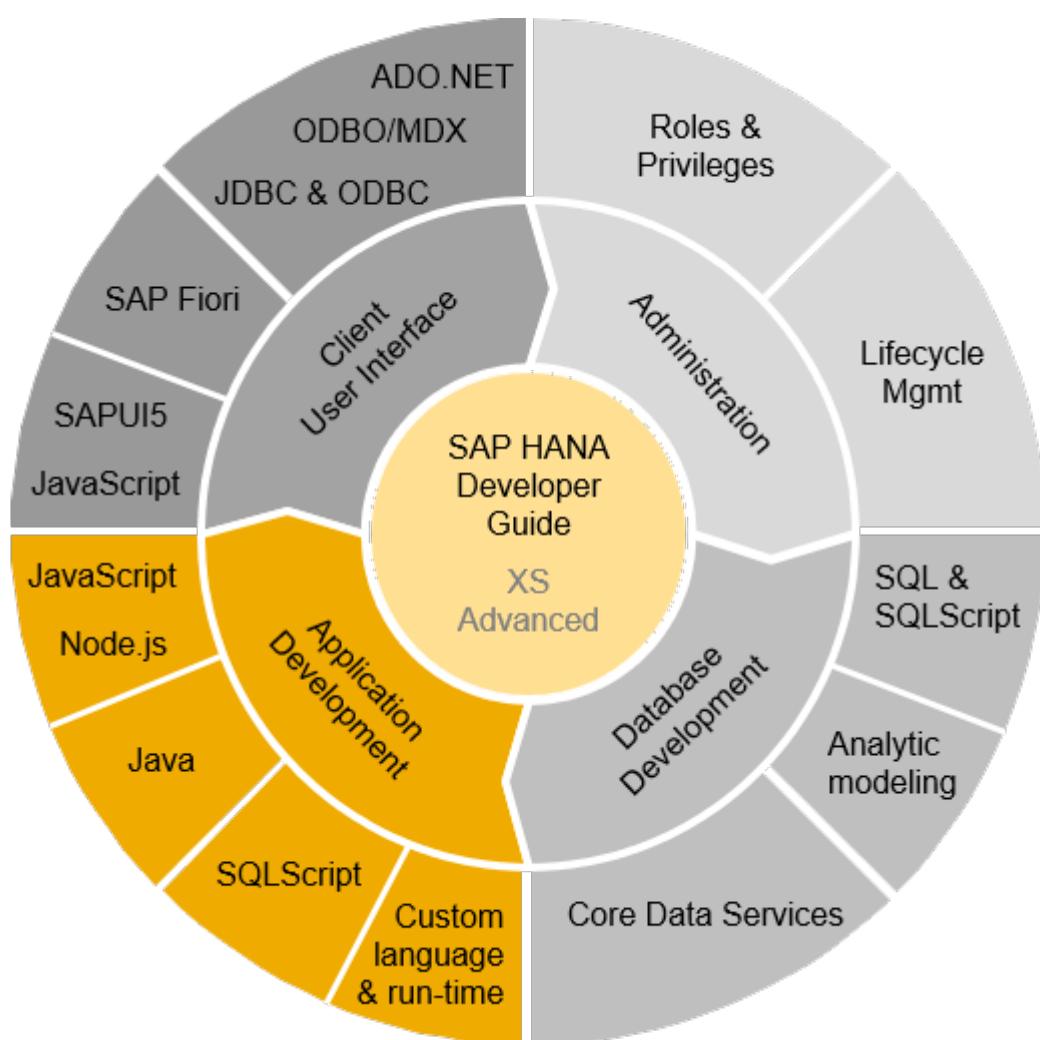


Figure 3: Application Development in SAP HANA XS Advanced Model

# 3 Getting Started with Application Development in XS Advanced

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

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

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

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

## i Note

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

- `db/`  
Contains only database artifact, for example: tables, views stored procedures, etc.
- `web/`  
Static HTML resources
- `java/`  
Java source code and any Odata service definitions
- `javascript/, js/, or xsjs/`  
JavaScript source code and any Odata service definitions; this can include either JavaScript or node.js (or both) in sub-folders

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

## Example

### Sample Code

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

## Related Information

[Set up an XS Advanced Application Project \[page 104\]](#)

[XS Advanced Application Resource Files \[page 110\]](#)

[Working with the XS Advanced Command-Line Client \[page 78\]](#)

[Working with the SAP Web IDE for SAP HANA \[page 26\]](#)

## 3.1 Working with the SAP Web IDE for SAP HANA

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

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

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

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

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

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

### ➔ Tip

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

To enable access to the development and administration tools included in SAP Web IDE for SAP HANA, some administration setup tasks are necessary after installation of SAP Web IDE for SAP HANA. For more information, see *Related Information* below.

2. Set up an application project.
3. Develop your application modules.

For example, for your data model, your application business logic, and your client user interface.

- 
4. Build your application modules.
  5. Run and debug your application modules.
  6. Package the modules into a multi-target application.
  7. Deploy the application to SAP HANA XS advanced.

## Related Information

[Post-Installation Administration Tasks \[page 962\]](#)

[Create a Simple "Tiny-World" Application \[page 27\]](#)

[Extend the "Tiny World" Application to Use an OData Service and a Client UI \[page 33\]](#)

[Working with the SAP HANA Database Explorer \[page 70\]](#)

[SAP Web IDE for SAP HANA Reference \[page 959\]](#)

### 3.1.1 Create a Simple "Tiny-World" Application

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

## Prerequisites

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

#### Caution

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

## Context

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

## Procedure

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

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

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

Choose the project template type **Multi-Target Application Project** and choose **Next**.

- c. Type the name **tinyworld** for the new MTA project and choose **Next** to confirm.
- d. Specify details of the new MTA project and choose **Next** to confirm.
- e. Create the new MTA project; choose **Finish**.

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

### Tip

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

### Sample Code

MTA Development Descriptor (`mta.yaml`)

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

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

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

- a. Create a database module.

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

- b. Provide a name for the new database module; type **tinydb**
- c. Customize the details of the new database module, if required. In this case, you can accept the default settings and choose **Next** to confirm your settings.
- d. Confirm the details of the new module; choose **Finish**.

The Wizard creates the new database module with the specified name: `tinydb`. The new module contains the folder `src` for your database artifacts and, in addition, two configuration files: `.hdiconfig` and `.hdinamespace`. The `.hdiconfig` file is mandatory and defines the plug-ins that SAP HANA uses to

create a catalog object from a design-time resource; the `.hdinamespace` is optional and defines the rules to use for run-time name spaces.

### Note

By default, the `.hdiconfig` and `.hdinamespace` configuration files are hidden. To display hidden files in the SAP Web IDE for SAP HANA, choose .

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

### Sample Code

Application Deployment Descriptor `mta.yaml`

```
schema-version: 2.0.0
ID: tinyworld
description: Tiny world tutorial project
version: 0.0.1
modules:
- name: tinydb
  type: hdb
  path: tinydb
  requires:
  - name: hdi-container
resources:
- name: hdi-container
  type: com.sap.xs.hdi-container
```

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

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

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

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

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

### Sample Code

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

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

5. Build the HDB module `tinydb`.

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

- a. In the SAP Web IDE for SAP HANA, choose ► **Build** ► **Build** ▾.

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

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

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

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

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

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

#### → Tip

Check the *Enable xsjs support* check box.

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

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

#### Sample Code

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

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

#### Sample Code

Application Deployment Descriptor `mta.yaml`

```
_schema-version: 2.0.0
ID: tinyworld
description: Tiny world tutorial project
version: 0.0.1
modules:
- name: tinydb
  type: hdb
  path: tinydb
  requires:
  - name: hdi-container
- name: tinyjs
  type: nodejs
  path: tinyjs
resources:
- name: hdi-container
```

```
type: com.sap.xs.hdi-container
```

7. Build the new Node.js application module.

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

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

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

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

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

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

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

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

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

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

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

#### Sample Code

```
<!DOCTYPE html>
<html>
  <body> Tiny HTML World!
  </body>
</html>
```

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

#### Sample Code

##### Application Deployment Descriptor `mta.yaml`

```
schema-version: 2.0.0
ID: tinyworld
description: Tiny world tutorial project
version: 0.0.1
modules:
- name: tinydb
  type: hdb
  path: tinydb
  requires:
  - name: hdi-container
  - name: tinyjs
```

```
type: nodejs
path: tinyjs
- name: tinyui
  type: html5
  path: tinyui
resources:
- name: hdi-container
  type: com.sap.xs.hdi-container
```

9. Run the new HTML5 application module.

Right-click the module folder `tinyui/` and, in the context menu, choose .

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

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

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

## Results

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

## Related Information

[Working with the XS Advanced Command-Line Client \[page 78\]](#)

[Extend the "Tiny World" Application to Use an OData Service and a Client UI \[page 33\]](#)

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

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

### Prerequisites

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

#### Caution

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

### Context

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

### Procedure

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

#### Sample Code

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

```
};
```

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

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

#### Sample Code

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

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

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

#### Sample Code

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

6. Build the HDB module `tinydb`.

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

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

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

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

- Open the database explore, by choosing *Tools* *Database Explorer*.
- If prompted, choose to add a database to the database explorer. Otherwise, at the top of the database browser, click *Add database to the Database Explorer*.
- In the *Add Database* window, choose *HDI Container* for the *Database Type*.
- In the list of HDI containers, choose the HDI container in which the tables were created.

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

Your HDI container is added to the database browser.

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

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

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

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

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

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

- a. Create the new OData service.

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

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

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

#### Source Code

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

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

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

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

#### Sample Code

```
- name: tinyjs
  type: nodejs
  path: tinyjs/
```

```

requires:
- name: tinydb
- name: hdi-container
provides:
- name: tinyjs_api
properties:
  service_url: ${default-url}

```

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

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

#### Sample Code

```
// configure HANA
options = xsjs.extend(options, xsenv.getServices({ hana: {tag:
"hana"} }));
```

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

Right-click the module `tinyjs/` and choose   in the context menu.

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

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

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

- Test the OData service is available.

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

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

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

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

#### Output Code

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

```

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

```

## 10. Create a simple SAPUI5 application that uses the OData service.

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

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

### Sample Code

```

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

```

The code snippet is designed to do the following actions:

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

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

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

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

### Sample Code

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

- b. Add the following URL destination to the `tinyui` module of the `mta.yaml` file.

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

### Sample Code

```
{
  "source": "^/euro.xsodata/.*$/,
  "destination": "tinyjs_be"
}
```

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

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

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

## Related Information

[Working with the SAP Web IDE for SAP HANA \[page 26\]](#)

[Create a Simple "Tiny-World" Application \[page 27\]](#)

[Add Business Logic to the "Tiny World" Application \[page 38\]](#)

## 3.1.3 Add Business Logic to the "Tiny World" Application

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

## Prerequisites

- You have access to a running SAP HANA system where the XS advanced run time is installed and configured.
- You have access to the XS advanced command-line client.
- You have access to the developer tools included in SAP Web IDE for SAP HANA.

If SAP Web IDE for SAP HANA is not available at the default URL `https://host.acme.com:53075`, log in to the XS advanced run time and use the command `xs app webide` to display the URL required to access the tool.

- You have successfully completed the instructions described in the following tutorials:
  - *Create a Simple “Tiny World” Application in XS Advanced*
  - *Extend the “Tiny World” Application to use an OData Service and Client UI*

### Caution

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

## Context

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

## Procedure

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

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

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

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

An SQL editor opens for the new stored procedure.

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

### Sample Code

```
PROCEDURE "tinyworld.tinydb::createCountry" (
    IN im_country NVARCHAR(100), im_continent NVARCHAR(100),
    OUT ex_error NVARCHAR(100))
LANGUAGE SQLSCRIPT SQL
SECURITY INVOKER AS
--READS SQL DATA AS
BEGIN
declare noc integer;
```

```

select count(*) into noc
  from "tinyworld.tinydb::tinyf.world"
  where "continent" = im_continent;
if :noc = 0 then
  ex_error := 'ERROR: Continent ' || :im_continent || ' does not
exist!';
else
  select count(*) into noc
    from "tinyworld.tinydb::tinyf.country"
    where "name" = im_country;
  if :noc > 0 then
    ex_error := 'ERROR: Country ' || :im_country || ' already
exists!';
  else
    insert into "tinyworld.tinydb::tinyf.country"
      values (im_country, im_continent);
  end if;
end if;
END;

```

- Build the application database module `tinydb` again.

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

### Output Code

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

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

- Choose **Tools** **Database Explorer**.

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

- Use the database browser to find the procedure `tinyworld.tinydb::createCountry` in the HDI container.
- Right-click the `tinyworld.tinydb::createCountry` procedure and choose **Call Procedure and Prompt for values**.
- When prompted add a country that exists in the database.

Specify **Spain** in the **IM\_COUNTRY** field and **Europe** in the **IM\_CONTINENT** field, and then choose **Run** from the global tool bar (or press **F8**).

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

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

Specify **Ireland** in the **IM\_COUNTRY** field, and **Europe** in the **IM\_CONTINENT** box, and then choose **Run** from the global tool bar (or press **F8**).

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

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

There is now a row in the table for Ireland.

4. In the development perspective, create a new JavaScript called "country.xsjs" and place it in the JavaScript application folder tinyworld/tinyjs/lib/country..

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

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

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

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

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

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

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

### Note

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

### Sample Code

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

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

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

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

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

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

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

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

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

In your Web browser, replace the URL element “`/index.xsjs`” with the string “`/country/country.xsjs?name=China&continent=Asia`”; the resulting URL should look like the following example:

```
https://host.acme.com:51012/country/country.xsjs?name=China&continent=Asia  
https://host.acme.com:51012/country/country.xsjs?name=Albania&continent=Europe  
https://host.acme.com:51012/country/country.xsjs?name=Sweden&continent=Europe
```

The output should look like the following example:

The screenshot shows the SAP Web IDE's "Output Code" panel. It contains three JSON objects, each representing a country with its name and continent. The objects are separated by horizontal lines.

```
{  
  "name": "China",  
  "partof": "Asia"  
}  
  
{  
  "name": "Albania",  
  "partof": "Europe"  
}  
  
{  
  "name": "Sweden",  
  "partof": "Europe"  
}
```

### 3.1.3.1 Debug the JavaScript Code in the "Tiny World" Application

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

#### Prerequisites

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

#### Tip

Alternatively, you can configure NPM to resolve SAP-related NPM dependencies using the SAP public NPM Registry (at [npm.sap.com](https://npm.sap.com)).

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

#### Caution

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

#### Context

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

## Procedure

1. Choose in the tool bar for the Node.js module `tinyjs` in the "Tiny World" application.
2. Choose in the tool bar, to display the built-in JavaScript debugging tools.
3. Choose in the pane and select your running Node.js application in the dialog.
4. Set a break point to test the debug mode.

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

5. Test the JavaScript code.

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

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

### Note

The debugger displays a notification indicating that the "tinyui" application is suspended at the break point you just set.

6. Debug the JavaScript code.

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

7. Resume the debugging process.

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

### Note

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

## Related Information

[The SAP HANA XS Advanced JavaScript Run Time \[page 510\]](#)

[Run Unit Tests on the "Tiny World" Application's JavaScript Code \[page 45\]](#)

[Add Business Logic to the "Tiny World" Application \[page 38\]](#)

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

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

#### Prerequisites

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

#### Tip

Alternatively, you can configure NPM to resolve SAP-related NPM dependencies using the SAP public NPM Registry (at [npm.sap.com](https://npm.sap.com)).

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

#### Caution

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

#### Context

SAP HANA XS classic supports a Jasmine-like unit-test pattern called "XSUnit", which can also be used in XS advanced to develop unit tests for a Node.js application. When you create a Node.js module for your multi-target application (MTA), the SAP Web IDE for SAP HANA creates a new folder named "test/", which

includes a sample test library named “`sampleTest.xsjslib`”. You can use this sample test library to run some simple tests on the Node.js files in the “Tiny World” application.

## Procedure

1. Create a new test run-configuration for your Node.js application module `tinyjs`.
  - a. Create a new test run-configuration.  
Choose  *Run > Run Configurations* in the tool bar.
  - b. In the *Run Configurations* dialog, choose `[+]`, select *Node.js Test* in the drop-down list.
  - c. Type the name `test` and choose *Save and Run*.

XSUnit runs your Node.js module `tinyjs` in XS JavaScript test mode. The test finds the `sampleTest.xsjslib` file because it matches the test file pattern “\*`test`” defined in the run configuration you created. The *Node.js Test Execution* pane displays the results of the test.

2. Add a more meaningful function to the unit test.

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

### Sample Code

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

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

3. Add the calculator test code.

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

### Sample Code

```
tinyjs/lib/calc

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

4. Add more tests as required.

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

#### Note

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

### 3.1.4 Deploy the "Tiny World" Application

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

#### Prerequisites

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

#### Tip

Alternatively, you can configure NPM to resolve SAP-related NPM dependencies using the SAP public NPM Registry (at [npm.sap.com](https://npm.sap.com)).

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

## Caution

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

## Context

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

## Procedure

### 1. Create the MTA archive.

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

- o *tinydb*

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

12:44:25 PM (Builder) Build of /tinyworld/tinydb completed successfully.

- o *tinyjs*

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

12:44:37 PM (Builder) Build of /tinyworld/tinyjs started.

12:44:39 PM (DIBuild) Build of /tinyworld/tinyjs in progress

12:45:25 PM (Builder) Build of /tinyworld/tinyjs completed successfully

- o *tinyui*

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

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

#### b. Specify the application **version**.

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

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

### Sample Code

```
ID: tinyworld
description: Tiny world tutorial
version: 1.0.0
```

[ ... ]

- c. Build the complete “tinyworld” application project.

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

### Output Code

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

If the build process completes successfully, an MTA archive named *tinyworld\_1.0.0.mtar* is created in the root folder of the application project (*mta\_archives/tinyworld/*).

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

In the project navigation pane, right-click the MTA archive *tinyworld\_1.0.0.mtar* and choose ► *Export* ▶ in the context menu.

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

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

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

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

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

### Note

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

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

### Output Code

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

Space: PROD

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

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

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

→ Tip

If necessary, grant RITA the required permissions:

```
xs set-space-role RITA myorg PROD SpaceDeveloper
```

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

```
xs target
```

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

→ Tip

If necessary, switch to the target space for deployment (“PROD”):

```
xs target -o myorg -s PROD
```

- d. Deploy the MTA archive to the target space.

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

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

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

 Output Code

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

### ➔ Tip

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

3. Test the deployed application in a Web browser.

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

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

The application starts in the Web browser; you can log on using any valid database user, for example, "RITA".

- b. Display details of the deployed "tinyui" application running in the current space.

In the command shell, run the following command:

`xs app tinyworld.tinyui`

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

- c. Paste the URL displayed into your Web browser.

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

The Web browser displays the database table "European Countries" table.

### ℹ Note

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

## Related Information

[The XS Command-Line Interface Reference \[page 848\]](#)

[Add Business Logic to the "Tiny World" Application \[page 38\]](#)

[Upgrade and Redeploy the "Tiny World" Application \[page 52\]](#)

## 3.1.5 Upgrade and Redeploy the "Tiny World" Application

### Prerequisites

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

#### Tip

Alternatively, you can configure NPM to resolve SAP-related NPM dependencies using the SAP public NPM Registry (at [npm.sap.com](https://npm.sap.com)).

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

#### Caution

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

### Context

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

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

## Procedure

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

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

### Sample Code

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

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

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

### Note

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

2. Create a new JavaScript resource named `Util.js` in the folder `tinyui/resources/` folder of the "tinyworld" application project.
  - a. Right-click the folder `tinyui/resources/` and choose   from the context menu.
  - b. Type the name **Util.js**.
  - c. Paste the following code into the new JavaScript `Util.js`.

## → Tip

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

## Sample Code

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

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

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

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

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

#### Sample Code

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

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

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

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

#### Sample Code

```
function saveCountry(country) {  
    var conn = $.hdb.getConnection();  
    var output = JSON.stringify(country);  
    var fnCreateCountry =  
conn.loadProcedure("tinyworld.tinydb:createCountry");  
    var result = fnCreateCountry({IM_COUNTRY: country.name, IM_CONTINENT:  
country.partof});  
    conn.commit();  
    conn.close();  
    if (result && result.EX_ERROR != null) {  
        return {body : result,  
                status: $.net.http.BAD_REQUEST};  
  
    } else {  
        return {body : output,  
                status: $.net.http.CREATED};  
    }  
}  
var body = $.request.bodyasString();  
var country = JSON.parse(body);  
// validate the inputs here!  
var output = saveCountry(country);  
$.response.contentType = "application/json";  
$.response.setBody(output.body);  
$.response.status = output.status;
```

5. Rebuild the application modules.

- Build the database application module `tinydb/`.

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

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

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

- c. Run the UI5 application module `tinyui`.
- Right-click the module `tinyui` and choose ► **Run** ► **Run as** ► **Web Application** ► .
6. Start the application in your Web browser.

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

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

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

7. Test the new `Add Country` feature.

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

The country “Ireland” appears in the database table `European Countries`.

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

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

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

#### Sample Code

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

9. Create the MTA archive.
  - a. Build the individual application modules in the following order:
    - o `tinydb`  
In the project navigation pane, right-click the module `tinydb` and choose ► **Build** ► in the context menu.
    - o `tinyjs`  
In the project navigation pane, right-click the module `tinyjs` and choose ► **Build** ► in the context menu.
    - o `tinyui`  
In the project navigation pane, right-click the module `tinyui` and choose ► **Build** ► in the context menu.
  - b. Build the complete “tinyworld” application project.

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

## Output Code

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

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

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

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

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

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

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

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

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

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

### Tip

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

- b. Deploy the MTA archive to the target space.

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

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

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

## Output Code

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

```
Process finished
```

→ Tip

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

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

In the command shell, run the following command:

```
xs apps
```

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

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

In the command shell, run the following command:

```
xs app RITA-j5d0r0x3ukczly6h-tinyworld-tinyui --urls
```

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

- c. Paste the URL displayed into a Web browser.

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

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

## Related Information

[Deploy the “Tiny World” Application \[page 47\]](#)

[Add User Authentication to the “Tiny World” Application \[page 59\]](#)

[The XS Command-Line Interface Reference \[page 848\]](#)

## 3.1.6 Add User Authentication to the "Tiny World" Application

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

### Prerequisites

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

#### Caution

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

### Context

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

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

## Procedure

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

### Note

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

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

```
// configure UAA
options = xsjs.extend(options, xsenv.getServices({ uaa: {tag: "xsuaa"} }));
```

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

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

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

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

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

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

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

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

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

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

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

### Sample Code

```
$(function() {
```

```

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

```

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

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

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

#### Sample Code

```

resources:
- name: hdi-container
  type: com.sap.xs.hdi-container
- name: tiny_uaa
  type: com.sap.xs.uaa

```

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

#### Note

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

#### Sample Code

```

- name: tinyjs
  type: nodejs
  path: tinyjs/
  requires:
    - name: tiny_uaa
    - name: tinydb
    - name: hdi-container
  provides:
    - name: tinyjs_api
      properties:
        service_url: ${default-url}

```

```
- name: tinyui
  type: html5
  path: tinyui/
  requires: # ----- use JS module URL as destination
    - name: tiny_uaa
    - name: tinyjs_api
      group: destinations
      properties:
        name: tinyjs_url
        url: ~{service_url}
```

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

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

#### Sample Code

```
- name: tinyui
  type: html5
  path: tinyui/
  requires: # ----- use JS module URL as destination
    - name: tiny_uaa
    - name: tinyjs_api
      group: destinations
      properties:
        name: tinyjs_url
        url: ~{service_url}
        forwardAuthToken: true
```

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

- a. Rebuild the application module `tinyjs/`.

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

- b. Rebuild the application module `tinyui/`.

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

## Related Information

[Add Authorization Checks to the "Tiny World" Application \[page 63\]](#)

[Setting Up Security Artifacts \[page 757\]](#)

## 3.1.7 Add Authorization Checks to the "Tiny World" Application

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

### Prerequisites

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

#### ⚠ Caution

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

### Context

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

### Procedure

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

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

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

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

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

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

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

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

### Sample Code

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

### Note

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

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

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

### Sample Code

#### mta.yaml Resources Section

```
resources:  
- name: hdi-container
```

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

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

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

#### Note

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

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

```
xs delete-service tiny_uaa
```

- c. Create a new UAA service for the “Tiny World” application.

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

#### Note

The service plan “`devuser`” is designed for use at development time; it enables multiple developers to create UAA service instances with the same application name in their `xs-security.json` file. In XS advanced, application names must be unique; in the event of a name clash, the UAA service broker adds a suffix to the application name to make every application unique per user, for example, “`tinyworld!u1`”.

5. Add authorization checks to the application.

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

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

We extend the routes `/euro.xsodata`, `/euro.xsodata`, and `/euro.xsodata`, as illustrated in **bold text** in the following example:

#### Sample Code

`xs-app.json` Application Routes

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

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

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

#### Note

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

- c. Rebuild the application module `tinyjs/`.

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

- d. Rebuild the application module `tinyui/`.

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

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

#### Note

You use the application **SAP HANA XS Advanced Administration Tools** to create user roles and assign them to SAP HANA XS advanced users.

## Related Information

[Create and Assign User Roles for the “Tiny World” Application \[page 67\]](#)

[Add User Authentication to the “Tiny World” Application \[page 59\]](#)

[Create the Security Descriptor for Your XS Advanced Application \[page 763\]](#)

[Configure the XS Advanced Application Router \[page 717\]](#)

## 3.1.8 Create and Assign User Roles for the "Tiny World" Application

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

### Prerequisites

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

#### **Restriction**

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

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

#### **Restriction**

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

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

#### **Caution**

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

## Context

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

## Procedure

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

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

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

➔ Tip

To display the URL required to start the tool, log on to the XS advanced run time in a command shell and run the following command:

```
xs app xsa-admin --urls
```

- b. Start the *Application Role Builder* tool.
- c. Display details of the application for which you want to create roles.

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

2. Create role collections for the “Tiny World” application users.

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

- a. Start the *Role Collection* tool.
- b. Create a new role collection for viewing content.

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

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

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

- c. Create a new role collection for creating content.

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

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

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

3. Assign the role collections to the application users.

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

- a. Start the **Security** tool.

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

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

#### **Restriction**

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

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

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

- c. Assign the role collection to the selected user.

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

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

## Related Information

[Add Authorization Checks to the "Tiny World" Application \[page 63\]](#)

[Create the Security Descriptor for Your XS Advanced Application \[page 763\]](#)

## 3.1.9 Working with the SAP HANA Database Explorer

In the SAP Web IDE for SAP HANA, the SAP HANA database explorer provides a set of run-time tools that enable access to objects in HDI containers.

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

The database explorer includes the following tools:

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

You can start the database Explorer in the following ways:

- From the SAP HANA cockpit, choose *Database Explorer*.
- In the SAP Web IDE for SAP HANA, choose *Tools* *Database Explorer* .

### Related Information

[Add HDI Containers and Databases to the SAP HANA Database Explorer \[page 71\]](#)

[Maintaining HDI Containers \[page 173\]](#)

[SAP Web IDE for SAP HANA Reference \[page 959\]](#)

[SAP Note 2373065](#)

### 3.1.9.1 Add HDI Containers and Databases to the SAP HANA Database Explorer

Add a built HDI container to the SAP HANA database explorer in the SAP Web IDE for SAP HANA, so that you can browse its catalog objects and test its procedures and functions. You can also connect to an SAP HANA database.

#### Prerequisites

The HDI container must be configured on the same XS advanced server that the SAP Web IDE is running on.

To add an SAP HANA database to the database explorer, you must have a user ID and a password for that database.

You must be a user of the SAP Web IDE for SAP HANA.

#### Context

Once an HDI container or database is added to the database explorer, it is listed in the database browser pane on the left.

#### Procedure

1. Open the database explorer from the SAP Web IDE by choosing Tools Database Explorer.
2. Add a database by clicking the *Add Database* icon (+) from the database browser toolbar.
3. From the *Database Type* dropdown list, choose the type of database to add:

Database Type	Action
An HDI container	<ol style="list-style-type: none"><li>1. Choose <i>HDI container</i>.</li><li>2. Choose an HDI container from the list. The list contains all HDI containers that are known to the XS advanced server that the SAP Web IDE and database explorer use, as well as all user-defined HDI containers.</li><li>3. Set the XS_APPLICATIONUSER session variable to make the connection personalized for the current user.</li></ol>
An SAP HANA database	<ol style="list-style-type: none"><li>1. Choose <i>SAP HANA database</i>.</li><li>2. Specify the fully qualified domain name (FQDN) of the host on which the system is installed. Specify the instance number of the database you are adding. When adding a database that is part of a multi-host system, specify the master host. You do not have to enter all host names explicitly as they are determined automatically. If the master host becomes unavailable, then the connection is automatically established</li></ol>

Database Type	Action
	through one of the other hosts. Hosts that are added to the system later are also detected automatically.
<b>A tenant or the system database that is part of a multitenant database container system</b>	<ol style="list-style-type: none"> <li>Choose <i>SAP HANA database (Multitenant)</i>.</li> <li>Specify the host and instance number of the system database.</li> <li>Specify whether you are adding the system database or a tenant database. When adding a tenant database, specify the name of the tenant database.</li> </ol>
<b>HDI containers contained in instance managers</b>	<ol style="list-style-type: none"> <li>Choose <i>Application Managed Service Instances</i>.</li> <li>Specify the instance manager that contains the HDI container you want to add.</li> <li>Choose an HDI container from the list. The list contains all HDI containers that are known to the XS advanced server that the SAP Web IDE and the database explorer use, as well as all user-defined HDI containers.</li> <li>Set the XS_APPLICATIONUSER session variable to make the connection personalized for the current user.</li> </ol>

4. (Optional) Specify encryption information and advanced connection properties as required.

Choose from the following encryption options:

Table 1:

Option	Description
<i>Save user and password (stored in the SAP HANA secure store.)</i>	<p>By default, your database credentials are not saved. Each time you need to connect to an added database, you must provide your user credentials.</p> <p>Choose this option to save these user credentials so that you do not have to re-enter them each time you connect. These credentials are saved to the SAP HANA secure store.</p>
<i>Connect to the database securely using TLS/SSL. (Prevents data eavesdropping.)</i>	Choose this option to encrypt communication between the database explorer and the SAP HANA database using the Transport Security Layer (TLS)/Secure Sockets Layer (SSL) protocol.

Option	Description
<i>Verify the server's certificate using the trusted certificate below.</i>	<p>Choose the <i>Verify the server's certificate using the trusted certificate below</i> option and provide a trusted certificate, if you want to verify the server's certificate when connecting. This prevents server impersonation. The certificate field must contain the contents of a certificate, and not a file name. For more information, see the SAP HANA Security Guide.</p> <div style="background-color: #ffffcc; padding: 10px;"> <p><b>i Note</b></p> <p>You can only choose this option if you have also chosen the <i>Connect to the database securely using TLS/SSL. (Prevents data eavesdropping.)</i> option.</p> </div>

In the *Advanced Options* field, specify the advanced options as a comma separated, option-name=value pair. For example: `locale=en-US, isolationLevel=READ COMMITTED`.

Table 2:

Option	Description
autoCommit	Specify ON to have the database explorer implicitly commit your transaction after executing each DDL statement. Otherwise, specify OFF to have your transactions committed after executing a commit statement, and to support rollbacks of DDL statements. The default value is OFF.
isolationLevel	The isolation level for the connection. The supported values are: READ COMMITTED, REPEATABLE READ, and SERIALIZABLE. The default is to specify no isolation level.
locale	The locale to use for the connection. If you do not set this option, then the database explorer looks for a locale setting in your user parameter, and then in your browser. If no locale setting is found, then the locale is set to en-US.
schema	The name of the schema that you want to use
CLIENT	Sets the session client for the connection. The value is a three character string. For example, CLIENT=100.

- Click **OK**.

## Results

The HDI container or database is added to the database browser.

By default, you are connected to the HDI container as the technical user.

## 3.1.9.2 View Database Procedures in the SAP HANA Database Explorer

Use the SAP HANA database explorer to view and test a database procedure.

### Prerequisites

- You have access to a running SAP HANA system that includes a database procedure.

### Procedure

1. Open the database explorer from either the SAP HANA cockpit or the SAP Web IDE for SAP HANA.
2. If your HDI container or cockpit resource is not listed in the database browser, add it by choosing *Add a database to the Database Explorer* from the database browser toolbar.
3. Browse to your procedure, and then choose one of the following options:

Option	Action
<b>View information about the procedure's parameters</b>	Click the procedure to open.
<b>View the CREATE statement associated with the procedure</b>	Open the procedure, and then click the <i>CREATE Statement</i> tab.
<b>Generate a CALL statement for the procedure</b>	Open the procedure, and then click <i>Generate CALL Statement</i> .
<b>Open the procedure to debug it (Web IDE only)</b>	Right-click the procedure, and then choose <i>Open for debugging</i> .

### Related Information

[Working with the SAP HANA Database Explorer \[page 70\]](#)

[Create a Simple "Tiny-World" Application \[page 27\]](#)

### 3.1.9.3 Debug Procedures in the SAP HANA Database Explorer

Debug your stored procedures in HDI containers by using the SQL debugger in the SAP HANA database explorer.

#### Prerequisites

You must have the DEBUG object privilege granted to the schema or the procedures, that you want to debug.

You must be a user of the SAP Web IDE for SAP HANA.

#### Context

In the Web IDE, use the development perspective to create stored procedures in .hdbprocedure modules and build them into HDI containers. Switch to the database explorer to test and debug the built procedures. To make corrections to the stored procedures, return to the development perspective. Repeat this process of editing the stored procedures in the development perspective and debugging them in the database explorer until you are satisfied.

You can only use the SQL debugger with HDI containers. Using the SQL debugger with user-provided services or application-managed service instances is not supported.

#### Procedure

1. Choose one of the following options to open the procedure for debugging:

Option	Action
<b>From the development perspective:</b>	<ol style="list-style-type: none"><li>1. Open the stored procedure (the .hdbprocedure file.)</li><li>2. Right-click in the editor and choose <i>Open Runtime Object</i>.</li></ol> <p>The database explorer opens, the HDI container appears in the database browser, the procedure opens in an editor, and the SQL debugger pane opens on the right.</p> <p>If the SQL debugger pane doesn't open, then click <i>SQL Debugger</i> on the right sidebar.</p>
<b>From the database explorer:</b>	<ol style="list-style-type: none"><li>1. Connect to the HDI container.</li></ol>

Option	Action
	<p>2. Locate the stored procedure in the database browser, and then right-click and choose <a href="#">Open for debugging</a>.</p> <p>The procedure opens in an editor and the SQL debugger pane opens on the right.</p> <p>If the SQL debugger pane doesn't open, then click <a href="#">SQL Debugger</a> on the right sidebar.</p>

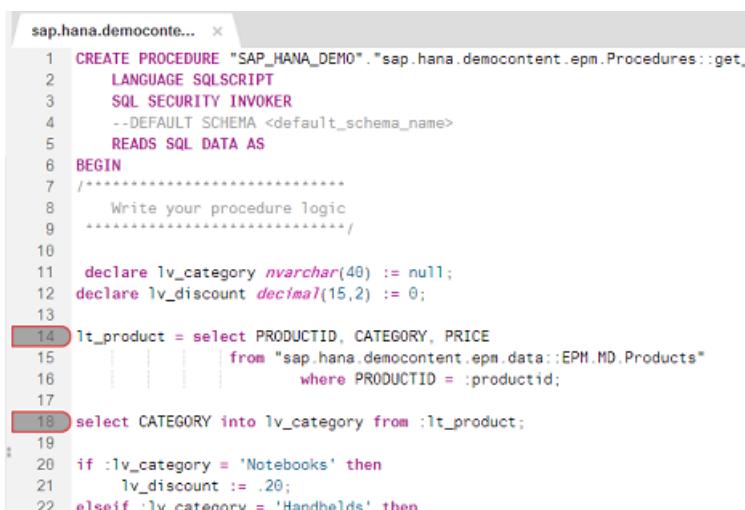
2. Connect the SQL debugger to the HDI container that contains the procedures that you want to debug.
  - a. From the SQL debugger toolbar, choose [Settings](#) to open the [SQL Debugger Settings](#) window.
  - b. Click in the [Service Name](#) field and press F4 to open the [Search Available Services](#) window. Choose your HDI container from the list.

If the [Service Name](#) field is disabled, then choose [Toggle Connection](#) from the [URL](#) field to disable the current connection, and then repeat this step.

  - c. From the [URL](#) field, click [Toggle Connection](#) to create a connection to the service for debugging.
  - d. Specify other debugger settings as required.
  - e. Click [Apply](#). Click OK if prompted.
  - f. Click [Close](#).

A connection to the SQL debugger is established.

3. Set breakpoints in the code of your stored procedures.
  - a. Set a breakpoint in front of a code line by clicking the line number, as shown in the example below:



```

sap.hana.democonte... ×
1 CREATE PROCEDURE "SAP_HANA_DEMO"."sap.hana.democontent.epm.Procedures::get_
2   LANGUAGE SQLSCRIPT
3   SQL SECURITY INVOKER
4   --DEFAULT SCHEMA <default_schema_name>
5   READS SQL DATA AS
6 BEGIN
7   *****
8   Write your procedure logic
9   *****
10
11 declare lv_category nvarchar(40) := null;
12 declare lv_discount decimal(15,2) := 0;
13
14 1t_product = select PRODUCTID, CATEGORY, PRICE
15  |           | from "sap.hana.democontent.epm.data::EPM.MD.Products"
16  |           | where PRODUCTID = :productid;
17
18 select CATEGORY into lv_category from :1t_product;
19
20 if :lv_category = 'Notebooks' then
21   lv_discount := .20;
22 elseif :lv_category = 'Handhelds' then

```

You can see a list of all of the breakpoints in the SQL debugger.

4. Call the procedure or function
  - a. In the global toolbar, choose  ([Invoke Procedure](#)).
 

The SQL editor opens and it contains the CALL statement for your procedure.
  - b. In the code, enter values for any input parameters.
  - c. Choose  ([Run](#)) from the global toolbar.

The debug session begins and the SQL debugger opens on the right, showing the status of the session. The debugger stops at the first breakpoint and the session is suspended until you resume the debugging.

5. Choose  (Resume) or  (Step over) to step through the code execution.

## Next Steps

Once you are finished debugging, click [Toggle Connection](#) to stop the debugging session.

## Related Information

[Add User Authentication to the "Tiny World" Application \[page 59\]](#)

### 3.1.9.4 Execute MDX Queries in the SAP HANA Database Explorer

Execute MDX queries on your databases by using the MDX console included with the SAP HANA database explorer.

## Prerequisites

You must have the required privileges in the SAP HANA database to execute your MDX statements.

## Context

The database explorer includes an MDX console that does the following:

- Executes batches of statements, separated by semicolons.
- Includes a code completion feature, as well as a code formatting feature. Right-click within the console to use these features.
- Includes a code verification feature.

For more information about MDX, see the *SAP HANA Developer Guide* for SAP HANA studio.

## Procedure

1. From the database browser choose the database that you want to run your MDX query against. You must be connected to the database.
2. From the global toolbar, click the *MDX console* icon.
3. Specify an MDX query.
4. Choose *Run* from the global toolbar to execute the query.

Option	Action
<b>Execute the query.</b>	Choose <i>Run</i> from the global toolbar.
<b>Verify that your MDX statements are syntactically and semantically correct before you try to run them.</b>	From the <i>Run</i> dropdown list in the global toolbar, choose <i>Verify</i> .

## 3.2 Working with the XS Advanced Command-Line Client

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

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

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

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

## Related Information

[SAP Note 2242468: Setting up the XS Advanced CLI](#)

[Get Started with the XS CLI Client \[page 79\]](#)

[Deploy a Node.js Hello-World Application using the XS CLI \[page 84\]](#)

[Deploy a Node.js Application that Uses Database Artifacts and OData Services \[page 89\]](#)

## 3.2.1 Get Started with the XS CLI Client

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

### Prerequisites

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

➔ Tip

Alternatively, you can configure NPM to resolve SAP-related NPM dependencies using the SAP public NPM Registry (at [npm.sap.com](http://npm.sap.com)).

- You have a Git client installed on your development machine.

➔ Tip

The SAP Web IDE for SAP HANA includes an integrated Git client.

- You have access to the Internet from your development machine.
- Logon access to the SAP HANA database with the privileges to create SAP HANA users (for example, SYSTEM)

### Context

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

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

## Procedure

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

Enter the following URL in your Web browser:

```
https://<hana_hostname>:3<instance>30/v2/info
```

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

```
https://xsa.acme.com:30030/v2/info
```

The response displayed in the Web browser is a JSON string with details that indicate a successful connection to the XS advanced controller.

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

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

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

```
xs help
```

```
xs -v
```

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

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

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

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

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

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

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

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

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

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

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

4. Log on to the XS advanced run time.

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

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

 Note

The password is assigned to the XSA\_ADMIN user during installation.

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

 Output Code

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

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

Make a note of the initial password.

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

- XS\_CONTROLLER\_USER  
Modify access within the assigned organization or space
- XS\_AUTHORIZATION\_DISPLAY  
Read-only access to tools

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

 Tip

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

```
xs create-space DEV
```

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

### ➔ Tip

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

```
xs create-org devteam1
```

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

In this step, you assign the following user roles:

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

```
xs set-org-role DEVUSER devteam1 OrgManager
```

```
xs set-space-role DEVUSER devteam1 DEV SpaceDeveloper
```

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

```
xs login -u DEVUSER
```

### ➔ Tip

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

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

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

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

### Output Code

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

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

```
xs login -u DEVUSER
```

```
xs -v
```

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

```
xs marketplace
```

The following information should be displayed:

### Output Code

```
$ xs marketplace
```

```

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

```

13. List all manually defined service instances.

**xs services**

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

#### Output Code

```

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

```

14. Create an instance of the XS User Authentication & Authorization (UAA) service.

**xs create-service xsuaa default my-uaa**

The following information should be displayed:

#### Output Code

```

Creating service "my-uaa"...
OK

```

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

**xs create-service hana hdi-shared my-hdi-container**

The following information should be displayed:

#### Output Code

```

Creating service "my-hdi-container"...
OK

```

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

**xs services**

The following information should be displayed:

 Output Code

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

## Related Information

[The XS Command-Line Interface Reference \[page 848\]](#)

[SAP Note 2242468: Setting up the XS Advanced CLI](#)

[Standard Node.js Packages for XS Advanced \[page 519\]](#)

[Deploy a Node.js Hello-World Application using the XS CLI \[page 84\]](#)

[Deploy a Node.js Application that Uses Database Artifacts and OData Services \[page 89\]](#)

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

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

### Prerequisites

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

### ➔ Tip

Alternatively, you can configure NPM to resolve SAP-related NPM dependencies using the SAP public NPM Registry (at [npm.sap.com](https://npm.sap.com)).

- You have a GIT client installed on your development machine.

### ➔ Tip

The SAP Web IDE for SAP HANA includes an integrated Git client.

- Your development machine has access to the Internet
- The XS advanced services specified in the applications' manifest files are available, for example:
  - my-uaa (user account and authorization)
  - my-hdi-container (access to database artifacts in an HDI container).

## Context

In the following tutorial, you learn how to use the XS CLI to deploy a simple Node.js application ("myapp1") to the XS advanced-model JavaScript run time; the deployed application displays a "Hello World" page and allows you to run a simple JavaScript service:

## Procedure

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

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

- "myapp1"  
Simple "Hello-World" JavaScript application
- "myapp2"  
JavaScript "Hello-World" application that uses database artifacts and an XS OData service

```
git clone https://github.com/saphanaacademy/XSA.git
```

### Output Code

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

In the cloned repository on your local machine, you should see the following high-level structure; in this tutorial, we focus on the first of the cloned applications, “myapp1”:

```
<gitrepo>
|- myapp1/
|   |- web/
|   |   |- xsjs/
|   |   \- manifest.yml
|- myapp2/
|   |- db/
|   |   |- web/
|   |   |   |- xsjs/
|   |   \- manifest.yml
.gitattributes
README.md
```

2. Check the contents of the cloned `myapp1` folder.

The cloned `myapp1` folder should look like the following example:

```
| - myapp1/
|   |- web/
|   |   |- resources/
|   |   |   |- index.html
|   |- xsjs/
|   |   |- package.json
|   |   |- start.js
|   |   \- lib/
|   |       \- welcome.xsjs
\- manifest.yml
```

3. Install the SAP-specific Node.js packages included in the `XS_JAVASCRIPT` archive.

The `XS_JAVASCRIPT` archive (`xs_javascript-1.3.0-bundle.tar.gz`) is included in the XS CLI client tools (`xs.onpremise.runtime.client_<platform>-<version>.zip`) specified in the prerequisites section. Extract the `xs_javascript-1.3.0-bundle.tar.gz` file to a central and easy-to-find location, for example:

- Windows  
`C:\xsclient\nodejs\node_modules`
  - Linux  
`/var/lib/xsclient/nodejs/node_modules`
4. Adapt the application's deployment dependencies defined in the `manifest.yml` file to suit your SAP HANA XS advanced environment.
    - a. Check the module names, for example: “`myapp1-web`” and “`myapp1-xsjs`”.
    - b. Choose a port for the `web/` and `xsjs/` modules.

You can allow SAP HANA XS advanced to allocate ports automatically, or choose a free port manually, for example, “64011” and “64012”, for `web/` and `xsjs/` respectively.

- c. Check the module paths, for example: “`web`” and “`xsjs`”. The path must point to the location where module-specific files are stored.
- d. Check that the service names match the service instances you created, for example, “`xs-uaa`” for user authentication and authorization in XS advanced.
- e. Adapt the application's destination in the `web/` module to point to the application's entry point `start.js` with the appropriate host and port.

### Sample Code

```
[  
  {"name":"xsjs", "url":"https://host.acme.com:64012",  
  "forwardAuthToken": true}  
]
```

The `manifest.yml` file for the application “myapp1” should look like the following example:

### Sample Code

```
---  
applications:  
- name: myapp1-web  
  port: 64011  
  memory: 100M  
  path: web  
  env:  
    destinations: >  
      [  
        {"name":"xsjs", "url":"https://host.acme.com:64012",  
        "forwardAuthToken": true}  
      ]  
    services:  
      - my-uaa  
- name: myapp1-xsjs  
  port: 64012  
  path: xsjs  
  memory: 128M  
  services:  
    - my-uaa
```

5. Ensure dependencies between required micro-services can be resolved.

Copy the `node_modules` folder from the central location where you stored it after download to the new application's `xsjs/` and `web/` folders, for example:

- `web/node_modules`  
In the folder `web/node_modules`, you do not need all the packages; you only need: `approuter`, `@sap/logging`, `@sap/xsenv`, and `@sap/xsjs`.
- `xsjs/node_modules`  
All packages are required.

6. Check the package dependencies for the application's `web` module “myapp2-web”; the dependencies are defined in the file `myapp1/web/package.json`.

### Sample Code

Application Router's Package Dependencies (`myapp1/web/package.json`)

```
{  
2   "name": "myapprouter",  
3   "dependencies": {  
4     "approuter": "1.1.5"  
5   },  
6   "scripts": {  
7     "start": "node node_modules/approuter/approuter.js"  
8   }  
9 }
```

7. Check the specifications of the application's router configuration; the details are specified in the file `myapp1/web/xs-app.json`.

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

### Sample Code

Application Router Configuration (`myapp1/web/xs-app.json`)

```
{  
    "welcomeFile": "index.html",  
    "logout": {  
        "logoutEndpoint": "/do/logout"  
    },  
    "routes": [  
        {  
            "destination": "xsjs",  
            "source": "(.*)(.xsjs)"  
        },  
        {  
            "destination": "xsjs",  
            "source": "(.*)(.xsodata)"  
        },  
        {  
            "localDir": "resources",  
            "source": "^/(.*)"  
        }  
    ]  
}
```

8. Deploy the JavaScript application “myapp1” to the XS advanced run time.

- Change directory to the application's root folder, which contains the deployment manifest file `manifest.yml`.
- Deploy the application

```
xs push
```

### Output Code

```
Using manifest file "manifest.yml"  
Creating app "myapp1-web" in org "devteam1" / space "DEV" as DEVUSER  
Creating route "myapp1-web.host.acme.com:64011" in org "devteam1" /  
space "DEV" as DEVUSER  
Binding route "myapp1-web.host.acme.com:64011" to app "myapp1-web"  
...  
staging app "myapp1-xsjs"  
starting app "myapp1-xsjs"  
Showing status and information about app "myapp1-xsjs"  
  name:          myapp1-xsjs  
  requested state:  STARTED  
  instances:      1  
  memory:        128 MB  
  disk:          <unlimited>  
  buildpack:     <default>  
  urls:  
           https://host.acme.com:64012  
  
Instances:  
index   created           state  
-----
```

```
0 Dec 25, 2015 10:22:37 AM RUNNING
```

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

```
xs apps
```

### Output Code

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

name requested state instances memory disk urls
-----
myapp1-web STARTED 1/1 100 MB <unlimited> https://
host.acme.com:64011
myapp1-xsjs STARTED 1/1 128 MB <unlimited> https://
host.acme.com:64012
```

10. Check the application in a Web browser.

Specify the entry point for the application router: <https://host.acme.com:64011>

A logon screen is displayed; log on as DEVUSER.

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

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

## Related Information

[Working with the XS Advanced Command-Line Client \[page 78\]](#)

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

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

## Prerequisites

- The XS advanced run time is installed and available on the SAP HANA server
- You have installed the XS command-line client on your development machine.  
The XS CLI client tools are installed by default on the SAP HANA server. However, you must download and install the client tools locally, if you want to connect to SAP HANA from your local machine. The XS CLI

client tools (`xs.onpremise.runtime.client_<platform>-<version>.zip`) can be downloaded from the SAP HANA server, from the installation DVD, or from the SAP support portal. For more information about the SAP support portal, see the link to SAP note 2242468 in the related links below.

- SAP Node.js packages

The archive (`xs_javascript-<version>-bundle.tar.gz`) is available on the SAP HANA media and the SAP Marketplace (SMP); it includes a selection of Node.js packages developed by SAP for use with the Node.js applications running in the XS advanced run time.

➔ Tip

Alternatively, you can configure NPM to resolve SAP-related NPM dependencies using the SAP public NPM Registry (at [npm.sap.com](https://npm.sap.com)).

- You have a GIT client installed on your development machine.

➔ Tip

The SAP Web IDE for SAP HANA includes an integrated Git client.

- Your development machine has access to the Internet
- The XS advanced services specified in the applications' manifest files are available, for example:
  - `my-uaa` (user authentication and authorization)
  - `my-hdi-container` (access to database artifacts in the HDI container).

## Context

In the following tutorial, you learn how to use the XS CLI to deploy a more advanced Node.js application ("myapp2") to the XS advanced-model JavaScript run time; the deployed application uses an HDI container with simple database artifacts, implements some simple business logic to insert and extract data from a table, and calls a simple OData service, too:

## Procedure

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

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

- "myapp1"  
Simple JavaScript application
- "myapp2"  
JavaScript application with database artifacts and an XS OData service

```
git clone https://github.com/saphanaacademy/XSA.git
```

Output Code

```
cloning into 'XSA' ...
```

```
remote: Counting objects: 33, done.  
remote: Compressing objects: 100% (27/27), done.  
remote: Total 33 (delta 2), reused 33 (delta 2), pack-reused 0  
unpacking objects: 100% (33/33), done.  
Checking connectivity... done.
```

In the cloned repository on your local machine, you should see the following high-level structure; in this tutorial, we focus on the cloned application “`myapp2`”:

### Output Code

```
<gitrepo>  
|- myapp1/  
|   |- web/  
|   |- xsjs/  
|   \- manifest.yml  
|- myapp2/  
|   |- db/  
|   |- web/  
|   |- xsjs/  
|   \- manifest.yml  
.gitattributes  
README.md
```

2. Check the contents of the cloned `myapp2` folder.

The cloned `myapp2` folder should look like the following example:

### Output Code

```
|- myapp2/  
|   |- db/                      # Database artifact  
|   |   |- package.json          # Node.js package dependencies  
|   |   \- src/  
|   |       |- .hdiconfig  
|   |       |- .hdinamespace  
|   |       \- MyContext.hdbcards  
|   |- web/                      # Static Web resources  
|   |   |- package.json          # Node.js package dependencies  
|   |   |- xs-app.json           # Route configuration  
|   |   \- resources/  
|   |       \- index.html  
|   |- xsjs/                      # Application code  
|   |   |- package.json          # Node.js package dependencies  
|   |   |- start.js              # Application entry point  
|   |   \- lib/  
|   |       |- insert.xsjs  
|   |       |- query.xsjs  
|   |       |- service.xsodata  
|   |       \- welcome.xsjs  
|   \- manifest.yml               # Application overview
```

3. Install the SAP-specific Node.js packages included in the `XS_JAVASCRIPT` archive. The applications `myapp1` and `myapp2` have dependencies to the packages include in this archive.

The `XS_JAVASCRIPT` archive (`xs_javascript-1.3.0-bundle.tar.gz`) is included in the XS CLI client tools (`xs.onpremise.runtime.client_<platform>-<version>.zip`) specified in the prerequisites

section. Extract the `xs_javascript-1.3.0-bundle.tar.gz` file to a central and easy-to-find location, for example:

- Windows  
`C:\xsclient\nodejs\node_modules`
- Linux  
`/var/lib/xsclient/nodejs/node_modules`

4. Adapt the application's deployment dependencies defined in the `manifest.yml` file to suit your SAP HANA XS advanced environment.

- a. Check the module names, for example: "myapp2-web", "myapp2-db", and "myapp2-xsjs".
- b. Choose a port for the `web/` and `xsjs/` modules.

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

### → Tip

A port allocation is not required for the database module `db`.

- c. Check the module paths, for example: "db", "web" and "xsjs". The path must point to the location where module-specific files are stored.
- d. Check that the database module `myapp2-db` includes the entry "`no-route: true`". This entry is mandatory and signifies that no direct access to the HDI container is necessary from the UI; access is ensured by other means, for example, JavaScript code or an OData service.
- e. Check that the service names match the service instances you created, for example, "`xs-hdi-container`" for the HDI container for the deployed database artifacts" and `xs-uaa`" for user authentication and authorization in XS advanced.
- f. Adapt the application's destination in the `web/` module to point to the application's entry point `start.js` with the appropriate host and port.

### Sample Code

```
[  
  {"name":"xsjs", "url":"https://host.acme.com:64022",  
  "forwardAuthToken": true}  
]
```

When complete, the `manifest.yml` file for the application "myapp2" should look like the following example:

### Sample Code

```
---  
applications:  
- name: myapp2-web  
  port: 64021  
  memory: 100M  
  path: web  
  env:  
    destinations: >  
      [  
        {"name":"xsjs", "url":"https://host.acme.com:64022",  
        "forwardAuthToken": true}  
      ]
```

```

services:
  - my-uaa
- name: myapp2-db
  path: db
  memory: 265M
  no-route: true
  services:
    - my-hdi-container
- name: myapp2-xsjs
  port: 64022
  path: xsjs
  memory: 128M
  services:
    - my-uaa
    - my-hdi-container

```

5. Ensure dependencies between required micro services can be resolved.

Copy the `node_modules` folder from the central location where you stored it after download to the new application's `db/`, `web/`, and `xsjs/folders`, for example:

- `db/node_modules`  
The following packages are required: `@sap/hdi-deploy`
- `web/node_modules`  
The following packages are required: `approuter`, `@sap/logging`, `@sap/xsenv`, and `@sap/xsjs`.
- `xsjs/node_modules`  
All packages are required.

6. Check the package dependencies for the new database module "myapp2-db"; the dependencies are defined in the file `myapp2/db/package.json`.

### Sample Code

Database Container Dependencies (`myapp2/db/package.json`)

```
{
  "name": "deploy",
  "dependencies": {
    "@sap/hdi-deploy": "1.0.0"
  },
  "scripts": {
    "start": "node node_modules/@sap/hdi-deploy/"
  }
}
```

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

### Output Code

```

|- myapp2/
  |- db/
    |- package.json
    \- src/
      |- .hdiconfig
      |- .hdinamespace
      \- MyContext.hdbcards

```

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

### Sample Code

HDI Configuration (`.hdiconfig`)

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

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

### Sample Code

HDI Namespace Configuration (`.hdinamespace`)

```
{  
  "name" : "myapp2"  
  "subfolder": "append"  
}
```

### Note

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

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

### Sample Code

Database Artifact (`MyContext.hdbcards`)

```
namespace myapp2;  
context MyContext {  
  Entity MyTable {  
    key id : Integer;  
    value : String(100)
```

```
    };
};
```

8. Check the contents of the `myapp2` application's JavaScript `xsjs` module; the `lib/` subfolder contains the code for the business logic used by the application.

#### Sample Code

```
| - myapp2/
|   | - xsjs/
|   |   | - package.json
|   |   | - start.js
|   |   \- lib/
|   |       | - insert.xsjs      # Insert data into a table
|   |       | - query.xsjs      # Query data in a table
|   |       | - service.xsodata # XS OData service to extract data
|   |       \- welcome.xsjs     # Display a simple welcome message
\- manifest.yml
```

The `service.xsodata` defines an OData service used by the application `myapp2` to extract data from the table "MyTable" (in the context "MyContext" and the name space "myapp2"):

#### Sample Code

```
service: {
    "myapp2": "MyContext.MyTable" as "MyEntity";
}
```

9. Check the package dependencies for the application's `web` module "myapp2-web"; the dependencies are defined in the file `myapp2/web/package.json`.

#### Sample Code

Application Router's Package Dependencies (`myapp2/web/package.json`)

```
{
  2   "name": "myapprouter",
  3   "dependencies": {
  4     "approuter": "1.1.5"
  5   },
  6   "scripts": {
  7     "start": "node node_modules/approuter/approuter.js"
  8   }
  9 }
```

10. Check the specifications of the application's router configuration; the details are specified in the file `myapp2/web/xs-app.json`.

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

#### Sample Code

Application Router Configuration (`myapp2/web/xs-app.json`)

```
{
  "welcomeFile": "index.html",
```

```

"logout": {
    "logoutEndpoint": "/do/logout"
},
"routes": [
{
    "destination": "xsjs",
    "source": "(.*)(.xsjs)"
},
{
    "destination": "xsjs",
    "source": "(.*)(.xsodata)"
},
{
    "localDir": "resources",
    "source": "^/(.*)"
}
]
}

```

11. Check the routing configuration for the application's Web module.
12. Deploy the JavaScript application "myapp2" to the XS advanced run time.
  - a. Change directory to the folder that contains the myapp2 application's deployment manifest file "manifest.yml".
  - b. Deploy the application

**xs push**

### Output Code

```

Using manifest file "manifest.yml"
Creating app "myapp2-web" in org "devteam1" / space "DEV" as DEVUSER
Creating route "myapp2-web.host.acme.com:64021" in org "devteam1" /
space "DEV" as DEVUSER
Binding route "myapp2-web.host.acme.com:64021" to app "myapp1-web"
Uploading "myapp2-web"...
...
staging app "myapp2-xsjs" ...
starting app "myapp2-xsjs" ...
Showing status and information about app "myapp2-xsjs"
  name:          myapp2-xsjs
  requested state:  STARTED
  instances:        1
  memory:         128 MB
  disk:            <unlimited>
  buildpack:       <default>
  urls:            https://host.acme.com:64022

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

```

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

**xs apps**

### Output Code

```

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

```

name	requested state	instances	memory	urls
<hr/>				
myapp2-web	STARTED	1/1	100 MB	https://host.acme.com:64021
myapp2-db	STARTED	1/1	100 MB	<none>
myapp2-xsjs	STARTED	1/1	128 MB	https://host.acme.com:64022

### i Note

The database application has no URL assigned because it not accessed directly from the UI (`no-route:true`); it is accessed from the HDI container by means of the JavaScript and OData business logic.

14. Check the running services.

**xs services**

The following information should be displayed:

### Output Code

```
xs services
Getting services in org "devteam1" / space "DEV" as DEVUSER
Found services:
name          service      plan        Bound apps
-----
my-uaa         xsuaa       default     myapp2-web, myapp2-xsjs
my-hdi-container hana       hdi-shared  myapp2-db
```

15. Check the application `myapp2` in a Web browser.

Specify the entry point for the application router: <https://host.acme.com:64021>

The following Web page is displayed

## Static HTML - Hello World!

[Welcome](#)

[Query](#)

**Insert** id:  value:

1. Test the “Welcome” link.

Click [Welcome](#); a Web page is displayed with the message: XSJS – Hello DEVUSER

2. Insert some data into your table by using the “Insert” button.

Enter an `id` (for example, “1”) and a `value` (for example, “Kofi”) and choose [Insert](#). If the operation is successful, a Web page is displayed with the message XSJS Insert.

3. Run a query on the table data by using the “Query” link.

Choose [Query](#). If the operation is successful, a Web page is displayed with the content: `[{"id": 1, "value": "Kofi"}]`. At this point, this is the only data in the table.

4. Test the OData service that you defined.

In the Web browser, enter the following URL: <https://host.acme.com:64021/service.xsodata>

### Output Code

```
<service xmlns:atom="http://www.w3.org/2005/Atom" xmlns:app="http://www.w3.org/2007/app" xmlns:app="http://www.w3.org/2007/app" xml:base="https://host:64021/service.xsodata/">
  <workspace>
    <atom:title>Default</atom:title>
    <collection href="MyEntity">
      <atom:title>MyEntity</atom:title>
    </collection>
  </workspace>
</service>
```

## Related Information

[Getting Started with Application Development in XS Advanced \[page 25\]](#)

[Set up an XS Advanced Application Project \[page 104\]](#)

[Get Started with the XS CLI Client \[page 79\]](#)

[Standard Node.js Packages for XS Advanced \[page 519\]](#)

[The SAP NPM Registry \[page 560\]](#)

## 3.3 Working with Gerrit Version Control in XS Advanced

An overview of the version-control tools available for the management of design-time artifacts in SAP HANA XS advanced model.

Gerrit is an open-source Git server that provides access control for the hosted Git repositories as well as a Web-based client for doing code review. Code review is a core (but optional) of Gerrit; individual development teams can decide whether to work with or without code review.

Gerrit is an optional component of the XS advanced platform which can be used to store and manage versions of the source code for XS advanced applications, for example, JavaScript or Java applications and SAPUI5 in Git repositories.

### ➔ Tip

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

Git is a widely used open-source system for revision management of source code that facilitates distributed and concurrent large-scale development workflows. It supports a large variety of commands for working with repositories.

### **i** Note

To work with Gerrit you must understand the basic concepts of Git and be familiar with the basic Git commands and workflows used when developing an application. In addition, to perform Git operations, you must also have access to a suitable Git client, for example, the SAP Web IDE for SAP HANA.

## Related Information

[Set up Gerrit for XS Advanced Application Development \[page 99\]](#)

[Using Source Control \(Git\) \[page 1113\]](#)

### 3.3.1 Set up Gerrit for XS Advanced Application Development

A description of the basic steps developers need to perform to get started with Gerrit and Git version-management tools.

## Prerequisites

To start working with Gerrit version control, the following prerequisites apply:

- A Gerrit server is available in your development environment
- The Gerrit server is configured to work with XS advanced, for example, the service broker
- You have access to a Git client, for example, the client included with SAP Web IDE for SAP HANA

### **i** Note

If the SAP Gerrit server is installed in an environment where SAP HANA XS advanced model is already available, the Gerrit installation process discovers the SAP HANA instance and configures the required configuration automatically. If you are installing SAP HANA and want to connect SAP HANA XS advanced to an existing Gerrit server, you must configure the connection manually.

## Context

To get started with Gerrit in XS advanced, you need to perform the following steps:

### ➔ Tip

The documentation for Gerrit for XS advanced is available on the system where the Gerrit for XS advanced server is running at the following URL:

<https://<GerritServer>.acme.com:8080/Documentation/index.html>

## Procedure

1. Sign in to Gerrit.

Before you can start uploading files and changes to Gerrit, you must create and activate your Gerrit account.

- a. Open the Gerrit Web UI's start page in a Web browser.

### i Note

You can obtain the URL of the Gerrit server from the XS advanced run time, for example, using the command `xs version`.

Check the information displayed for `git-service` in the system output under the section `Registered Service URLs`, as illustrated in the following example:

### i Note

You will need to change the port number to the one published on your system, for example:

[https://host.acme.com:29<HANA\\_Instance\\_Nr>9](https://host.acme.com:29<HANA_Instance_Nr>9)

### Output Code

```
xs version
Client version: xs v0.1614-cons
Server version information:
  name          = XS Controller
  support       = http://service.acme.com/message
  build         = v0.1614-cons
  version       = 1
  user          = <not set>
  description   = SAP XS Runtime on premise
  controllerEndpoint = https://host.acme.com:30030
  authorizationEndpoint = https://host.acme.com:30032/uaa-security
  loggingEndpoint = <not set>
  allowDebug    = true
  ...
Registered service URLs:
  deploy-service      = https://host.acme.com:51002
  component-registry = https://host.acme.com:51003
```

```
product-installer      = https://host.acme.com:51005  
git-service           = https://host.acme.com:29419
```

- b. Choose *Sign In* in the top right-hand corner of the screen.

Gerrit for XS Advanced is attached to the user management system (XSUAA) of the XS Advanced platform. As a result, the sign-in process redirects you to the standard log-in screen of the SAP HANA database.

**i Note**

If you do not have an SAP HANA identity in the XS UAA server, contact your SAP HANA system administration team and ask them to create a user account for you.

- c. Enter your SAP HANA user credentials to sign in to Gerrit.  
Gerrit displays your personal *My Reviews* dashboard.
2. Have a look at your *Reviews Dashboard*.

You can manage the projects you are assigned to in *Watched Projects*, which you can find in ► *Settings* ► *Watched Projects* ▶.

**i Note**

When you sign in to Gerrit for the first time, your personal dashboard will typically be empty, because you have not yet been assigned to any Gerrit project.

3. Select a Gerrit development project to join.

To display a list of all available projects, choose ► *Projects* ► *List* ▶.

This list shows all projects to which you have at least “read” access rights together with a short description and a link to the repository browser. The repository browser allows you to examine the content and history of the Git repository associated with the Gerrit project.

4. Display details of a Gerrit development project.

To display details of a Gerrit project, choose the name of the project in the project list. The Gerrit project contains the reference to the Git repository used to manage the project’s application code.

**➔ Tip**

To clone the project to your workplace, you need the project’s URL, which is displayed immediately below the project title. To clone the Gerrit project, you can use any standard Git client, for example, a Git command-line client or the Git tools provided with SAP Web IDE for SAP HANA.

5. Create a new project.

If you are not assigned to a Gerrit project, and cannot find a project in the list of existing projects, you might have to create a new project. Gerrit provides the following methods for creating a new project:

**i Note**

In a production environment, the permissions required to create a new Gerrit project are granted to a selected group of “admin” users. As a new Gerrit user, you will need to ask the Gerrit administration group to create the new project for you.

- In the Gerrit Web UI under [Projects](#) [Create Project](#)
- With the XS advanced service broker

### ➔ Tip

If you intend to bind the new repository to an XS advanced application, you must create it with the XS advanced service broker. Git repositories not created with the XS advanced service broker are not visible for the XS Advanced run time.

## 6. Access the Git repository.

The Git repository stores the authoritative copy of the application source code being developed in the project. Project team members fetch code from (and push code to) this repository. Build servers and other packaging tools use the Git repository to obtain the source-code, too.

By default, Gerrit requires authentication and authorization for all Git operations. For example, to upload changes to a Git repository, you must either be a project owner or have been granted the appropriate access rights for the target repository. If you need write access to a Gerrit project, contact the project owner or the Gerrit administrator.

### i Note

Default access rules allow all authenticated users to push changes for review to any Gerrit project. However, only project owners are allowed to submit changes to the underlying repository. This facilitates the implementation of the simple contributor-committer work flow followed by many open-source projects.

It is possible to enable anonymous read access to a repository by adding a “Read” permission for the reference refs/\* and assigning that permission to the predefined group “Anonymous Users”. To grant anonymous read access for **all** repositories, you must adapt the configuration of the project [All-Projects](#), from which all other projects inherit their permissions.

## 7. Set up your Gerrit profile.

Before you start working with Gerrit you should check and, if necessary, amend your contact information, and customize Gerrit to your personal needs and preferences. You might also want to enable SSH communication for your account.

- Contact information  
Git identifies the author and committer of a change by their e-mail addresses, and you cannot upload any changes to Gerrit if the configured e-mail address is wrong or invalid. Gerrit stores these attributes in your contact information, which you can access under [Settings](#) [Contact Information](#).

### i Note

When you sign in to Gerrit for the first time, an account is created using the full name and e-mail address received from the XS advanced user-management system (XSUAA). In some cases, this information might not be accurate, especially if you sign in using the credentials of an SAP HANA database user. Furthermore, in a productive environment it might be necessary to use your corporate e-mail address and real name, regardless of your identity in the SAP HANA system.

- Preferences

You can customize Gerrit by configuring your preferences in your profile, for example, under .

- Secure Shell (SSH) settings

Gerrit supports the SSH protocol for Git operations. If you want to use SSH for communication with Gerrit, you first must generate a suitable SSH key pair and upload the public key to your profile, for example, under .

8. Clone an existing repository. (optional)

If you already have a Git repository that you want to clone, you can use the tools provided by SAP Web IDE for SAP HANA to perform the clone operation.

- a. Obtain the URL of the Gerrit server where the Git repository that you want to clone is located.

**Note**

You can display the URL of the SAP Gerrit server for XS advanced from the XS advanced run time, for example, using the `xs version` command.

Check the information displayed for `git-service` in the system output under the section Registered Service URLs, as illustrated in the following example:

**Output Code**

```
xs version
Client version: xs v0.1614-cons
Server version information:
  name          = XS Controller
  support       = http://service.acme.com/message
  build         = v0.1614-cons
  version       = 1
  user          = <not set>
  description   = SAP XS Runtime on premise
  controllerEndpoint = https://host.acme.com:30030
  authorizationEndpoint = https://host.acme.com:30032/uaa-security
  ...
Registered service URLs:
  deploy-service      = https://host.acme.com:51002
  component-registry = https://host.acme.com:51003
  product-installer  = https://host.acme.com:51005
  git-service        = https://host.acme.com:29419
```

b. Start SAP Web IDE for SAP HANA.

If SAP Web IDE for SAP HANA is not available at the default URL `https://host.acme.com:53075`, you can use the `xs` CLI to display the URL allocated to the application. In a command shell, log in to the XS advanced run time and use the command `xs app webide --urls` to display the URL required to access SAP Web IDE for SAP HANA in a Web browser.

c. Clone the Git repository.

In SAP Web IDE for SAP HANA, choose .

You will need to provide the following details:

- **URL**

The URL to the Gerrit server where the project's Git repository is located, for example, `https://host.acme.com:29419`

- **Repository Path**  
The path to (and name of) the Git repository you want to clone, for example, `/<RepositoryName>.git`
- **Connection Protocol**  
The protocol to use for the connection to the Gerrit server, for example, `HTTPS`.
- **Authentication**  
The log-on credentials for a SAP HANA user with access permissions for the cloned Git repository.

## Related Information

[Working with Gerrit Version Control in XS Advanced \[page 98\]](#)

[Using Source Control \(Git\) \[page 1113\]](#)

## 3.4 Set up an XS Advanced Application Project

An overview of the steps required to set up and deploy an application project for SAP HANA XS advanced.

### Prerequisites

- Development tools

You can choose between graphical and command-line tools:

- SAP Web IDE for SAP HANA  
If the SAP Web IDE for SAP HANA is not available at the default URL `https://host.acme.com:53075`, log in to the XS advanced run time with the XS command-line interface (CLI) and use the command `xs app webide --urls` to display the URL required to access the tool.
- You have installed the XS command-line client on your development machine.  
The XS CLI client tools are installed by default on the SAP HANA server. However, you must download and install the client tools locally, if you want to connect to SAP HANA from your local machine. The XS CLI client tools (`xs.onpremise.runtime.client_<platform>-<version>.zip`) can be downloaded from the SAP HANA server, from the installation DVD, or from the SAP support portal. For more information about the SAP support portal, see the link to SAP note 2242468 in the related links below.
- SAP Node.js packages  
The archive (`xs_javascript-<version>-bundle.tar.gz`) is available on the SAP HANA media and the SAP Marketplace (SMP); it includes a selection of Node.js packages developed by SAP for use with the Node.js applications running in the XS advanced run time.

#### → Tip

Alternatively, you can configure NPM to resolve SAP-related NPM dependencies using the SAP public NPM Registry (at [npm.sap.com](http://npm.sap.com)).

- Version control, for example, GIT/Gerrit  
Gerrit for SAP HANA XS advanced is an optional component of the XS advanced platform which can be used to store and manage versions of the source code for XS advanced applications (for example, JavaScript or Java applications and SAPUI5) in Git repositories.

➔ Tip

The SAP Web IDE for SAP HANA includes an integrated Git client.

- Access to a running SAP HANA system where the XS advanced run time is installed and configured

⚠ Caution

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

## Context

The end-to-end process of developing an application that you want to deploy on SAP HANA is straight forward, as illustrated in the steps described in this high-level overview.

Table 3: XS Advanced Application Development Overview

Step	Description
1	Create the design-time folder infrastructure for your application source files
2	Create the deployment descriptor files for your application
3	Add database artifacts and content
4	Add application business logic to work with the database artifacts
5	Create any required OData service definitions
6	Create the user interface in which to display the database content
7	Add security to protect what you have built
8	Define application routes (and destinations)
9	Create any service instances needed by your application
10	Add scheduled jobs if required (optional)
11	Deploy your project
12	Test your project in the user interface

### Note

Some configuration artifacts are subject to restrictions regarding their location; SAP HANA XS advanced deployment tools **expect** to find particular artifacts in a particular location.

## Procedure

1. Create the design-time folder infrastructure for your application source files.

To make the maintenance of source files easier and facilitate the task of deploying the application, it is also essential to create a clear and consistent structure for the design-time artifacts that you develop for your application. The folders typically include modules for the following areas: database, Web, application code, OData (optional). The security description (OAuth2 client configuration) can be placed either in its own root-level module or in the application code structure, for example, `js/security/` or `java/security/`.

### Tip

SAP Web IDE for SAP HANA provides Wizards and templates to assist with the creation of application modules and the related resources for your MTA. If you are using the XS command-line tools to develop an MTA for SAP HANA XS advanced, you must create all modules and resources yourself.

### Sample Code

```
<myAppName>
|- db/                                # Database module
|  |- package.json                      # Database details/dependencies
|  \- src/
|     |- .hdiconfig                     # Database artifacts: tables, views,
|     |- .hdinamespace                  # HDI plug-in configuration
|     \- myModel.hdbcds                # HDI plug-in namespace configuration
(optional)
|     \- myModel.hdbcds                # Database design-time definition
|- js/                                 # JavaScript app module
|  |- start.js                          # JavaScript app entry point
|  |- package.json                      # JavaScript app details/dependencies
|  \- src/
|     |- odata/
|        |- resources/                 # OData service resources
|        \- services/                 # OData service definitions
|           | - srv1.xsodata          # OData service definition
|           \- srvN.xsodata          # OData service definition
|- ui/                                  # Application UI module
|  |- xs-app.json                       # Application routes configuration
|  |- package.json                      # Application details/dependencies
|  \- resources/
|     \- index.html                     # Static UI resources
|- xs-security.json                    # Client UI start page
|- manifest.yml                        # Application security description
|- mta.yaml                            # Manifest description incl. destinations
# Central development manifest (SAP Web)
IDE only
\- mtad.yaml                          # Central deployment manifest
```

2. Create the development and deployment descriptor files for your application.

The development and deployment descriptors specify what application components to build (and how), the dependencies between the application components, and where (and how) to deploy the components.

The development and deployment manifest files for the multi-target application must be located in the application's root folder.

- `mta.yaml`  
Development manifest for a multi-target application (MTA)

 **Tip**

The `mta.yaml` is only required if you are working with SAP Web IDE for SAP HANA, which creates (and maintains) the file automatically when you create a new application project and maintain the project's components.

- `mtad.yaml`  
Deployment manifest for a multi-target application (MTA)

 **Tip**

If you are working with the XS advanced command-line client, you must create (and maintain) the `mtad.yaml` file manually. If you are using SAP Web IDE for SAP HANA, no action is required; SAP Web IDE for SAP HANA creates the deployment manifest `mtad.yaml` transparently at deployment time using the information specified in the corresponding development descriptor (`mta.yaml`).

3. Add database artifacts and content.

Create database artifacts such as tables and views, for example using Core Data Services (CDS). You can also create procedures and sequences, for example, using SQLScript.

 **Note**

The database content must include a `package.json` file containing details of dependencies and, in addition, the container configuration (`.hdiconfig`) and run-time name space configuration (`.hdinamespace`) files.

4. Add application business logic to work with the database artifacts.

The business logic is used to help retrieve data from the database, for example, using OData services. The data is requested from and displayed in the client user interface.

For applications written in JavaScript (Node.js or XS JavaScript), you need to perform the following high-level steps:

- a. Add a package description file (`package.json`) to your JavaScript resource-files folder with details of your application's contents and dependencies.
- b. Add a startup file for your JavaScript application, for example, `main.js`.
- c. Create any additional JavaScript (`.js` or `.xsjs`) files and libraries.

For applications written in Java, you need to perform the following high-level steps:

- a. Write Java application's code.
- b. Look up the required data source.
- c. Build the WAR file based on the project object model file (`pom.xml`), for example, with Maven.
- d. Use the Java Persistence API (JPA) to work with CDS entities (if required).

5. Create any required OData service definitions.

An OData service definition is described in an `xsodata` artifact, for example, `myODataService.xsodata`; the service definition should be placed in the `resources/` folder of your OData project structure.

6. Create the user interface (GUI) to display the database content (HTML pages).
7. Add security to protect what you have built (`xs-security.json`).

The tasks required to set up authorization artifacts in SAP HANA XS advanced are performed by two distinct user roles:

- Application developer  
Creates the security descriptor (`xs-security.json`) for the XS advanced application; maintains role templates and authorization scopes (in the `xs-security.json`) to define who has access to the application; binds the application to an XS advanced service instance; deploys the application to the XS advanced run-time environment.
- SAP HANA XS administrator.  
After deployment of the authorization artifacts as role templates, the XS advanced run-time administrator uses the role templates provided by the developers to build roles, group the roles into role collections, and assign the role collections to business users, for example, using the [SAP HANA XS Administration Tools](#).

8. Define application routes (and destinations).

The application router is used to serve static content, propagate user information, and act as a proxy to forward requests to other micro services. Application routes are defined in the application-router descriptor `xs-app.json`, a JSON-formatted file that should be located in the application's Web resources module.

Destinations are defined in an environment variable for the `@sap/approuter` application. The destinations are typically specified in the application manifest (`manifest.yml`) file. The router information is added to the `env: destinations` section of the `manifest.yml`, as illustrated in the following example.

9. Create any service instances needed by your application.

### Note

SAP Web IDE for SAP HANA creates any required services based on the information in the "resources" section of the application's development/deployment descriptor (`mta.yaml`) and binds the application to the service instance created.

If you are using the XS advanced CLI, you must create any services and manually bind the application to the service instance, for example, using the `xs create-service` and `xs bind-service` commands.

```
xs create-service <SERVICE> <PLAN> <SERVICE_INSTANCE>
  ○ HDI container service instance
    xs create-service hana hdi-shared my-hdi-container
  ○ User Account and Authentication (UAA) service
    Using information specified in the application's security descriptor (xs-security.json)
    xs create-service xsuaa default -c </path/to/>xs-security.json
```

10. Add scheduled jobs if required (optional).

Jobs can be scheduled using an instance of the Job Scheduler service.

### Note

For more information, see *Scheduling Jobs in XS advanced* in the SAP HANA Administration Guide (*Related Links* below).

11. Deploy your application project.

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

- o XS CLI client
  - o Push an individual application component/module:  
`xs push <myAppName>`
  - o Deploy a complete MTA archive:  
`xs deploy <myMTAName>. [mtar | zip]`
  - o Install a Software Component Version (SCV):  
`xs install <myAppArchive>.zip`

12. Test your application project in the user interface.

On build or deployment, an XS advanced application is assigned a URL with a specific port. You can paste the assigned URL into a Web Browser to test it. This information is displayed differently depending on the tools you are using:

- o SAP Web IDE for SAP HANA

In the UI, select the application's UI module and, in the context menu, choose    **Run** **Run as** **Web Application**. When the application is up and running, SAP Web IDE displays the URL required to access the application as well as the application's current status at the bottom of the run console in the details pane:

### Tip

If the application is already running, you do not need to restart it; selecting an application module in the SAP Web IDE project pane displays the URL and status of the application in the run console:

### Output Code

```
Run tinyworld/tinyui
Updating application
Application is starting
Application is running
```

Application: <https://acme.com:51031> Status: **Running**

- o XS advanced CLI

Displays the ports allocated to applications running in a given run-time space, for example, using the `xs apps` command, as illustrated in the following example:

### Output Code

```
$>xs apps
Getting apps in org "myorg" / space "PROD" as XSA_ADMIN...
Found apps:
name          state    instances  urls
-----
----- 
htt-service    STARTED   1/1       https://acme.com:51021
```

hrtt-core	STARTED	1/1	https://acme.com:51020
hrtt-db	STOPPED	0/1	<none>
di-core-db	STOPPED	0/1	<none>
di-local-npm-registry	STARTED	1/1	https://acme.com:51025
di-core	STARTED	1/1	https://acme.com:53030
di-runner	STARTED	1/1	https://acme.com:51027
webide	STARTED	1/1	https://acme.com:53075
di-builder	STARTED	1/1	https://acme.com:51029
[...]-tinyjs	STARTED	1/1	https://acme.com:51030
[...]-tinyui	STARTED	1/1	https://acme.com:51031

## Related Information

[XS Advanced Application Resource Files \[page 110\]](#)

[Maintaining Application Services in XS Advanced \[page 660\]](#)

[Maintaining HDI Containers \[page 173\]](#)

[Maintaining Application Security in XS Advanced \[page 758\]](#)

[The XS Command-Line Interface Reference \[page 848\]](#)

[SAP Web IDE for SAP HANA Reference \[page 959\]](#)

### 3.4.1 XS Advanced Application Resource Files

SAP HANA XS advanced applications require some mandatory configuration artifacts.

In SAP HANA XS advanced, applications require a number of mandatory configuration and deployment files. Which files are required depends on the language you are using to develop the application, for example, JavaScript (including node.js) or Java, and the deployment options you need. The following table provides an overview of the artifacts required by each application. There are rules and requirements concerning where to store these artifacts in the application folder hierarchy.

Table 4: XS Advanced Application Artifacts Overview

File Name	Description	Database	App Router	JavaScript	Java
mta.yaml	MTA development descriptor	✓	✓	✓	✓
mtad.yaml	MTA deployment descriptor	✓	✓	✓	✓
package.json	Description and dependencies of node modules	✓	✓	✓	
xs-app.json	Application router description		✓		
*.hdi*	SAP HANA deployment infrastructure configuration artifacts (.hdiconfig and .hdinamespace)	✓			

File Name	Description	Database	App Router	JavaScript	Java
*.java	Java application source files				✓
*.xsjs	SAP HANA XS JavaScript source files			✓	
*.js	JavaScript (node.js) source files			✓	
xs-security.json	Shared description of OAuth2 client				

**i Note**

If you use SAP Web IDE for SAP HANA to develop an MTA, both the `mta.yaml` (**development** descriptor) and the `mtad.yaml` (**deployment** descriptor) are created automatically. The `mta.yaml` is generated when you create the application project in SAP Web IDE for SAP HANA, and the `mtad.yaml` file is created when you build the project.

The `mta.yaml` is not required if you are using command-line tools to build an MTA for XS advanced; you must specify all build and deployment dependencies in the deployment descriptor (`mtad.yaml`).

## Related Information

[Maintaining the XS Advanced Application Development & Deployment Descriptors \[page 112\]](#)

[The XS Advanced Application Descriptor \[page 721\]](#)

[The XS Advanced Application Package Descriptor \[page 653\]](#)

[The HDI Container Configuration File \[page 180\]](#)

[The HDI Name-Space Configuration File \[page 186\]](#)

[The Application Security Descriptor \[page 764\]](#)

# 4 Maintaining the XS Advanced Application Development & Deployment Descriptors

Development descriptors are used to generate deployment descriptors, which define the details required at application-deployment time.

The deployment description is contained in the application deployment “descriptor”, which specifies what you want to build (and how) as well as where (and how) to deploy it, as follows:

Table 5: MTA Development and Deployment Descriptors

File Name	Description	SAP Web IDE	MTAR Builder	XS CLI
mta.yaml	<b>Development</b> descriptor for a multi-target application (MTA). The information in the <code>mta.yaml</code> file provides instructions for the MTA development and build process.	✓	✓	-
mtad.yaml	<b>Deployment</b> descriptor for a multi-target application (MTA). The information in the <code>mtad.yaml</code> file provides instructions for the deploy service.	*	*	✓

The information defined in the development descriptor (`mta.yaml`) is used by the SAP Web IDE for SAP HANA and the MTA archive (MTAR) builder to generate the `mtad.yaml` file that is required to deploy the application to the XS advanced run-time environment. The information in the generated **deployment** descriptor `mtad.yaml` file provides the instructions that are carried out when the application is deployed.

## i Note

If you use SAP Web IDE for SAP HANA to develop an MTA, both the `mta.yaml` and the `mtad.yaml` are created automatically. The `mta.yaml` is generated when you create the application project, and this is where you specify details of the development and build process. The `mtad.yaml` deployment descriptor is created during the application project's build process.

MTAs are transported in the form of a compressed and digitally signed archive called an MTA archive or “MTAR”. Deploying an MTA involves the deployment of each application module to its corresponding target run time. The MTA archive (for example, `myMTAarchive.mtar`) that is deployed conforms to the standard Java JAR specification. For this reason, an additional file, the `MANIFEST.MF` file, is also required for the deployment operation; the `MANIFEST.MF` contains a list of the files in the MTAR archive to be deployed.

The development and deployment descriptors for an multi-target application must be located in the root folder of the application project, as illustrated (in **bold text**) in the following example.

## Example

### Sample Code

AppName

```
| - db/
|   |- package.json
|   \- src/
|- web/
|   |- package.json
|   \- resources/
|- js/
|   |- start.js
|   |- package.json
|   \- src/
|- xs-security.json
\- mta[d].yaml      # Development [deployment] build manifest *
```

### i Note

\* You only need to concern yourself with the contents of the `mtad.yaml` deployment descriptor if you are using command-line or custom tools; in these cases you must also ensure that the `mtad.yaml` is checked into version control along with the other application-project artifacts. If you are using SAP Web IDE for SAP HANA or the MTA Archive Builder, the `mtad.yaml` file is generated for you automatically and does not need to be managed with version-control tools.

## Related Information

[Create the MTA Description Files \[page 113\]](#)

[The MTA Development Descriptor \[page 117\]](#)

[The MTA Deployment Descriptor \[page 121\]](#)

## 4.1 Create the MTA Description Files

Multi-target applications are defined in multiple descriptor files.

## Context

A multi-target application (MTA) comprises multiple pieces of software called “modules” which all share a common lifecycle for development and deployment. An MTA is described in two descriptor files: the design-

time descriptor and the deployment descriptor. To create a multi-target application, perform the following steps:

## Procedure

1. Create the application resource-file structure.
2. Create the **development** descriptor for the multi-target application.

The development descriptor for an XS advanced MTA is defined in the design-time artifact `mta.yaml`, which must be placed in the root folder of your MTA project. The MTA development descriptor is used by the SAP Web IDE for SAP HANA to generate an MTA deployment descriptor (`mtad.yaml`).

### **i** Note

This step is only required if you are using the SAP Web IDE for SAP HANA to develop and deploy an MTA, and the SAP Web IDE for SAP HANA creates the `mta.yaml` file automatically when you create a new project.

If you use command-line tools to deploy an MTA, you do not need an MTA **development** descriptor. However, for this reason, you must create the MTA **deployment** descriptor (`mtad.yaml`) manually.

3. Create the **deployment** descriptor for the multi-target application.

The deployment descriptor for an XS advanced MTA is defined in the design-time artifact `mtad.yaml`, which must be placed in the root folder of your MTA project.

### **i** Note

This step is only required if you are using command-line tools to deploy an MTA. If you use the SAP Web IDE for SAP HANA to deploy an MTA, the deployment descriptor is created for you automatically from the MTA **development** descriptor (`mta.yaml`).

The deployment description of an MTA is defined in the `mtad.yaml` file, which must be located in the application's root folder.

4. If required, create a deployment extension description for the multi-target application. (optional)

The **extension** of an MTA deployment description is defined in a file with the suffix `.mtaext`, for example, `production-config.mtaext`.

## Related Information

[Multi-Target XS Advanced Applications \[page 115\]](#)

[The MTA Development Descriptor \[page 117\]](#)

[The MTA Deployment Descriptor \[page 121\]](#)

## 4.1.1 Multi-Target XS Advanced Applications

Multi-target applications collect multiple modules and resource references in a single, deployable archive.

In SAP HANA Extended Application Services, Advanced Model (XS Advanced), the applications you develop are so-called multi-target applications (MTA). An MTA is basically the natural evolution of the so-called "multi-part" application; an MTA, however, comprises multiple software "**modules**" that share a common lifecycle for development and deployment. Although these modules can be developed using different technology and languages and then deployed to different targets, they all serve (different aspects of) a particular purpose for the application users.

An MTA is any unit of deployable functionality. This can be a single module or a group of modules packaged into an MTA archive along with an MTA deployment descriptor. MTAs can be distributed across multiple containers and target platforms. The MTA approach enables you to adopt a common life-cycle process (from development to deployment) for all application artifacts to be deployed to the target platform, for example, Cloud Foundry or SAP HANA XS Advanced.

### ➔ Tip

MTAs also enable you to model and configure run-time dependencies.

You define a multi-target application in the following files:

- MTA deployment descriptor (`mtad.yaml`)
- MTA deployment **extension** descriptor (`myMTAextension.mtaext`)

### i Note

The extension descriptor is optional; it is used to provide system-, service-, or module-specific details that are not known until deployment time.

### ⚠ Restriction

For the MTA deployment operation, locally specific modifications to the deployment configuration must be defined in an MTA **deployment-extension** file.

## The MTA Model

The MTA model is a platform-independent description of the different modules that define an application, the dependencies between the modules, the configuration data the modules expose, and the resources the application modules require to run. The MTA model is specified using YAML, in descriptor files that accompany the development and deployment processes. The MTA model serves the following purposes:

1. It defines an application composed of multiple (heterogeneous) subcomponents.
2. It declares the resources that the application depends on at run time and (or) at deployment time.
3. It defines configuration variables (and their relationship), whose values distinguish different deployment scenarios for the application.

Many people find it useful to see the MTA model as a sort of formal contract between developers (using development tools) and the MTA deployer. The deployer is a tool that consumes a description of the MTA

model and translates it into target platform-specific “native” commands that can be used in the following scenarios:

- The provisioning of run-time containers
- The creation and binding of resources, for example, “service instances” on Cloud Foundry or in SAP XS advanced
- The installation, running, or update of the application modules

### **i Note**

Although the MTA model is not target-specific, the MTA deployer is multi-target aware. Indeed, even when targeting a single platform (for example, SAP XS advanced or Cloud Foundry), the MTA deployer is still needed to resolve the dependencies between the different application parts. Using an MTA deployer rather than the underlying “native” installation tools provides a single orchestrated application-deployment step which can span multiple run-time tiers on more than one target platform.

Since deployment (for example, for testing) is an integral part of the development process, development environments contain such functionality as well.

## **MTA Architecture**

An SAP HANA XS advanced application that consists of UI and database modules, and even application code, is an example of an MTA. A Java EE application composed of Beans, Web and application modules, resource adapters etc., which are all subject to the same development lifecycle and deployed across multiple computing tiers, is another example of a typical MTA. Both the application developer and the landscape administrator want to be able to manage the development, versioning, deployment, and operation of MTAs as one logical unit.

SAP Cloud Platform introduces new distribution requirements for orchestrated cross-platform deployments. When acting as a Software-as-a-Service (SaaS) extension platform and employing Fiori as a Service (Faas) concepts, application developers need to be able to distribute their applications across heterogeneous targets (for example, Java Virtual Machines (VM), the front-end server, or the SaaS back end component), each of which has its own deployment application programming interface (API), while providing a carefully managed, single application life cycle.

By adopting a model that does not require an application to depend on any formal target or technology, the MTA addresses the challenges of deploying applications to multiple targets by isolating the developer from target-specific native tools. Instead, developers describe the application’s modules, any dependencies to other modules, MTAs and (micro) services, and any required or exposed interfaces. Application lifecycle-management frameworks that are MTA-aware can validate, orchestrate, and automate the MTA deployment process both on-premise and on Cloud platforms.

## **MTA Development Tools**

SAP Web IDE provides “MTA-aware” development support both in the context of XS advanced applications and Fiori-as-a-Service (Faas) solutions. MTA-aware deployers for SAP Cloud Platform, XS advanced, and Cloud Foundry provide deployment-related services for these platforms; the services integrate with SAP

product assembly and production, are integrated into traditional SAP transport tools, and support delivery via SAP Service Market Place for on-premise deployment.

### **Restriction**

The MTA-aware tools used in each of these different use cases plan to implement the full MTA specification over a period of time. For this reason, at any point in time, each tool might reflect only a subset of the features described here; support for the features described will depend on priorities derived from the corresponding use case. For more information about the current feature scope, refer to the documentation for the respective tool.

## Related Information

[Create the MTA Description Files \[page 113\]](#)

[The MTA Development Descriptor \[page 117\]](#)

[The MTA Deployment Descriptor \[page 121\]](#)

## 4.1.2 The MTA Development Descriptor

Multi-target applications are defined in a design-time **development** descriptor.

The **development** descriptor (`mta.yaml`) is used to define the elements and dependencies of an XS advanced-compliant multi-target application (MTA), as illustrated in the following example:

### Sample Code

MTA Development Descriptor (`mta.yaml`)

```
ID: com.acme.mta.sample
version: 1.0.1
modules:
  - name: pricing-ui
    type: javascript.nodejs
    requires:
      - name: thedatabase
    - name: pricing-backend
      type: java.tomcat
    provides:
      - name: price_opt
        properties:
          protocol: http
          uri: myhost.mydomain
resources:
  - name: thedatabase
    type: com.sap.xs.hdi-container
```

### Note

The MTA development descriptor (`mta.yaml`) is used by SAP HANA GUI tools to generate the **deployment** descriptor (`mtad.yaml`) that is required to deploy an MTA to the XS advanced runtime. If you use

command-line tools to deploy an MTA, you do not need an MTA **development** description (`mta.yaml`). However, you must create the MTA deployment descriptor (`mtad.yaml`) manually.

The MTA development descriptor (`mta.yaml`) includes the following main components:

- **Global elements**  
An identifier and version that uniquely identify the MTA, plus additional **optional** information such as: a description, the providing organization, and a copyright notice for the author.
- **Modules**  
A set of code or content which needs to be deployed to a specified location.
- **Resources**  
Something required by the MTA at run time but not provided by the MTA, for example, a managed service or an instance of a user-provided service.
- **Properties**  
Key-value pairs that must be made available to the respective module at run time
- **Parameters**  
Parameters are reserved variables that affect the implementation of the MTA-aware tools, such as the deployer.

## Modules

A **module** is a set of code or content which needs to be deployed to a specified location. The `modules` section of the MTA development descriptor enables you to define a set of source artifacts of a certain type which resides in a file system and which can be addressed by a file path. The following elements are mandatory:

- `name`  
Must be unique within the MTA it identifies

Optional module attributes include: `type`, `properties`, and `parameters`, plus `requires` and `provides`.

## Order of deployment

Modules and therefore applications are deployed in an order that is based on the dependencies they have on each other. This is done in order to ensure that an application that depend on other applications is deployed only after any dependent applications are already up and running. To determine the order of application deployment and start up, the modules defined in the application development (and deployment) descriptor are sorted in such a way that the first module has the least number of (and preferably no) dependencies on other modules in the MTA, while the last one has the most dependencies. For example, the modules in the following deployment descriptor will be deployed in the order m3, m2, and finally m1:

### i Note

The number of dependencies between the modules and resources has no influence the on the deployment order; during deployment, the creation of services always takes place before any of the modules are deployed.

## Sample Code

MTA Development Descriptor (`mta.yaml`)

```
ID: com.sap.xs2.sample
version: 0.1.0
modules:
  - name: m1
    type: java.tomcat
    requires:
      - name: m2
      - name: m3
  - name: m2
    type: java.tomcat
    requires:
      - name: m3
  - name: m3
    type: java.tomcat
    requires:
      - name: r1
      - name: r2

resources:
  - name: r1
    type: com.sap.xs.uaa
  - name: r2
    type: com.sap.xs.hdi-container
```

## Circular dependencies

If two modules have a circular dependency (for example `m1` depends on `m2`, but `m2` also depends on `m1`), the parameter “dependency-type” can be specified to control the order in which the modules are deployed. Modules with parameter `dependency-type: hard` (`m1`, in the following example) are always deployed first.

### Note

The default setting for the dependency type `dependency-type: soft` for all modules. The obvious consequence of this default dependency setting is that the deployment order of modules with circular dependencies is arbitrary and cannot be relied upon.

## Sample Code

MTA Development Descriptor (`mta.yaml`)

```
ID: com.sap.xs2.sample
version: 0.1.0
modules:
  - name: m1
    type: java.tomcat
    requires:
      - name: m2
    parameters:
      dependency-type: hard
  - name: m2
    type: java.tomcat
    requires:
      - name: m1
```

## Resources

A **resource** is something which is required by the MTA at run time but not provided by the MTA, for example, a managed service or an instance of a user-provided service. In the development description's `resources` section, the following elements are mandatory:

- `name`  
Must be unique within the MTA it identifies

Optional resource attributes include: `type`, `description`, `properties`, and `parameters`. The resource `type` is one of a reserved list of resource types supported by the MTA-aware deployment tools, for example: `com.sap.xs.uaa`, `com.sap.xs.hdi-container`, `com.sap.xs.job-scheduler`; the `type` indicates to the deployer how to discover, allocate, or provision the real resource, for example, a managed service such as a database, or a user-provided service.

### Restriction

System-specific parameters for the deployment descriptor must be included in a so-called MTA deployment **extension** descriptor.

## Properties

The MTA development descriptor (`mta.yaml`) contains two types of properties: **modules** and **requires**, which are basically very similar.

The values of properties can be specified at design time, in the deployment description (`mta.yaml`). More often, though a property value is determined during deployment, where the value is either explicitly set by the administrator (for example, in an extension descriptor file) or inferred by the MTA deployer from a target platform configuration. When set, the deployer injects the property values into the module's environment. The deployment operation reports an error, if it is not possible to determine a value for a property.

Both kinds of properties ("module" and "required") are injected into the module's environment. In the "requires" section, properties can reference other properties declared in the corresponding "provides" section (for example, using the `~{}` syntax).

## Parameters

Parameters are reserved variables that affect the behavior of the MTA-aware tools, such as the deployer. Parameters can be "system" (read-only) values, "write-only", or "read-write" (default value can be overwritten). Each tool publishes a list of reserved parameters and their (default) values for its supported target environments. All parameter values can be referenced as part of other property or parameter value strings. To reference a parameter value, use the placeholder notation  `${<parameter>}` . The value of a parameter which is "read-only" cannot be changed in descriptors; only its value can be referenced using the placeholder notation.

Examples of common read-only parameters are `user`, `app-name`, `default-host`, `default-uri`. The value of a writable parameter can be specified within a descriptor. For example, a module might need to specify a

non-default value for a target-specific parameter that configures the amount of memory for the module's target run-time environment.

### Sample Code

#### Parameters and Placeholders

```
modules:
  - name: node-hello-world
    type: javascript.nodejs
    path: web/
    requires:
      - name: nodejs-uaa
      - name: nodejs
    group: destinations
    properties:
      name: backend
      url: ~{url}
      forwardAuthToken: true
  parameters:
    host: ${user}-node-hello-world
```

## Related Information

[Create the MTA Description Files \[page 113\]](#)

[The MTA Deployment Descriptor \[page 121\]](#)

### 4.1.3 The MTA Deployment Descriptor

Description of the deployment options for a multi-target XS Advanced application.

The MTA **deployment** descriptor defines the prerequisites and dependencies for the deployment of a multi-target application. The deployment description of an XS Advanced-compliant MTA is specified in an `mtad.yaml` file.

#### Note

If you use command-line tools to deploy an MTA, you must create the MTA deployment descriptor manually.

### Sample Code

#### Simple MTA Deployment Description (`mtad.yaml`)

```
ID: com.acme.mta.sample
version: 1.0.1
modules:
  - name: pricing-ui
    type: javascript.nodejs
    requires:
      - name: thedatabase
```

```

- name: pricing-backend
  type: java.tomcat
  provides:
    - name: price_opt
      properties:
        protocol: http
        uri: myhost.mydomain
  resources:
    - name: thedatabase
      type: com.sap.xs.hdi-container

```

The deployment descriptor (`mtad.yaml`) for a multi-target application comprises the following high-level components:

- **Global elements**  
An identifier and version that uniquely identify the MTA, plus additional **optional** information such as: a description, the providing organization, and a copyright notice for the author.
- **Modules**  
The application modules contained in the MTA deployment archive
- **Resources**  
The external resource that the application modules (defined in the “modules” section) depend on
- **Properties**  
Key-value pairs that must be made available to the respective module at run time
- **Parameters**  
Reserved variables that affect the behavior of the MTA-aware tools, such as the deploy service.
- **Metadata**  
Provide additional information about the declared parameters and properties.

## Modules

The `modules` section of the deployment descriptor lists the names of the application modules contained in the MTA deployment archive; the following elements are mandatory:

- `name`  
Must be unique within the MTA it identifies

Optional module attributes include: `path`, `type`, `description`, `properties`, and `parameters`, plus `requires` and `provides` lists.

## Order of deployment

Modules and therefore applications are deployed in an order that is based on the dependencies they have on each other. This is done in order to ensure that an application that depends on other applications is deployed only after any dependent applications are already up and running. To determine the order of application deployment and start up, the modules defined in the application development (and deployment) descriptor are sorted in such a way that the first module has the least number of (and preferably no) dependencies on other modules in the MTA, while the last one has the most dependencies. For example, the modules in the following deployment descriptor will be deployed in the order m3, m2, and finally m1:

## Note

The number of dependencies between the modules and resources has no influence on the deployment order; during deployment, the creation of services always takes place before any of the modules are deployed.

## Sample Code

MTA Deployment Descriptor (`mtad.yaml`)

```
ID: com.sap.xs2.sample
version: 0.1.0
modules:
  - name: m1
    type: java.tomcat
    requires:
      - name: m2
      - name: m3
  - name: m2
    type: java.tomcat
    requires:
      - name: m3
  - name: m3
    type: java.tomcat
    requires:
      - name: r1
      - name: r2

resources:
  - name: r1
    type: com.sap.xs.uaa
  - name: r2
    type: com.sap.xs.hdi-container
```

## Circular dependencies

If two modules have a circular dependency (for example `m1` depends on `m2`, but `m2` also depends on `m1`), the parameter “dependency-type” can be specified to control the order in which the modules are deployed. Modules with parameter `dependency-type: hard` (`m1`, in the following example) are always deployed first.

## Note

The default setting for the dependency type `dependency-type: soft` for all modules. The obvious consequence of this default dependency setting is that the deployment order of modules with circular dependencies is arbitrary and cannot be relied upon.

## Sample Code

MTA Deployment Descriptor (`mtad.yaml`)

```
ID: com.sap.xs2.sample
version: 0.1.0
modules:
  - name: m1
    type: java.tomcat
    requires:
      - name: m2
    parameters:
```

```
dependency-type: hard
- name: m2
  type: java.tomcat
  requires:
    - name: m1
```

## Shared Module Entries

It is possible for multiple MTA modules to reference a single deployable application, for example, a code bundle in the MTA archive. This means that, during the deployment operation, the same piece of code can be executed separately in multiple application (or application instances) but with different parameters and properties. This results in multiple, deployed run-time modules based on the same source module, which are usually started with different configuration parameters. A development project can have one source folder which is referenced by multiple module entries in the MTA deployment descriptor `mtad.yaml`, as illustrated in the following example:

### Code Syntax

Multiple MTA Module Entries in the Deployment Descriptor (`mtad.yaml`)

```
...
modules:
- name: fileloader-master
  type: nodejs
  path: js
  properties:
    FL_ROLE: master
- name: fileloader-worker
  type: nodejs
  path: js
  parameters:
    instances: 3
  properties:
    FL_ROLE: worker
...
```

If deployment is based on an MTA archive, it is not necessary to duplicate the code to have two different deployable modules; the specification for the MTA-module entry in `MANIFEST.MF` is extended, instead. The following (incomplete) example of a `MANIFEST.MF` show how to use a comma-separated list of module names to associate one set of deployment artifacts with all listed modules:

### Code Syntax

Multiple MTA Modules Listed in the `MANIFEST.MF` Deployment Manifest

```
Manifest-Version: 1.0
...
Name: js/
MTA-Module: fileloader-master, fileloader-worker
...
```

## Resources

The application modules defined in the “modules” section of the deployment descriptor depend on **resources**. In the `resources` section, the following elements are mandatory:

- `name`  
Must be unique within the MTA it identifies

Optional resource attributes include: `type`, `description`, `properties`, and `parameters`. The resource **type** is one of a reserved list of resource types supported by the MTA-aware deployment tools, for example: `com.sap.xs.uaa`, `com.sap.xs.hdi-container`, `com.sap.xs.job-scheduler`; the **type** indicates to the deployer how to discover, allocate, or provision the resource, for example, a managed service such as a database, or a user-provided service

### Restriction

System-specific parameters for the deployment descriptor must be included in a so-called MTA deployment extension descriptor.

## Properties

The MTA deployment descriptor can contain two types of properties, which are very similar, and are intended for use in the **modules** or **resources** configuration, respectively. Properties can be declared in the deployment description both in the `modules:` configuration (for example, to define `provides:` or `requires:` dependencies), or in the `resources` configuration to specify `requires:` dependencies. Both kinds of properties (`modules:` and `requires:`) are injected into the module's environment. In the `requires:` configuration, properties can reference other properties that are declared in the corresponding `provides:` configuration, for example, using the `~{ }` syntax.

The values of properties can be specified at design time, in the deployment description (`mtad.yaml`). More often, however, a property value is determined during deployment, where the value is either explicitly set by the administrator, for example, in an deployment-extension descriptor file (`myDeployExtension.mtaext`), or inferred by the MTA deployer from a target-platform configuration. When set, the deployer injects the property values into the module's environment. The deployment operation reports an error, if it is not possible to determine a value for a property.

### Tip

It is possible to declare metadata for properties defined in the MTA deployment description; the mapping is based on the parameter or property keys. For example, you can specify if a property is **required** (**optional**; `false`) or can be modified `overwritable: true`.

## Cross-References to Properties

To enable resource properties to resolve values from a property in another resource or module, a resource must declare a dependency. However, these “requires” declarations do not affect the order of the application deployment.

## Restriction

It is not possible to reference **list** configuration entries either from resources or “subscription” functionalities (deployment features that are available to subscriber applications).

## Code Syntax

Cross-References between Properties in the MTA Deployment Descriptor

```
modules:
  - name: java
    ...
    provides:
      - name: backend
        properties:
          url: ${default-url}/foo
resources:
  - name: example
    type: example-type
    properties:
      example-prop: my-example-prop
  - name: uaa
    type: uaa-type
    requires:
      - name: example
      - name: backend
        properties:
          prop: ~{url}
        parameters:
          param: ~{url}
    properties:
      pro1: ~{example/example-prop}
  parameters:
    config:
      app-router-url: ~{backend/url} # reference via a module's provides
section
  example-prop: ~{example/example-prop} # reference a resource's
property
```

## Parameters and Placeholders

Parameters are reserved variables that affect the behavior of the MTA-aware tools, such as the deployer. Parameters can be “system”, “write-only”, or “read-write” (default value can be overwritten). Each tool publishes a list of system parameters and their (default) values for its supported target environments. All parameter values can be referenced as part of other property or parameter value strings. To reference a parameter value, use the placeholder notation  `${<parameter>}`. The value of a system parameter cannot be changed in descriptors. Only its value can be referenced using the placeholder notation.

Examples of common read-only parameters are `user`, `app-name`, `default-host`, `default-uri`. The value of a writable parameter can be specified within a descriptor. For example, a module might need to specify a non-default value for a target-specific parameter that configures the amount of memory for the module’s runtime.

## ➔ Tip

It is also possible to declare metadata for parameters and properties defined in the MTA deployment description; the mapping is based on the parameter or property keys. For example, you can specify if a parameter is **required** (optional; false) or can be modified **overwritable: true**.

## Sample Code

### Parameters and Placeholders

```
modules:
  - name: node-hello-world
    type: javascript.nodejs
    path: web/
    requires:
      - name: nodejs-uaa
      - name: nodejs
        group: destinations
    properties:
      name: backend
      url: ~{url}
      forwardAuthToken: true
  parameters:
    host: ${user}-node-hello-world
```

Descriptors can contain so-called placeholders (also known as substitution variables), which can be used as sub-strings within property and parameter values. Placeholder names are enclosed by the dollar sign (\$) and curly brackets ({}). For example: \${host} and \${domain}. For each parameter “P”, there is a corresponding placeholder \${P}. The value of <P> can be defined either by a descriptor used for deployment, or by the deploy service itself.

## ➔ Tip

Placeholders can also be used without any corresponding parameters; in this scenario, their value cannot be overridden in a descriptor. Such placeholders are Read only.

Placeholders can be used to read the value of parameters. For example, the following placeholder can be used in a descriptor to get the CF / XS API URL: \${xs-api-url}. Placeholders can also be used in map and list values in properties and parameters sections of modules and resources, as illustrated in the following example:

## Sample Code

```
resources:
  - name: uaa
    type: com.sap.xs.uaa
    parameters:
      config:
        users: [ "${generated-user}", "XSMASTER" ]
```

## Metadata for Properties and Parameters

It is possible to declare metadata for parameters and properties defined in the MTA deployment description, for example, using the “parameters-metadata:” or “properties-metadata:” keys, respectively; the

mapping is based on the keys defined for a parameter or property. You can specify if a property is required (optional: false) or can be modified (overwritable: true), as illustrated in the following (incomplete) example:

The overwritable: and optional keywords are intended for use in extension scenarios, for example, where a value for a parameter or property is supplied at deployment time and declared in a deployment-extension descriptor file (myMTADeploymentExtension.mtaext).

You can declare metadata for the parameters and properties that are already defined in the MTA deployment description. However, any parameters or properties defined in the mtad.yaml deployment descriptor with the metadata value overwritable: false cannot be overwritten by a value supplied from the extension descriptor. In this case, an error would occur in the deployment.

## Code Syntax

### Metadata for MTA Deployment Parameters and Properties

```
modules:
  - name: frontend
    type: javascript.nodejs
  parameters:
    memory: 128M
    domain: ${default-domain}
  parameters-metadata:
    memory:
      optional: true
      overwritable: true
    domain:
      overwritable: false
  properties:
    backend_types:
      order_management: sap-erp
      data_warehouse: sap-bw
  properties-metadata:
    backend_types:
      overwritable: false
      optional: false
```

### Note

Parameters or properties can be declared as sensitive. Information about properties or parameters flagged as "sensitive" is not written as plain text in log files; it is masked, for example, using a string of asterisks (\*\*\*\*\*).

## Example Deployment Description File

In many cases, an MTA will consist of multiple modules, which have interdependencies. The following, slightly more complex example shows an MTA deployment description including three modules:

- A database model
- An SAP UI5 application (hello world)
- An application written in node.js

In this example, the UI5 application, “hello-world”, will use the environment-variable `<ui5_library>` as a logical reference to some version of UI5 on a public Website.

## Sample Code

Complex MTA Deployment Description (`mtad.yaml`)

```
ID: com.acme.xs2.samples.javahelloworld
version: 0.1.0
modules:
  - name: hello-world
    type: javascript.nodejs
    requires:
      - name: uaa
      - name: java_details
        properties:
          backend_url: ~{url3}/
    properties:
      ui5_library: "https://sapui5.hana.acme.com/"

  - name: java-hello-world-backend
    type: java.tomee
    requires:
      - name: uaa
      - name: java-hello-world-db           # deploy ordering
      - name: java-hdi-container
    provides:
      - name: java_details
        properties:
          url3: ${url}                      # the value of the place-holder ${url}
                                              # will be made known to the deployer

  - name: java-hello-world-db
    type: com.sap.xs.hdi
    requires:
      - name: java-hdi-container
resources:
  - name: java-hdi-container
    type: com.sap.xs.hdi-container

  - name: java-uaa
    type: com.sap.xs.uaa
    parameters:
      name: java-uaa                     # the name of the actual service
```

The example describes three modules, which define a specific deployment order: `hello-world` requires `java-details`, which is provided by `java-hello-world-backend`, which in turn requires `java-hello-world-db`. These modules depend on a several resources, which are provisioned by the deployer as service instances in XS advanced, for example a database service instance `java-hdi-container` and a user authentication and authorization (UAA) service instance `java-uaa`. `uaa` is an existing service that provides the `hello_world` application with an OAuth user and password in order to enable XS Advanced to support authentication and authorization for this application. If your application does not require authentication services, you do not need to include this type of dependency.

The `hello_world` UI application will need to know the “access-point” for the `java-hello-world-backend` module. However, since this type of URL is often platform-specific, the UI application uses a property value (`backend_url` in this example) which is evaluated by the deployer as a value-substitution of the variable `<~{url3}>`; the variable `<~{url3}>` is provided as a property of the backend module `java-hello-world-backend`.

### Note

The value of `url3` is specified via the system parameter `default-url`, whose value is known to the deployer at deployment time.

## Cross-MTA dependencies

In addition to having dependencies on modules in the same MTA, modules can have dependencies on modules from **other** MTAs, too. For these so-called cross-MTA dependencies to work, the MTA that **provides** the dependencies must declare them as “public” (this is true by default). There are two methods that you can use to declare that one module has a dependency on a module in a different MTA:

- `mta-provided` [page 130]
- `configuration` [page 131]

### Note

Both declaration methods require the addition of a resource in the deployment descriptor; the additional resource defines the provided dependency from the other MTA.

### Cross MTA Dependency Method 1: Resource Type “`mta-provided`”

The resource defined in this method contains the special parameters `mta-id` (the ID of the provider MTA), `mta-version` (the version of the provider MTA) and `mta-provides-dependency` (the name of the provides dependency in the provider MTA). In addition, the `mta-version` parameter can contain version ranges as described in the Semver specification.

You can declare metadata for the parameters and properties defined in the MTA deployment. Bear in mind that this method can only be used for cross-MTA dependency resolution, and does not allow the specification of the organization and space in which the “providing” MTA resides. As a result, a provided dependency will be “found” by this syntax, only if it is published by an MTA that is deployed in the same space as the consuming MTA.

### Caution

Because of the limitations listed above, it is not recommended to use the “`mta-provided`” dependency-declaration method; whenever possible, use the alternative declaration method (“`configuration`”) instead.

The following example shows the dependency declaration in the deployment descriptor of the “consumer” MTA :

### Sample Code

Consumer MTA Deployment Descriptor (`mtad.yaml`)

```
_schema-version: "2.0.0"
ID: com.sap.sample.mta.consumer
version: 0.1.0
modules:
```

```

- name: consumer
  type: java.tomee
  requires:
    - name: message-provider
      properties:
        message: ~{message}
  resources:
    - name: message-provider
      type: mta-provided
      parameters:
        mta-id: com.sap.sample.mta.provider
        mta-version: ">=1.0.0"
        mta-provides-dependency: message-provider

```

The following example shows the dependency declaration in the deployment descriptor of the “provider” MTA :

### Sample Code

Provider MTA Deployment Descriptor (`mtad.yaml`)

```

schema-version: "2.0.0"
ID: com.sap.sample.mta.provider
version: 2.3.0
modules:
  - name: provider
    type: javascript.nodejs
    provides:
      - name: message-provider
        public: true
        properties:
          message: "Hello! This is a message provided by application \"${app-name}\", deployed in org \"${org}\" and space \"${space}\\"!"

```

## Cross-MTA Dependency Method 2: Resource Type “configuration”

This method can be used to access any entry that is present in the configuration registry. The parameters used in this cross-MTA declaration method are `provider-nid`, `provider-id`, `version`, and `target`. The parameters are all optional and are used to filter the entries in the configuration registry based on their respective fields. If any of these parameters is not present, the entries will not be filtered based on their value for that field. The `version` parameter can accept any valid Semver ranges.

When used for cross-MTA dependency resolution the `provider-nid` is always “`mta`”, the `provider-id` follows the format `<mta-id>:<mta-provides-dependency-name>` and the `version` parameter is the version of the provider MTA. In addition, as illustrated in the following example, the `target` parameter is structured and contains the name of the organization and space in which the provider MTA is deployed. In the following example, the placeholders  `${org}` and  `${space}` are used, which are resolved to the name of the organization and space of the consumer MTA. In this way, the provider MTA is deployed in the same space as the consumer MTA.

### Note

By default, all provided dependencies are public.

The following example shows the dependency declaration in the deployment descriptor of the “consumer” MTA :

### Sample Code

Consumer MTA Deployment Descriptor (`mtad.yaml`)

```
schema-version: "2.0.0"
ID: com.sap.sample.mta.consumer
version: 0.1.0
modules:
  - name: consumer
    type: java.tomee
    requires:
      - name: message-provider
        properties:
          message: ~{message}
resources:
  - name: message-provider
    type: configuration
    parameters:
      provider-nid: mta
      provider-id: com.sap.sample.mta.provider:message-provider
    version: ">=1.0.0"
    target:
      org: ${org}      # Specifies the org of the provider MTA
      space: ${space} # Wildcard * searches in all spaces
```

### ➔ Tip

If no target organization or space is specified by the consumer, then the current organization and space are used to deploy the provider MTA. If you specify a wildcard value (\*) for organization or space of the provider MTA, the provider would be searched in all organization or spaces for which the wildcard value is provided. If no match is found for the provider MTA in the specified target organization or space, then `org: <current-org>` and `space: SAP` is searched.

The following example shows the dependency declaration in the deployment descriptor of the “provider” MTA :

### Sample Code

Provider MTA Deployment Descriptor (`mtad.yaml`)

```
_schema-version: "2.0.0"
ID: com.sap.sample.mta.provider
version: 2.3.0
modules:
  - name: provider
    type: javascript.nodejs
    provides:
      - name: message-provider
        public: true
        properties:
          message: "Hello! This is a message provided by application \"${app-name}\", deployed in org \"${org}\" and space \"${space}\\"!"
```

## Plugins

The deployment service also supports a method that allows an MTA to consume **multiple** configuration entries per `requires` dependency; the configuration is provided in a `list` property, as illustrated in the following example:

### Posting Instructions

#### Sample Code

Multiple “requires” Dependencies in the MTA Deployment Descriptor (`mtad.yaml`)

```
schema-version: "2.1"
ID: com.acme.framework
version: "1.0" modules:
  - name: framework
    type: javascript.nodejs
    requires:
      - name: plugins
        list: plugin_configs
        properties:
          plugin_name: ~{name}
          plugin_url: ~{url}/sources
        parameters:
          managed: true # true | false. Default is false
resources:
  - name: plugins
    type: configuration
    parameters:
      target:
        org: ${org}
        space: ${space}
    filter:
      type: com.acme.plugin
```

The MTA deployment descriptor shown in the example above contains a module that specifies a “`requires`” dependency to a configuration resource. Since the `requires` dependency has a `list` property, the deploy service will attempt to find multiple configuration entries that match the criteria specified in the configuration resource.

### Tip

It is possible to create a subscription for a **single** configuration entry, for example, where no “`list:`” element is defined in the required dependency.

The resource itself contains a `filter` parameter that is used to filter entries from the configuration registry based on their content. In the example shown above, the filter only matches entries that are provided by an MTA deployed in the current space, which have a `type` property in their content with a value of `com.acme.plugin`.

### Note

The filter parameter can be used in combination with other configuration resource specific parameters, for example: `provider-nid`, `provider-id`, `target`, and `version`.

If the “list” element is missing, the values matched by the resources filter are **single** configuration entries – not the usual list of multiple configuration entries. In addition, if either no value or multiple values are found during the deployment of the consuming (subscribing) MTA, the deployment operation fails. If a provider (plug-in) contributes additional configuration details after subscriber applications have been deployed, the subscriber applications will not receive the new information immediately; they will be made aware of the new configuration details only when they are updated. Note, however, that the update operation will fail because multiple configuration entries will be available at that point.

The XML document in the following example shows some sample configuration entries, which would be matched by the filter if they were present in the registry.

## Sample Code

### MTA Configuration Entries Matched in the Registry

```
<configuration-entry>
  <id>8</id>
  <provider-nid>mta</provider-nid>
  <provider-id>com.sap.sample.mta.plugin-1:plugin-1</provider-id>
  <provider-version>0.1.0</provider-version>
  <target-space>2172121c-1d32-441b-b7e2-53ae30947ad5</target-space>
  <content>{"name": "plugin-1", "type": "com.acme.plugin", "url": "https://xxx.mo.sap.corp:51008"}</content>
</configuration-entry>
<configuration-entry>
  <id>10</id>
  <provider-nid>mta</provider-nid>
  <provider-id>com.sap.sample.mta.plugin-2:plugin-2</provider-id>
  <provider-version>0.1.0</provider-version>
  <target-space>2172121c-1d32-441b-b7e2-53ae30947ad5</target-space>
  <content>{"name": "plugin-2", "type": "com.acme.plugin"}</content>
</configuration-entry>
```

The JSON document in the following example shows the environment variable that will be created from the `requires` dependency defined in the example deployment descriptor above, assuming that the two configuration entries shown in the XML document were matched by the filter specified in the configuration resource.

## Note

References to non-existing configuration entry content properties are resolved to “`null`”. In the example above, the configuration entry published for `plugin-2` does not contain a `url` property in its content. As a result, the environment variable created from that entry is set to “`null`” for `plugin_url`.

## Sample Code

### Application Environment Variable

```
plugin_configs: [
  {
    "plugin_name": "plugin-1",
    "plugin_url": "https://xxx.mo.sap.corp:51008/sources"
  },
  {
    "plugin_name": "plugin-2",
    "plugin_url": null
  }
]
```



Requires dependencies support a special parameter named “`managed`”, which registers as a “subscriber” the application created from the module containing the `requires` dependency. One consequence of this registration is that if any new configuration entries are published in the configuration registry during the deployment of another MTA, and those new entries match the filter specified in the subscription of an application, then that application’s environment would be updated, and the application itself would be restarted in order for it to see its new environment’s state.

#### ➔ Tip

When starting the deployment of an MTA (with the `xs deploy` command), you can use the special option `--no-restart-subscribed-apps` to specify that, if the publishing of configuration entries created for that MTA result in the update of a subscribing application’s environment, then that application should **not** be restarted.

## Service-Creation Parameters

Some services support additional configuration parameters in the `create-service` request; these parameters are passed in a valid JSON object containing the service-specific configuration parameters. The `deploy` service supports the following methods for the specification of service creation parameters:

- Method 1  
An entry in the MTA deployment descriptor (or the extension)
- Method 2  
A JSON file with the required service-configuration parameters

#### i Note

If service-creation information is supplied both in the deployment (or extension) descriptor **and** in a supporting JSON file, the parameters specified directly in the deployment (or extension) descriptor override the parameters specified in the JSON file.

## Service-Creation Parameters in the MTA Deployment Descriptor

The following example shows how to define the service-configuration parameters in the MTA deployment descriptor (`mtad.yaml`). If you use this method, all parameters under the special `config` parameter are used for the service-creation request.

#### Sample Code

##### Service-Configuration Parameters in the MTA Deployment Descriptor

```
resources:  
  - name: java-uaa  
    type: com.sap.xs.uaa  
    parameters:  
      config:  
        xsappname: java-hello-world
```

## Service-Creation Parameters in a JSON File

The following example shows how to define the service-configuration parameters for a service-creation request in a JSON file; with this method, there are dependencies on further configuration entries in other configuration files. For example, if you use this JSON method, an additional entry must be included in the `MANIFEST.MF` file which defines the path to the JSON file containing the parameters as well as the name of the resource for which the parameters should be used.

### Sample Code

#### Service-Configuration Parameters in a JSON File

```
resources:
- name: java-uaa
  type: com.sap.xs.uaa
  parameters:
    config:
      xsappname: java-hello-world
```

The MTA's security descriptor (`xs-security.json`) should contain the following:

### Sample Code

#### xs-security.json File

```
{
  "xsappname": "java-hello-world"
}
```

The application manifest (`MANIFEST.MF`) should contain the following:

### Sample Code

#### Service-Configuration Parameters in the MANIFEST.MF

```
Name: xs-security.json
MTA-Resource: java-uaa
Content-Type: application/json
```

The following example of an MTA deployment descriptor shows how to combine both methods to achieve the desired application-service creation on deployment:

### Sample Code

#### Combined Service-Configuration Parameters in the MTA Deployment Descriptor

```
resources:
- name: java-uaa
  type: com.sap.xs.uaa
  parameters:
    config:
      xsappname: java-hello-world
```

## Service-Binding Parameters

Some services support additional configuration parameters in the `create-bind` request; these parameters are passed in a valid JSON object containing the service-specific configuration parameters. The deployment service supports the following methods for the specification of service-binding parameters:

- Method 1
  - An entry in the MTA deployment descriptor (or the extension)
- Method 2
  - A JSON file with the required service-configuration parameters

### i Note

If service-binding information is supplied both in the MTA's deployment (or extension) descriptor **and** in a supporting JSON file, the parameters specified directly in the deployment (or extension) descriptor override the parameters specified in the JSON file.

In the MTA deployment descriptor, the `requires` dependency between a module and a resource represents the binding between the corresponding application and the service created from them (if the service has a `type`). For this reason, the `config` parameter is nested in the `requires` dependency parameters, and a distinction must be made between the `config` parameter in the `modules` section and the `config` parameter used in the `resources` section (for example, when used for service-creation parameters).

## Service-Binding Parameters in the MTA Deployment Descriptor

The following example shows how to define the service-binding parameters in the MTA deployment descriptor (`mtad.yaml`). If you use this method, all parameters under the special `config` parameter are used for the service-bind request.

### Sample Code

#### Service-Binding Parameters in the MTA Deployment Descriptor

```
modules:  
  - name: node-hello-world-backend  
    type: javascript.nodejs  
    requires:  
      - name: node-hdi-container  
    parameters:  
      config:  
        permissions: debugging
```

## Service-Binding Parameters in a JSON File

The following example shows how to define the service-binding parameters for a service-bind request in a JSON file; with this method, there are dependencies on entries in other configuration files. For example, if you use this JSON method, an additional entry must be included in the `MANIFEST.MF` file which defines the path to the JSON file containing the parameters as well as the name of the resource for which the parameters should be used.

## Sample Code

Service-Binding Parameters in a JSON File

```
modules:
  - name: node-hello-world-backend
    type: javascript.nodejs
    requires:
      - name: node-hdi-container
        parameters:
          config-path: xs-hdi.json
```

The MTA's `xs-hdi.json` file should contain the following:

## Sample Code

`xs-hdi.json` File

```
{
  "permissions": "debugging"
}
```

The application manifest (`MANIFEST.MF`) should contain the following:

### Note

To avoid ambiguities, the name of the module is added as a prefix to the name of the `requires` dependency; the name of the manifest attribute uses the following format: `<module-name>#<requires-dependency-name>`.

## Sample Code

Service-Binding Parameters in the `MANIFEST.MF`

```
Name: xs-hdi.json
MTA-Requires: node-hello-world-backend#node-hdi-container
Content-Type: application/json
```

The following example of an MTA deployment descriptor shows how to combine both methods to achieve the desired application-service binding on deployment:

## Sample Code

Combined Service-Binding Parameters in the MTA Deployment Descriptor

```
modules:
  - name: node-hello-world-backend
    type: javascript.nodejs
    requires:
      - name: node-hdi-container
        parameters:
          config:
            permissions: debugging
```

## Service Keys

You can use the `service-keys` parameter in the MTA deployment descriptor to configure the deployment service to create and update the corresponding service keys during the creation or update of a service instance. The special `service-keys` parameter must be added to the `resources` in the MTA deployment descriptor which represent services that support service keys.

The following example shows how to set up the configuration of the service keys in the applications deployment descriptor (`mtad.yaml`) file. Each entry for the `service-keys` parameter can include optional configuration parameters, which must be placed under the `config` parameter.

### Sample Code

#### Service-Key Configuration in the MTA Deployment Descriptor

```
resources:
  - name: node-hdi-container
    type: com.sap.xs.hdi-container
    parameters:
      service-keys:
        - name: tool-access-key
          config:
            permissions: read-write
        - name: reader-endpoint
          config:
            permissions: read-only
```

## Service Tags

Some services provide a list of tags that are later added to the `<VCAP_SERVICES>` environment variable. These tags provide a more generic way for applications to parse `<VCAP_SERVICES>` for credentials. You can provide custom tags when creating a service instance, too. To inform the deployment service about custom tags, you can use the special `service-tags` parameter, which must be located in the `resources` definition that represent the managed services, as illustrated in the following example:

### Sample Code

#### Defining Service Tags in the MTA Deployment Descriptor

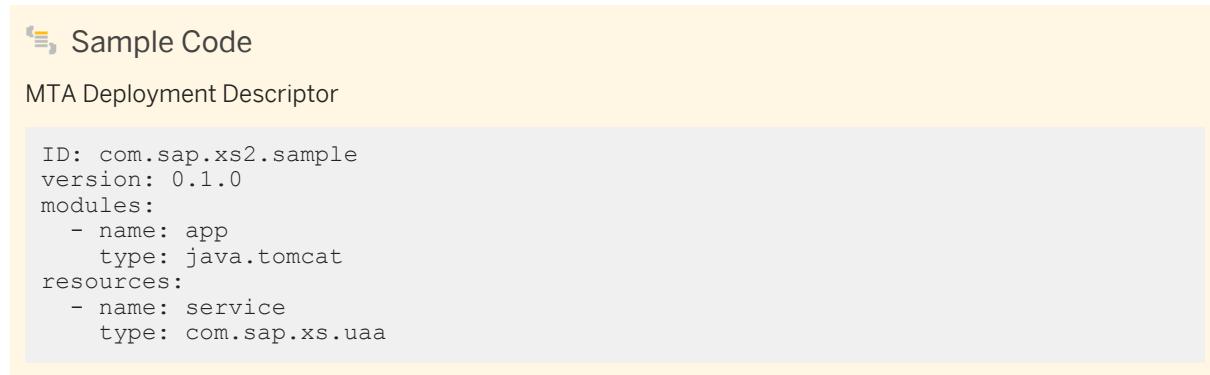
```
resources:
  - name: nodejs-uaa
    type: com.sap.xs.uaa
    parameters:
      service-tags: ["custom-tag-A", "custom-tag-B"]
```

### i Note

Some service tags are inserted by default, for example, `xsuaa`, for the XS User Account and Authentication (UAA) service.

## Namespaces

To prevent name clashes for applications and services contained in different MTAs, but deployed in the same space, the deployment service provides an option that enables you to add the MTA IDs in front of the names of the applications and services contained in those MTAs. For example, an application named “com.sap.xs2.sample.app” and a service named “com.sap.xs2.sample.service” are created when the --use-namespaces option is specified during the deployment of an MTA with the following deployment descriptor:



**Sample Code**

MTA Deployment Descriptor

```
ID: com.sap.xs2.sample
version: 0.1.0
modules:
  - name: app
    type: java.tomcat
resources:
  - name: service
    type: com.sap.xs.aaa
```

By default the “use namespaces” feature is disabled. However, specifying the --use-namespaces option as part of the xs deploy command allows you to enable it. If you want namespaces to be used for applications, but not for services, you can use the --no-namespaces-for-services option in combination with the --use-namespaces. For example, if you use the --use-namespaces and --no-namespaces-for-services options when deploying an MTA with the deployment descriptor shown in the sample code above, the deployment operation creates an application named “com.sap.xs2.sample.app” and a service named “service”.

## Application Versions and Deployment Consistency

The deployment service follows the rules specified in the Semantic Versioning Specification (Semver) when comparing the versions of a deployed MTA with the version that is to be deployed. If an MTA submitted for deployment has a version that is lower than or equal to an already deployed version of the same MTA, the deployment might fail if there is a conflict with the version rule specified in the deployment operation. The version rule is a parameter of the deployment process, which specifies which relationships to consider between the existing and the new version of an MTA before proceeding with the deployment operation. The version rules supported by the deploy service are as follows:

- **HIGHER**  
Only MTAs with versions that are bigger than the version of the currently deployed MTA, are accepted for deployment.
- **SAME\_HIGHER**  
Only MTAs with versions that are higher than (or the same as) the version of the currently deployed MTA are accepted for deployment. This is the **default** setting for the version rule.
- **ALL**  
All versions are accepted for deployment.

## Registration of Service URLs

An application can declare a service URL for automatic registration with the XS advanced controller; the URL is made available so that other applications can find out where (which URL) the application can be accessed. The following example shows the parameters to use in the MTA's deployment (or extension) descriptor to ensure that the corresponding application specifies that a service URL should be registered as part of its deployment process:

### → Tip

You can use placeholders \${ } in the service-URL declaration.

### Sample Code

```
modules:
  - name: jobscheduler-dashboard
    type: javascript.nodejs
    parameters:
      register-service-url: true
      service-name: job-scheduler-service-dashboard
      service-url: ${default-url}
```

## Service-Broker Creation

Service brokers are applications that advertise a catalog of service offerings and service plans, as well as interpreting calls for creation, binding, unbinding, and deletion. The deploy service supports automatic creation (and update) of service brokers as part of an application deployment process. For example, an application can declare that a service broker should be created as part of its deployment process, by using the following parameters in its corresponding module in the MTA deployment (or extension) descriptor:

### → Tip

You can use placeholders \${ } in the service-URL declaration.

### Sample Code

```
- name: jobscheduler-broker
  properties:
    user: ${generated-user}
    password: ${generated-password}
  parameters:
    create-service-broker: true
    service-broker-name: jobscheduler
    service-broker-user: ${generated-user}
    service-broker-password: ${generated-password}
    service-broker-url: ${default-url}
```

The value of the "\${generated-user}" and "\${generated-password}" placeholders in the properties section is the same as in the parameters section. The service-broker application uses this mechanism to inject the user-password credentials.

The `create-service-broker` parameter should be set to true if a service broker must be created for the specified application module. You can specify the name of the service broker with the `service-broker-name` parameter; the default value is  `${app-name}`. The `service-broker-user` and `service-broker-password` are the credentials that will be used by the controller to authenticate itself to the service broker in order to make requests for creation, binding, unbinding and deletion of services. The `service-broker-url` parameter specifies the URL on which the controller will send these requests.

### Note

During the creation of the service broker, the XS advanced controller makes a call to the service-broker API to inquire about the services and plans the service broker provides. For this reason, an application that declares itself as a service broker must implement the service-broker application-programming interface (API). Failure to do so might cause the deployment process to fail.

## Module-Based User-Provided Service Creation

SAP HANA XS advanced allows you to expose an application as a user provided service which other applications can bind to. The following example of the application's the MTA deployment (or extension) descriptor shows the syntax required:

### Sample Code

```
modules:
  - name: provider
    properties:
      userID: ${generated-user}
      password: ${generated-password}
    parameters:
      create-user-provided-service: true
      user-provided-service-name: technical-user-provider
      user-provided-service-config:
        userID: ${generated-user}
        password: ${generated-password}
        url: ${default-url}
```

The `create-user-provided-service` parameter must be set to "true" if you want a user-provided service to be created for the specified module. In addition, you can specify custom parameters for the application (for example, `userID`, `password`, or `url`); these custom parameters are used for the creation of the user-provided service. The custom parameters are exposed to any other applications that bind to the user-provided service that is created during the application-deployment process.

### Restriction

The names of environment variables already defined at the platform level (for example, `<XS_*`) cannot be used to define the name of a custom parameter, too. In addition, since "user" is the (protected) name of a platform environment variable, it cannot be used as the name of a custom parameter. To specify the name of a user in a custom parameter, use "userID", "username", or something similar instead.

## Alternative Grouping of MTA Properties

The deploy service supports the extension of the standard syntax for references in module properties; this extension enables you to specify the name of the `requires` section inside the property reference. You can use this syntax extension to declare implicit groups, as illustrated in the following example:

### Sample Code

#### Syntax Extension: Alternative Grouping of MTA Properties

```
modules:
  - name: pricing-ui
    type: javascript.nodejs
    properties:
      API: # equivalent to group, but defined in the module properties
        - key: internal
          protocol: ~{price_opt/protocol} #reference to value of protocol
    defined in price_opt of module pricing-backend
        - key: external
          url: ~{competitor_data/url} # reference to string value of property
    'url' in required resource 'competitor_data'
          api_keys: ~{competitor_data/creds} # reference to list value of
    property 'creds' in 'competitor_data'
        requires:
          - name: competitor_data
          - name: price_opt
    - name: pricing-backend
      type: java.tomcat
      provides:
        - name: price_opt
          properties:
            protocol: http ...
    resources:
      - name: competitor_data
        properties:
          url: "https://marketwatch.acme.com/"
          creds:
            app_key: 25892e17-80f6
            secret_key: cd171f7c-560d
```

## MTA Deployment from a Directory

It is possible to deploy an MTA directly from directory, provided that directory structure is suitably defined. When deploying an MTA from a directory structure, the deploy service's command-line client (`xs deploy`) builds an MTA archive (`myMTA.mtar`) as a first step and then uses the prepared archive to complete the deployment operation. To ensure a successful deployment of an MTA from a directory structure, the following prerequisites apply:

- `mtad.yaml`  
The `mtad.yaml` file should be located in the root of the MTA directory one would like to deploy.
- `MANIFEST.MF`  
To enable the deploy service to create a valid `MANIFEST.MF` entry for each MTA element with the associated content, you must ensure that the conditions listed in the following table are met:

Table 6: Required Entries in the MANIFEST.MF File for Directory Deployment

Element with Associated Content	Element/Parameter	Example mtad.yaml	Manifest Entry
MTA module	Element: path	<pre>modules: - name: jobscheduler-backend   type: javascript.nodejs   path: backend/</pre>	Name: backend/ MTA-Module: jobscheduler-backend
MTA resource that requires creation configuration specified in an external file	Parameter: config-path	<pre>- name: jobscheduler-uaa   type: com.sap.xs.uaa   parameters:     config-path: security/xs-security.json</pre>	Name: security/xs-security.json MTA-Resource: jobscheduler-uaa Content-Type: application/json
Module required dependency, pointing to the resource that needs binding configuration specified in an external file	Parameter: config-path	<pre>modules: - name: backend   requires: db   parameters:     config-path: cfg/backend-db-params.json</pre>	Name: cfg/backend-db-params.json MTA-Requires: backend/db Content-Type: application/json

### i Note

The values specified for `path` and `config-path` values should be the **relative** path from the root directory of the MTA to be deployed to the file or directory containing the specified content.

## Related Information

[MTA Deployment Descriptor Syntax \[page 145\]](#)

[Multi-Target XS Advanced Applications \[page 115\]](#)

[Create the MTA Description Files \[page 113\]](#)

### 4.1.3.1 MTA Deployment Descriptor Syntax

Description of an MTA's deployment-related prerequisites and dependencies.

#### Sample Code

MTA Deployment Description (`mtad.yaml`)

```
ID: com.sap.xs2.samples.nodehelloworld
version: 0.1.0
modules:
  - name: node-hello-world
    type: javascript.nodejs
    path: web/
    requires:
      - name: nodejs-uaa
      - name: nodejs
        group: destinations
    properties:
      name: backend
      url: ~{url}
      forwardAuthToken: true
    properties-metadata:
      name:
        optional: false
        overwritable: false
      url:
        overwritable: false
    parameters:
      host: !sensitive ${user}-node-hello-world
      memory: 128MB
    parameters-metadata:
      memory:
        optional: true
        overwritable: true
  - name: node-hello-world-backend
    type: javascript.nodejs
    path: js/
    provides:
      - name: nodejs
        properties:
          url: "${default-url}"
    requires:
      - name: nodejs-uaa
      - name: nodejs-hdi-container
      - name: node-hello-world-db
    parameters:
      host: ${user}-node-hello-world-backend
  - name: node-hello-world-db
    type: com.sap.xs.hdi
    path: db/
    requires:
      - name: nodejs-hdi-container
    parameters:
      - tasks:
          - name: task-1
            command: node deploy.js
            env:
              env1: value1
resources:
  - name: nodejs-hdi-container
    type: com.sap.xs.hdi-container
    parameters:
      config:
        schema: ${default-container-name}
```

```
- name: nodejs-uaa
  type: com.sap.xs.uaa
  parameters:
    config_path: xs-security.json
- name: log
  type: application-logs
  optional: true
```

## ID

Use the `ID` property to specify a unique identifier for the MTA. For example, the ID could be a reverse-URL dot-notation string that uniquely identifies the run-time namespace of the multi-target application to be deployed.

### Code Syntax

```
ID: com.acme.mta.samples.javahelloworld
```

## version

Use the `version` property to specify a version of the MTA defined in the `mtad.yaml` deployment descriptor.

The version value must follow the standard format for semantic versioning, for example, `2.0.0 (<major>. <minor>. <patch>)`. The restriction ensures a reliable contract between a development environment and the deployed system components.

### Sample Code

```
"version": "<Major_Version>.<Minor_Version>.<Patch_Version>"
```

### i Note

It is also possible to declare a “partial” or incomplete version, for example, “`1`” or “`1.3`”.

You can specify all the information used to define the MTA version, for example:

“`<Major_Version>.<Minor_Version>.<Patch_Version>`”, or any combination of the information, for example, `<Major_Version>.<Minor_Version>`, or simply `<Major_Version>`. If the version declaration does not specify a value for the `<Minor_Version>` or the `<Patch_Version>`, the deployer assumes the most recent known minor or patch version.

## modules

modules section of the MTA deployment descriptor, you can define a set of modules of a certain type, the contents of which reside in the MTA archive or in a file system.

### Sample Code

#### Module Definition in the MTA Deployment Description

```
modules:
  - name: node-hello-world
    type: javascript.nodejs
    path: web/
    requires:
      - name: nodejs-uaa
      - name: nodejs
      group: destinations
    properties:
      name: backend
      url: ~{url}
      forwardAuthToken: true
  parameters:
    host: ${user}-node-hello-world
  tasks:
    - name: task-1
      command: node deploy.js
      env:
        env1: <value1>
        env1: <value2>
    - name: task-2
```

### Tip

It is also possible to declare metadata for parameters and properties defined in the MTA deployment description; the mapping is based on the parameter or property keys. For example, you can specify if a parameter is **required** (optional; false) or can be modified `overwritable: true`.

Use the following attributes to define each new dependent module:

Table 7: MTA Deployment Descriptor Modules Keys

Attribute	Description	Example Value
name	Name of the module to be deployed	java-hello-world-backend
type	Determines the run-time to which the specified module is deployed.	
path	The file-system path relative to the root of the MTA directory tree.	web/
group	Combine properties from multiple providers into one look-up object (for example, an environment variable). This reduces the number of look ups performed by the code of the requiring module.	DESTINATIONS API

Attribute	Description	Example Value
provides	Used to specify the names of <code>provides</code> sections, each containing configuration data; the data <b>provided</b> can be <b>required</b> by other modules in the same MTA.	< <code>provides_section_name</code> >
requires	<p>Used to specify the names of “<code>requires</code>” sections that are provided “resource” that has been declared for the same MTA. Tools check if all required names are provided within the MTA. The deployment mechanism for the requiring module will access properties of the matching <code>providesMetadata</code> for MTA Deployment Parameters and section and provide them to the module runtime, for example, in environment variables.</p> <p>The <code>requires</code> section can declare a <b>group</b> assignment. Groups are used to combine properties from multiple providers into one lookup object (for example, an environment variable). This reduces the number of lookups performed by the code of the requiring module.</p>	< <code>requires_section_name</code> >
properties	<p>A structured set of name-value pairs, which contain values that are injected into the environment of requiring modules at run time.</p> <p>A <code>properties</code> property section has a map structure at its first level; any deeper levels can be structured in any way, as long as they can be transformed into a valid JSON object or array.</p> <p>Keys defined at the first level map are used as lookup names. If environment variables are used, one variable is created per map key, and the name of the variable equals the key value.</p> <div style="background-color: #ffffcc; padding: 10px;"> <p><b>➔ Tip</b></p> <p>It is possible to declare metadata for parameters and properties defined in the MTA deployment description; the mapping is based on the parameter or property keys. For example, you can specify if a parameter is <b>required</b> (<code>optional: false</code>) or can be modified <code>overwritable: true</code>.</p> </div>	<pre>properties:   name: backend   url: \${default-url}   forwardAuthToken: true</pre>

Attribute	Description	Example Value
<code>parameters</code>	Reserved variables that affect the behavior of the MTA-aware tools, such as the deployer. Parameters can be “read-only” values, “write-only”, or “read-write” (current/defined value can be overwritten). Modules and resources can read parameters by referencing them with place-holder notation, for example, enclosed by \${ }.	host: \${user}-node-hello-world
<code>tasks</code>	Execute the task(s) defined in <code>command</code> : against the droplet of the module/application which is being deployed. This allows an MTA to execute one or more tasks which are run in a copied droplet of the application. The tasks are executed after the deployment of the application in which they are specified. Use <code>env:</code> to define environment variables which will be available during the task execution.	<pre>tasks   - name: task-1     command: node     deploy.js     env:       env1: &lt;value1&gt;       env2: &lt;value2&gt;   - name: task-2</pre>

### i Note

In the context of the module `type`, the “deployment environment” is any required set of configuration tools, deployment orchestration tools, and target-platform-specific services used to deploy an MTA.

The following table shows how the modules `type` defined in the MTA deployment descriptor is mapped, where appropriate, to a corresponding build pack and which, if any parameters and properties can be used.

Table 8: MTA Default Modules Types

Module Type	Module Parameters (default)	Module Properties	Result
<code>javascript.nodejs</code>	None	None	Node.js run time
<code>native</code>	None	None	Native C++ run time
<code>custom</code>	None	None	Custom run time
<code>java</code>	None	None	Java run time
<code>java.tomcat</code>	None	TARGET_RUNTIME (tomcat)	Tomcat run time
<code>java.tomee</code>	None	TARGET_RUNTIME (tomee)	TomEE run time

Module Type	Module Parameters (default)	Module Properties	Result
com.sap.xs.hdi	<ul style="list-style-type: none"> <li>• <code>no-route</code> (true) Do not assign a route to the application</li> <li>• <code>memory</code> (256MB)</li> <li>• <code>health-check-type</code> (none)</li> <li>• <code>execute-app</code> (true) After start and upon completion, application sets [success   failure]-marker in a log message</li> <li>• <code>success-marker</code> (deploy: done)</li> <li>• <code>failure-marker</code> (deploy: failed)</li> <li>• <code>check-deploy-id</code> (true) Check the deployment (process) id when checking the application execution status</li> <li>• <code>dependency-type</code> (hard) In circular module-dependencies, deploy modules with dependency type "hard" first</li> </ul>	None	HDI content activation
com.sap.xs.sds	<ul style="list-style-type: none"> <li>• <code>no-route</code> (true) Do not assign a route to the application</li> <li>• <code>dependency-type</code> (hard) In circular module-dependencies, deploy modules with dependency type "hard" first</li> </ul>	None	Streaming Analytics

Module Type	Module Parameters (default)	Module Properties	Result
com.sap.xs.dwf	<ul style="list-style-type: none"> <li>• no-route (true)</li> <li>• memory (256M) Memory allocated to the application on deployment/startup</li> <li>• execute-app (true) Execute application after deployment. The application should set success-marker or failure-marker for start/completion in the application logs.</li> <li>• success-marker The success marker for application execution, for example, (STDOUT): The deployment of DWF content to .* container finished successfully .*</li> <li>• failure-marker The failure marker for application execution, for example, (STDERR): The deployment of DWF content to .* container failed .*</li> <li>• stop-app (true) Stop the application after execution.</li> <li>• check-deploy-id (true) Check the deploy (process) ID when checking the application-execution status.</li> <li>• dependency-type (hard) In circular module-dependencies, deploy modules with dependency type "hard" first</li> </ul>	None	Data Warehousing Foundation

Module Type	Module Parameters (default)	Module Properties	Result
com.sap.portal.site-content	<ul style="list-style-type: none"> <li>• no-route (true) Do not assign a route to the application</li> <li>• memory (256MB)</li> <li>• health-check-type (none)</li> <li>• execute-app (true) After start and upon completion, application sets [success   failure]-marker in a log message</li> <li>• success-marker (deploy: done)</li> <li>• failure-marker (deploy: failed)</li> <li>• check-deploy-id (true) Check the deployment (process) id when checking the application execution status</li> <li>• dependency-type (hard) Do not assign a route to circular module-dependencies, deploy modules with dependency type "hard" first</li> </ul>	None	HDI content activation

You can use the `properties` key to define more detailed information in a deeper structure, as illustrated in the following example:

#### Sample Code

```
properties:
  address:
    name: Kodjo
    tel: +2331234567890
```

## resources

A resource is something which is required by the MTA at run time but not provided by the MTA, for example, a managed service instance, an instance of a user-provided service, or an external Web resource.

#### Sample Code

Resource Definition in the MTA Deployment Description

```
resources:
  - name: nodejs-hdi-container
    type: com.sap.xs.hdi-container
    parameters:
      config:
```

```

schema: ${default-container-name}

- name: nodejs-uaa
  type: com.sap.xs.uaa
  parameters:
    config_path: xs-security.json
- name: log
  type: application-logs
  optional: true

```

In the `resources` section of the MTA deployment descriptor, the following attributes are used to define each resource:

Table 9: MTA Deployment Descriptor Resources Keys

Attributes	Description	Example Value
<code>name</code>	The name of the required resource	<Module_Name>
<code>type</code>	<p>Determines the way in which the resource is provisioned by the deploy service. In XS advanced, typed resources are provisioned as service instances.</p> <p><b>Note</b>  The resource type is mapped to a service and a corresponding service plan. For more information, see "parameters" below and <a href="#">service mappings</a>.</p>	com.sap.xs.hana-sbss com.sap.xs.hana-schema com.sap.xs.hana-securestore com.sap.xs.hdi-container com.sap.xs.managed-hdi-container com.sap.xs.managed-hana-schema com.sap.xs.managed-hana-secure-store com.sap.xs.jobscheduler com.sap.xs.uaa com.sap.xs.uaa-devuser com.sap.xs.uaa-space com.sap.xs.fs com.sap.portal.site-content com.sap.portal.site-host org.cloudfoundry.user-provided-service org.cloudfoundry.managed-service org.cloudfoundry.existing-service
<code>description</code>	Non-translatable, free-text string; the string is not meant to be presented on application user interfaces (UI)	Resource provides features that ...

Attributes	Description	Example Value
<code>parameters</code>	Reserved variables that affect the behavior of the MTA-aware tools, such as the deployer. Parameters can be “read-only” values, “write-only”, or “read-write” (current/defined value can be overwritten). Modules and resources can read parameters by referencing them with place-holder notation, that is; enclosed by \${ }.	service-plan: hdi-shared
<code>properties</code>	A structured set of name-value pairs, which contain values that are injected into the environment of requiring modules at run time.	<pre>properties:   name: backend   url: ~{url}   forwardAuthToken: true</pre>
<code>optional</code>	An application can declare required resources to be optional, if it can compensate for their non-existence. If a required resource is declared as <code>optional: true</code> and the deployer cannot allocate the required resource, the deployer issues a warning and continues processing. If a required resource is <b>not</b> optional ( <code>optional: false</code> ), the deployer indicates an error and stops processing.	optional: true (default = false)

### i Note

The resource type is mapped to a service and a corresponding service plan.

For example, the resource type “com.sap.xs.hana-sbss” is mapped to the “hana” service plan “sbss”; the resource type “com.sap.xs.uaa-devuser” is mapped to the “xsuaa” service plan “devuser”. The command `xs marketplace` displays a list of all currently available services and service plans.

The following table shows how the **resource type** defined in the MTA deployment descriptor is mapped to a corresponding service and service plan.

Table 10: MTA Default Resource Types and Mapped Services

Resource Type	Service	Service Plan	Created Service
com.sap.xs.hana-sbss	hana	sbss	Service-broker security
com.sap.xs.hana-schema	hana	schema	Plain schema

Resource Type	Service	Service Plan	Created Service
com.sap.xs.hana-securestore	hana	securestore	SAP HANA secure store
com.sap.xs.hdi-container	hana	hdi-shared	HDI container
com.sap.xs.managed-hdi-container	managed-hana	hdi-shared	Managed HDI container
com.sap.xs.managed-hana-schema	managed-hana	schema	Managed plain schema
com.sap.xs.managed-hana-securestore	managed-hana	securestore	Managed HANA secure store
com.sap.xs.jobscheduler	jobscheduler	default	Job Scheduler
com.sap.xs.uaa	xsuaa	default	Global UAA *
com.sap.xs.fs	fs-storage	free	File-system storage
com.sap.xs.uaa-devuser	xsuaa	devuser	Development-user UAA service
com.sap.xs.uaa-space	xsuaa	space	UAA service for a space
com.sap.xs.sds	sds	default	Streaming Analytics
com.sap.xs.auditlog	auditlog	free	Audit-log service
com.sap.portal.site-content	portal-services	site-content	Portal services
<b>⚠ Restriction</b> Only for use with the SAP Node.js module <code>@sap/site-content-deployer</code>			
com.sap.portal.site-host	portal-services	site-host	Portal services
<b>⚠ Restriction</b> Only for use with the SAP Node.js module <code>@sap/site-entry</code>			

## i Note

UAA is the User Account and Authentication service.

The deployment service also supports a number of “special” resource types, as illustrated in the following table:

Table 11: Additional Special MTA Resource Types: Service Mapping

Resource Type	Resource Parameter ID	Created Service
org.cloudfoundry.user-provided-service	<ul style="list-style-type: none"> <li>• <code>service-name</code>  <b>Optional.</b> Name of the service to create. Default value: the resource name.</li> <li>• <code>config</code>  <b>Required.</b> Map value, containing the service creation configuration, for example, <code>url</code> and user credentials (<code>user</code> and <code>password</code>)</li> </ul>	Create or update a user-provided service configured with the specified resource parameters
org.cloudfoundry.managed-service	<ul style="list-style-type: none"> <li>• <code>service</code>  <b>Required.</b> Name of the service to create.</li> <li>• <code>service-plan</code>  <b>Required.</b> Name of the service plan</li> </ul>	Create a managed service
org.cloudfoundry.existing-service	<ul style="list-style-type: none"> <li>• <code>service-name</code>  <b>Optional.</b> Name of the service to ignore. Default value: the resource name.</li> </ul>	Assume that the named service exists and do not try to manage its life-cycle
mta-provided	<ul style="list-style-type: none"> <li>• <code>mta-id</code>  <b>Required.</b> The ID of the provider MTA. Can also contain version ranges as described in the Semantic Version (Semver) specification.</li> <li>• <code>mta-version</code>  <b>Required.</b> The version of the provider MTA.</li> <li>• <code>mta-provides-dependency</code>  <b>Required.</b> The name of the provides dependency in the provider MTA.</li> </ul>	Define dependencies between different MTAs

Resource Type	Resource Parameter ID	Created Service
configuration	<ul style="list-style-type: none"> <li>• provider-nid <b>Optional.</b> Name space ID of the provider MTA. For cross-MTA dependency resolution this value is always "mta"</li> <li>• provider-id <b>Optional.</b> The ID of the provider MTA. Requires the format &lt;mta-id&gt;:&lt;mta-provides-dependency-name&gt;.</li> <li>• version <b>Optional.</b> Version number of the provider MTA; required format is defined in the Semver specification.</li> <li>• target <b>Optional.</b> Name of the provider MTA's target "space"; a structured parameter that must contain the "org" and "space" of the provider MTA.</li> </ul> <p><b>Tip</b> If you specify a wildcard value (*) for organization or space of the provider MTA, the provider would be searched in all organization or spaces for which the wildcard value is provided. If no match is found for the provider MTA in the specified target organization or space, then org: &lt;current-org&gt; and space: SAP is searched.</p> <ul style="list-style-type: none"> <li>• filter <b>Optional.</b> Map value, containing the content entries used to match configuration entries, for example:</li> </ul> <pre>filter:   type: com.acme.plugin   name: some-cool-plugin</pre>	Define the mechanism for filtering configuration entries based on their configuration content

## Parameters

Module, resource, and dependency parameters have special, platform-specific semantics. In general, many of these parameters are the same and have the same syntax and semantics as those that can be specified in manifest files for XS advanced application-deployment, for example, with the `xs push` command. All parameter values can be referenced as part of other property or parameter value strings. To reference a parameter value, use the placeholder notation  `${<parameter>}` , for example,  `${default-host}` .

➔ Tip

It is also possible to declare metadata for parameters and properties defined in the MTA deployment description; the mapping is based on the parameter or property keys. For example, you can specify if a parameter is **required** (optional; false) or can be modified **overwritable: true**.

The following parameters are supported:

Table 12: MTA Development and Deployment Parameters

Parameter	Scope	Read-Only (System)	Default Value	Example
host	modules		<code> \${default-host}</code>	<code>host: \${space}-node-hello-world</code>
hosts	modules		<code>hosts: - \${host}</code>	<code>hosts: - \${space}-node-hello-world - test-\${space}-node-hello-world</code>
domain	modules		<code> \${default-domain}</code>	<code>domain: \${default-domain}.acme.com</code>
domains	modules		<code>domains: - \${default}</code>	<code>domains: - \${default-domain}.acme.com - test-\${default-domain}.acme.com</code>
port	modules		<code> \${default-port}</code>	<code>port: 52001</code>
ports	modules		<code>ports: - \${port}</code>	<code>ports: - 52001 - 52002</code>
route-path	modules	Yes	The context "route-path" which is part of the application default URI. Context path routing is routing based not only on domain names (host header) but also the path specified in the URL	Host-based routing: <code> \${host}.\${domain}:&lt;router-port&gt;\${route-path}</code> Port-based routing: <code> \${domain}:\${port}\${route-path}</code>
no-start	modules		false. Do not start the application on deployment.	<p>➔ Tip</p> <p>Depends on the command-line option --no-start</p> <code>no-start: true</code>

Parameter	Scope	Read-Only (System)	Default Value	Example
tasks	modules	Yes	Specify tasks which are available for execution in the current droplet of the application. Also provide use of environment variables which are specified with the <code>env</code> scope.	<pre>tasks:   - name: task-1     command: some-script.sh     env:       env1: value1       env2: value2</pre>
command	modules		Empty, or as specified in the deploy service configuration	<code>command: node index.js</code>
buildpack	modules		Empty, or as specified in the deploy service configuration	<code>buildpack: git://github.acme.com/xs2-java/xs2javabuildpack</code>
disk-quota	modules		-1, or as specified in module-type	<code>disk-quota: 1G</code>
memory	modules		256M, or as specified in module-type	<code>memory: 128M</code>
instances	modules		1, or as specified in module-type	<code>instances: 2</code>
service	resources		Empty, or as specified in resource-type	<code>service: hana</code>
service-plan	resources		Empty, or as specified in resource-type	<code>service-plan: hdi-shared</code>
service-name	resources	Yes	The name of the XS advanced service to be created for this resource, based on the resource name with or without a name-space prefix	<code>nodejs-hdi-container</code> <code>com.sap.xs2.samples.xsj</code> <code>shelloworld.nodejs-hdi-container</code>
service-broker-name	modules	Yes	The name of the XS advanced service broker to be created and registered for the specified application module; default value is: <code> \${app-name}</code>	<code>service-broker-name: jobscheduler</code> <code>service-broker-name: \${app-name}</code>
service-broker-user	modules	Yes	The name of the user required for authentication by the XS controller at the service broker when performing service-related requests. The parameter is mandatory if <code>create-service-broker: true</code> .	<code>service-broker-user: \${generated-user}</code>

Parameter	Scope	Read-Only (System)	Default Value	Example
service-broker-password	modules	Yes	The password used for authentication by the XS controller at the service broker when performing service-related requests. The parameter is mandatory if <code>create-service-broker: true</code> .	service-broker-password: \${generated-password}
service-broker-url	modules	Yes	Specifies the value of the service broker universal resource locator (URL) to register; service requests are sent to this URL. The parameter is mandatory if <code>create-service-broker: true</code> .	service-broker-url: \${default-url}
create-service-broker	modules	Yes	Specifies whether [true false] a service broker should be registered for the application module; default value is: false	create-service-broker: true
platform	All	Yes	Name of the target platform. If not specified explicitly, a target platform is created implicitly as " <code>&lt;org&gt; &lt;space&gt;</code> ".	XSA-INITIAL,"initial initial","trial a007007"
org	All	Yes	Name of the target organization	initial, trial
space	All	Yes	Name of the target organizational space	initial, a007007
user	All	Yes	Name of the current user	
app-name	modules	Yes	The name of the XS advanced application to be deployed for this module, based on the module name with or without a name-space prefix	node-hello-world com.sap.xs2.samples.xsj shellworld.node-hello-world
default-domain	All	Yes	The default domain (configured in XS advanced)	accra6024 cfapps.acme.com

Parameter	Scope	Read-Only (System)	Default Value	Example
default-host	modules	Yes	The default host name, composed based on the target platform name and the module name, which ensures uniqueness. Used with host-based routing to compose the default URI, see below.	trial-a007007-node-hello-world
default-port	modules		The default, automatically allocated port value. Used with port-based routing to compose the default URI (explained below)	
default-uri	modules	Yes	The default URI, composed either as \${domain}:\${port} with port-based routing or as \${host}.\${domain} with host-based routing. Note that \${host} will be the same as \${default-host}, unless specified explicitly as a parameter. Similarly, \${domain} will be the same as \${default-domain}, and \${port} will be the same as \${default-port}, unless specified explicitly.	accra6024:52001
default-url	modules	Yes	The default URL, composed as \${protocol}://\${default-uri}	\${protocol}://\${default-uri}
default-container-name	resources	Yes	Default value for the container-name parameter that is used during HDI creation. It is based on the organization, space and service name, which are combined in a way that conforms to the container-name restrictions for length and legal characters.  N/A	INITIAL_INITIAL_SERVICE_NAME

Parameter	Scope	Read-Only (System)	Default Value	Example
default-xsappname	resources	Yes	Default value for the xsappname parameter that is used during UAA creation. It is based on the service name, which is modified in a way that conforms to the xsappname restrictions for length and legal characters.  N/A	xs_deploy-service-database (if the service name is "xs@deploy-service-database")
protocol	modules	Yes	The protocol used by XS advanced, for example: "http" or "https"	https
xs-api-url	All	Yes	The URL for the XS advanced application programming interface (API)	XS: http://localhost:30030
xs-auth-url	All	Yes	N/A	https://localhost:9999/uaa-security
xs-type	All	Yes	N/A	CF, XSA
dependency-type	modules		Deployment order of modules with circular dependencies.  soft	dependency-type: hard  dependency-type: soft
generated-password	All	Yes	A generated user id that is composed of 16 characters that may contain upper and lower case letters, digits and special characters (_-, @, \$, &, #, *).  N/A	IG@zGg#2g-cvMvsW
generated-user	All	Yes	A generated user id that is composed of 16 characters that may contain upper and lower case letters, digits and special characters (_-, @, \$, &, #, *).  N/A	uYi\$d41TzM1-Dm6f
siteId	resources		A globally unique ID (GUID) for your Fiori LaunchPad site	siteId=4c736e0c-a096-45f1-9ae5-a613eb24b2b9

## Metadata for Properties and Parameters

It is possible to declare metadata for parameters and properties defined in the MTA deployment description, for example, using the “parameters-metadata:” or “properties-metadata:” keys, respectively; the mapping is based on the keys defined for a parameter or property. You can specify if a property is required (optional: false) or can be modified (overwritten: true), as illustrated in the following (incomplete) example:

### Code Syntax

```
modules:
- name: frontend
  type: javascript.nodejs
  parameters:
    memory: 128M
    domain: !sensitive ${default-domain}
  parameters-metadata:
    memory:
      optional: true
      overwritable: true
    domain:
      overwritable: false
  properties:
    backend_types:
      order_management: sap-erp
      data_warehouse: sap-bw
  properties-metadata:
    backend_types:
      overwritable: false
      optional: false
      sensitive: false
```

The `overwritable:` and `optional` keywords for defining metadata are intended for use in extension scenarios, for example, where a value for a parameter or property is supplied at deployment time and declared in a deployment-extension descriptor file (`myDeploymentExtension.mtaext`). Parameters or properties defined in the `mtad.yaml` deployment descriptor with the metadata value `overwritable: false` cannot be overwritten by a value supplied from the extension descriptor. In this case, an error would occur in the deployment.

### Tip

Parameters or properties can also be declared as sensitive.

#### **overwritable:**

The effect of the metadata key `overwritable: [true | false]` differs depending on the value type of the property or parameter it modifies. For example, if a value is declared in a deployment-**extension** descriptor for a value that in the original deployment descriptor is declared with the metadata parameter `overwritable: false`, then the value in the extension descriptor is ignored. The following rules apply for the modification of the specified parameter or property:

- For parameters and properties of value type scalar (or an array), the overwritable operation replaces the value declared
- If a parameter (or property) value type is structured (for example, as a map or object), then “overwritable” means “extendable” or “merged”

- A parameter (or property) value type with no declared value can be overwritten by either a scalar value or a structured value
- If there is mismatch in the value type declared for the original parameter (or property) in the MTA deployment descriptor and a corresponding value for the same parameter (or property) in an MTA deployment-**extension** descriptor, an error occurs with notification that the overwrite operation was not successful

#### sensitive:

The `sensitive` keyword enables you to hide sensitive information from public view, for example, if it is written to log or audit files. If a property or parameter is flagged as “sensitive”, any related information is not written as plain text in log files; it is masked, for example, using a string of asterisks (\*\*\*\*\*). You can declare a parameter or property as “sensitive” in either of the following ways:

- `!sensitive`  
Add the string `!sensitive` before the parameter value, for example: `domain: !sensitive $ {default-domain}`
- `sensitive: true`  
Add the `sensitive: true` key to the list of parameter-metadata

## Related Information

[Create the MTA Description Files \[page 113\]](#)

[The MTA Deployment Descriptor \[page 121\]](#)

[Maintaining Application Services in XS Advanced \[page 660\]](#)

## 4.1.4 The MTA Deployment Extension Description

Provide system-specific details that are not known until deployment time.

The MTA deployment descriptor provides the input required by the MTA deployer; it contains the necessary configuration for deployment and for passing values to the applications. However, in some instances, the final values are not known to the developer of the application. This type of system-specific information is **only** known to the administrator of the system on which the application is deployed.

For the cases where system-specific information is required at deployment time, it is possible to define extensions to the deployment configuration; the modifications must be specified in an MTA **extension**. An MTA extension is a file with the suffix `.mtaext`, for example, `myMTADeploymentExtension.mtaext`. The information defined in the deployment extension is used during deployment; the file name is specified as an option in the deployment command, for example:

### Sample Code

```
xs deploy target/<MTA_archive>.mtar -e myMTADeployExtension.mtaext
```

Extension descriptors can be applied in the following scenarios:

- Deployment:

In theory, the developer-created deployment descriptor can contain all the information required to deploy the corresponding MTA, but this is unlikely in reality. Usually, some deployment information will not be stored with the application code in a version-control system; it is added at deployment time. An extension descriptor provides a convenient way to declare such last-minute deployment-related information. In addition, multiple, alternative deployments can be configured by using multiple extension descriptors for the same application.

You use the MTA deployment **extension** to specify information that cannot be included in the MTA deployment descriptor, for example, because the details are not yet known, or differ from system to system.

- Proxy information  
For example, host name and port number for connections to a service
- Memory requirements
- URLs  
For connections to application modules
- User-provided service credentials

The syntax required in an MTA deployment extension is illustrated in the following example:

### i Note

The syntax used in the MTA deployment extension (`myMTAextension.mtaext`) is the same as the syntax required to create the MTA development descriptor (`mta.yaml`).

### Sample Code

#### MTA Deployment Extension Description (`MyMTADeployExtension.mtaext`)

```
schema-version: "2.0.0"
ID: com.sap.xs2.samples.javahelloworld.config1
extends: com.sap.xs2.samples.javahelloworld
modules:
  - name: java-hello-world
    parameters:
      memory: 128M
  - name: java-hello-world-backend
    parameters:
      memory: 512M
      instances: 1
```

You can use the MTA deployment extension descriptor to add the following information (at deployment time) to an MTA deployment descriptor:

- A property element, for example, a new property or parameter.
- A value for a property that does not already have a valued defined

### i Note

It is not possible to use the MTA deployment extension descriptor to add new `modules` or `resources`, or to add new `requires` or `provides` nodes.

## Related Information

[MTA Deployment Descriptor Syntax \[page 145\]](#)

[Create the MTA Description Files \[page 113\]](#)

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

Install a multi-target application (MTA) on SAP HANA.

### Prerequisites

Deploying a multi-target application (MTA) to SAP HANA requires **some or all** of the following components and tools; which components or tools you need depends on what you are deploying: an MTAR archive or a folder structure:

- A file system containing the source files for the multi-target application you want to deploy
- A deployment description for the multi-target application; the mandatory file name is `mtad.yaml`
- An optional deployment description **extension** for the multi-target application deployment description, for example, `myMTADeploymentExtension.mtaext`
- An application's archive manifest `MANIFEST.MF`: required only if you create the MTAR deployment archive manually, for example, with the `jar` command
- An MTA archive, for example, `MTApp-Name.mtar` (optional)
- Knowledge of the `xs deploy` command as well as its parameters and options

### Context

To deploy a multi-target application (MTA) to SAP HANA using the `xs` command-line tools, perform the following steps:

### Procedure

1. Navigate to the parent folder of the file-system containing the application that you want to deploy.
2. Create a deployment descriptor (`mtad.yaml`) for your multi-target application.
3. Create an application manifest file (`MANIFEST.MF`) for the multi-target application.
4. Create an MTA archive.

If you want to deploy a folder structure (including all sub-folders), you can skip this step.

## Restriction

The MTA archive **must** have the extension `.mtar`, for example, `myMTApp.mtar`.

1. Change directory to the parent folder containing **all** the application modules that you want to deploy.
2. Create an MTAR archive that contains the complete MTA folder structure (including all sub-folders). To create the MTA archive, you can use the following tools:
  - o Any archiving tool that produces Zip output
  - o Java's `jar` command, for example:  
`jar cvMf <myMTApp>.mtar -C <myMTApp> .`
5. Deploy the multi-target application to SAP HANA.

You can use either of the following deployment methods:

- o Deploy a **file system** (or part thereof) that includes MTA modules and the MTA deployment descriptor (`mtad.yaml`):
  1. Change directory to the folder containing the application you want to deploy.
  2. Use the `xs deploy` command to deploy the complete MTA folder structure (including all sub-folders) to the default target SAP HANA container.  
`xs deploy <relative/path/to/MTA_folder>`
- o Deploy an **MTA archive**, which you create using Zip (or Java) tools:
  1. Use the `xs deploy` command to deploy the MTAR archive.  
`xs deploy <myMTApp_Name>.mtar`
  2. Check the MTA was successfully deployed.

The information printed to the console should indicate the successful completion of the deployment process, as illustrated in the following example.

## Output Code

```
SSL context configured!
...
Upload finished successfully
Starting process xs-deploy ...
Processing MTA archive...
MTA archive processed successfully.
Building model...
...
Process finished successfully
```

## ➔ Tip

If your deployment requires system-specific information that is only available at deployment time, you can include the information in a so-called MTA deployment **extension**. The file containing the extension details can then be specified as a deployment parameter, for example, `xs deploy --archive <myMTApp_Name>.mtar -e myMTADeploymentExtension.mtaext`

6. Check the deployment results.

You can check that an MTA was successfully deployed by listing the corresponding services and applications and accessing the deployed application at the assigned URL.

- a. Log on to the XS controller.

### Sample Code

```
xs login -a https://<hana-host>:3<instance-nr>30 -u <org-master-user> -p <org-master-pwd> --skip-ssl-validation
```

- b. List all running services.

### Sample Code

```
xs services
```

You should see the HDI service instance used by the application (for example, nodejs-hdi-container), and also the UAA service instance (for example, nodejs-uaa)

### Output Code

```
xs services
name           service  plan      bound apps
-----
-----
auditlog-db-container   hana    hdi-shared auditlog-db, auditlog-
server, ...
auditlog-sbss          hana    sbss       auditlog-server, auditlog-
broker, ...
...
hdi-container          hana    hdi-shared
...
nodejs-hdi-container   hana    hdi-shared node-hello-world-db, node-
hello...
nodejs-uaa              xsuaa   default    node-hello-world-backend,
node-hello...
...
```

- c. List all applications.

### Sample Code

```
xs apps
```

A list of applications is displayed showing all applications that are part of the deployed MTA, for example, node-hello-world-db, node-hello-world-backend, and node-hello-world. The status of the db application should be STOPPED; the status of the other applications should be STARTED. The db application is stopped by the deployment process after the HDI deployment finishes, to prevent it from wasting resources.

### Output Code

```
xs apps
name           requested state  instances  memory  [...]
-----
hdi-broker      STARTED      1/1        256 MB   [...]
uaa-security-db STARTED      1/1        256 MB   [...]
uaa-security    STARTED      1/1        2.00 GB  [...]
jobscheduler-db STARTED      1/1        256 MB   [...]
jobscheduler-service STARTED      1/1        1.00 GB  [...]
```

jobscheduler-broker	STARTED	1/1	1.00 GB	[...]
xsa-monitor	STARTED	1/1	512 MB	[...]
deploy-service	STARTED	1/1	512 MB	[...]
product-installer	STARTED	1/1	512 MB	[...]
<b>node-hello-world-db</b>	<b>STOPPED</b>	0/1	256 MB	[...]
<b>node-hello-world-backend</b>	<b>STARTED</b>	1/1	256 MB	[...]
<b>node-hello-world</b>	<b>STARTED</b>	1/1	256 MB	[...]

### → Tip

The information displayed by the `xs apps` command (but not shown in the example above) also contains the URL and port number to use to access the deployed application, for example, `http://xsa007.acme.com:40003`. You can log on to the listed application at the assigned URL with the name and password of the user with the appropriate credentials.

- d. List all deployed MTAs.

### Sample Code

```
xs mtas
```

You should see the MTA that you just deployed, for example, `com.sap.xsa.samples.nodehelloworld`.

- e. Display the URL assigned to application.

### Sample Code

```
xs app <AppName>
```

With the correct logon credentials, you should be able to access the deployed application.

### Output Code

```
xs app node-hello-world
Showing status and information about "node-hello-world"
  name:           node-hello-world
  requested state: STARTED
  instances:      1
  memory:         128 MB
  disk:           <unlimited>
  buildpack:      sap_nodejs_buildpack
  urls:            https://host.acme.com:40009
Instances of droplet 1 created at May 7, 2017 11:35:52 AM:
index  created          state    host        internal port
-----
0      May 7, 2017 11:37:56 AM  RUNNING  host.acme.com  40323
```

## Related Information

[The MTA Deployment Descriptor \[page 121\]](#)

## 4.2.1 The MTA Archive

An MTA archive is a Jar file containing deployable application modules.

A multi-target application (MTA) archive is used to deploy an application that comprises multiple modules in one deployment operation. An MTA archive contains all the application modules to be deployed; MTA archives have the file extension `.mtar`. MTA archives can be built either with the Java SDK `jar` command or with Zip-tools. An MTA archive contains the following minimum mandatory content:

- One or more MTA modules
- Two mandatory configuration files in the `META-INF/` folder:
  - `MANIFEST.MF`  
The deployment manifest
  - `mtad.yaml`  
The deployment description

The MTA deployment tools assume that the content of an MTA archive has the following mandatory structure:

### Sample Code

SAP HANA MTA Archive

```
/java-backend/
+--- backend.war
/ui5-frontend/
+--- ui.zip
/META-INF/
+--- MANIFEST.MF
+--- mtad.yaml
```

The MTA archive can be deployed using the MTA deployment tools, for example, the `xs deploy` command.

## 4.2.2 The MANIFEST.MF File

The `MANIFEST.MF` file contains a list of the files in the standard Jar archive.

In the context of the SAP HANA MTA archive, the (`manifest.mf`) file contains the usual elements of the standard JAR manifest file plus a `Name` section for the deployment descriptor (`mtad.yaml`). For each MTA module, the `Name` section links the module file (or directory name) to the corresponding MTA module name in the deployment descriptor. In the following example of a simple `manifest.mf` file one of the modules (`utils`) is a directory; the other two (`runner.war` and `builder.war`) are single files.

### Sample Code

```
Manifest-Version: 1.0
Trusted-Library: true
Created-By: Apache Maven 3.2.5
```

```
Build-Jdk: 1.8.0_11

Name: META-INF/mtad.yaml
Content-Type: text/plain

Name: devx/runner.war
MTA-module: di-runner
Content-Type: application/war

Name: devx/utils
MTA-module: di-utils
Content-Type: content-type text/directory

Name: devx-di/builder.war
MTA-module: di-builder
Content-Type: application/war
```

The manifest file may also contain other standard jar-file information, for example, hash values and security signatures.

**i Note**

If the modules specified in the corresponding deployment description file (`mtad.yaml`) contain a `path` attribute, its value must be identical to the `Name` information in the `MANIFEST.MF` file.

## Related Information

[The MTA Deployment Descriptor \[page 121\]](#)

# 5 Defining the Data Model in XS Advanced

The application data model comprises all the database artifacts used to store and provision data for your application's back end and user interface.

As part of the process of defining the database persistence model for your application, you create database design-time artifacts such as tables and views, for example using Core Data Services (CDS). However, you can also create procedures and functions, for example, using SQLScript, which can be used to insert data into (and remove data from) tables or views. For database applications, the structure of the design-time resources should look something like the db/ module illustrated in the following example:

## Sample Code

```
<MyAppName>
|- db/
|   |- package.json
|   \- src/
|       |- .hdiconfig
|       |- .hdinamespace
|       |- data/
|           |- myEntity.hdbcards
|           |- myDataType.hdbcards
|           |- myDoc.hdbcards
|           \- mySynonym.hdbsynonym
|       \- procedures
|           |- myProc.hdbprocedure
|           \- myFunc.hdbfunction
# Database deployment artifacts
# Database details/dependencies
# Database artifacts
# HDI build plug-in configuration
# HDI run-time name-space configuration
# Database artifacts
# CDS table definition
# CDS type definition
# CDS data model definition
# Database synonym definition
# Database procedures, functions, ...
# Database procedure
# Database function
|- web/
|   |- xs-app.json
|   \- resources/
|- js/
|   |- start.js
|   |- package.json
|   \- src/
|- security/
|   \- xs-security.json
\- mtad.yaml
```

The following steps describe the high-level steps you perform when defining, deploying, and consuming the database persistence model for an application deployed to the XS advanced model run time.

1. Define the data model.

Set up the folder structure for the design-time representations of your database objects; this could include CDS documents that define tables, data types, views, and so on. But it could also include other database artifacts, too, for example: your stored procedures, synonyms, sequences, scalar (or table) functions, and so on.

In addition, you can also define the analytic model, for example, the calculation views and analytic privileges that are to be used to analyze the underlying data model and specify who (or what) is allowed access.

2. Set up the SAP HANA HDI deployment infrastructure.

This includes the following components:

- o The HDI configuration

Map the design-time database artifact type (determined by the file extension, for example, .hdbprocedure, or .hbcds in XS advanced) to the corresponding HDI build plug-in in the HDI configuration file (.hdiconfig).

- Run-time name space configuration

Define rules that determine how the run-time name space of the deployed database object is formed. For example, you can specify a base prefix for the run-time name space and, if desired, specify if the name of the folder containing the design-time artifact is reflected in the run-time name space that the deployed object uses.

Alternatively, you can specify the use of freestyle names, for example, names that do not adhere to any name-space rules.

3. Deploy the data model.

Use the design-time representations of your database artifacts to generate the corresponding active objects in the database catalog.

4. Consume the data model.

Reference the deployed database objects from your application, for example, using OData services bound to UI elements.

## Related Information

[Create the Data Persistence Artifacts in XS Advanced \[page 200\]](#)

[Design-Time Database Resources in XS Advanced \[page 202\]](#)

[Maintaining HDI Containers \[page 173\]](#)

## 5.1 Maintaining HDI Containers

Database development artifacts are deployed from and to so-called containers.

The SAP HANA Deployment Infrastructure (HDI) provides a service that enables you to deploy database development artifacts to so-called containers. This service includes a family of consistent design-time artifacts for all key HANA platform database features which describe the target (run-time) state of SAP HANA database artifacts, for example: tables, views, or procedures. These artifacts are modeled, staged (uploaded), built, and deployed into SAP HANA.

**i Note**

The HDI focuses strictly on deployment; HDI does not include any version-control tools, nor does it provide any tools for life-cycle management.

The SAP HANA service broker is used to create and destroy HDI containers; each HDI container comprises a design-time container (DTC) and a run-time container (RTC). The HDI deployment tools deploy database

artifacts to an HDI container. Design-time database objects are typically located in the db folder of the application design-time hierarchy, as illustrated in the following example:

### Output Code

```
app1-hello-world
|- db
|  |- src
|    |- .hdiconfig                                # HDI build-plugin configuration
|    |- .hdinamespace                            # Run-time namespace configuration
|    |- myCDSdoc.hdbcdfs                         # CDS document
|    |- mySynonym.hdbsynonym                      # DB synonym definition
|    |- mySynoConfig.hdbsynonymconfig            # DB synonym configuration file
|    |- myProcedure.hdbprocedure                 # DB procedure definition
|    \- myImportData.hdbtabledata                # Data-import definition
\- package.json                                  # Deployment dependencies
```

The deployment process populates the database run-time with the specified catalog objects. In addition to database artifacts, HDI also enables you to import and export table content such as business configuration data and translatable texts.

### Restriction

HDI enables you to deploy database objects only; it is not possible (or necessary) to deploy application-layer artifacts such as JavaScript programs or OData objects.

As part of the maintenance and management of containers, system administrators typically perform the following operations:

- Configure access to the SAP HANA Deployment Infrastructure (including containers)
- Create and remove HDI containers, each consisting of both a design-time and a run-time container

The configuration of HDI containers involves the creation and configuration of the following design-time artifacts:

- Container deployment configuration (`.hdiconfig`)  
A JSON file containing a list of the bindings between database artifact types (for example, sequence, procedure, table) and the corresponding deployment plugin (and version).
- Run-time container namespace rules (`.hdinamespace`)  
A JSON file containing a list of design-time file suffixes and the naming rules for the corresponding runtime locations.

### Tip

You can apply the rules defined in the `.hdinamespace` file either exclusively to the folder it is located in or to the folder it is located in **and** its subfolders.

## Related Information

[HDI Containers \[page 178\]](#)

[HDI Run-Time Name spaces \[page 185\]](#)

## 5.1.1 Naming Conventions in HDI

Rules and restrictions apply to the names of artifacts in the HDI virtual file system

When specifying the name of a database object that is deployed to a HDI container and written to the HDI virtual file system, bear in mind the rules listed here. For HDI runtime objects and their namespaces, valid characters are those that are also valid for catalog names (described in the *SAP HANA SQL and System Views Reference* ), except for the following restrictions:

- Namespace names cannot contain the forward-slash character (/).
- The backslash (\) can be used but must be escaped in the .hbdbamespace file with a second backslash.
- Namespace and object names cannot contain the colon character (:)
- The double-colon (:) is used as separator between the namespace and the object name.

### Note

HANA 2 SPS01 Rev 11: For synonyms, the separator can also be the forward-slash character (/). The forward slash should only be used for migration scenarios where the double-colon cannot be used.

- The namespace name can be empty, in which case no separator is used. If a namespace is used, then object names must include the declared namespace as a prefix, for example, <namespace>::myObject

### Restriction

Due to catalog rules, the maximum length of the fully qualified object name (including the namespace) is 127 characters.

In the HDI virtual file system, the following restrictions apply to the names of files and folders:

- Names can include only a subset of ASCII, namely, the following characters:
  - Lower- or upper-case letters (aA-zZ)
  - Digits (0-9)
  - In addition to the standard characters listed above, HDI files and folders can also include the following special characters:  
! # % & ' ( ) + , - . ; = @ [ ] ^ \_ ` { } ~
- The maximum length of the object's name (including the path to the source file) is 255 characters in the HDI virtual file system.
- The forward slash (/) is used as the folder delimiter in the HDI path:  
path/to/HDI/object.hbdsuffix
- An HDI path must **not** start with a forward slash (/). Only relative paths are permitted, for example:  
path/to/HDI/object.hbdsuffix
- Trailing spaces ‘‘ or periods ‘.’ are not allowed in the name of an HDI file or folder
  - Trailing space:  
path/to/HDI/object.hbdsuffix<trailing space>
  - Trailing period:  
path/to/HDI/object.hbdsuffix.

- For HDI folders:
  - An empty path denotes the root folder
  - Other paths must end with a forward slash (/)
    - path/
    - path/to/
    - path/to/HDI/object/
- Core Data Services (CDS)  
Additional restrictions apply to the names of CDS artifacts deployed to an HDI container. For more details, see *Related Information*.
- SAP Web IDE for SAP HANA  
If you are using SAP Web IDE for SAP HANA to develop applications for XS advanced, additional restrictions apply to the names of design-time artifacts. For more details, see *Related Information*.

## Related Information

[CDS Naming Conventions in XS Advanced \[page 222\]](#)  
[SAP Web IDE for SAP HANA Reference \[page 959\]](#)

## 5.1.2 Set up an HDI Container

Create the containers and configure the build plug-ins and name-space rules required for deployment operations.

### Context

The service broker creates HDI container on a shared database. A HDI container is a set of schemata and a set of users that, together, allow an isolated deployment of database artifacts. From an application perspective, it is just a technical user and a schema.

Design-time artifact types (indicated by the artifact's file suffix) must be associated with a corresponding build plug-in to ensure the successful deployment of an application. The binding between an artifact type and a build plug-in is established by the contents of the `.hdiconfig` file, which must be deployed into a container.

For backward compatibility with legacy applications, the deployed objects can also be assigned to a run-time name space, which is defined in the `.hdinamespace` file.

### Procedure

1. Check that the HDI service is available for your organizational space.

```
xs marketplace
```

- Create an HDI container using the service broker.

```
xs create-service hana hdi-shared <myHDIcontainer>
```

- Bind the new HDI container to the application that you want to use it.

```
xs bind-service <myApplication> <myHDIcontainer>
```

- Configure the build plug-ins required for the artifacts deployed to the HDI container.

By default, a file suffix (for example, .hdbprocedure) is not bound to a specific version of a build plug-in. Instead, the binding is established by the contents of the .hdiconfig file, which must be deployed into a container along with the database artifacts it lists. An .hdiconfig file is a JSON file with the structure illustrated in the following example:

#### Sample Code

```
{
  "file_suffixes" : {
    "hdbview" : {
      "plugin_name" : "com.sap.hana.di.view",
      "plugin_version": "11.1.0" },
    "hdbprocedure" : {
      "plugin_name" : "com.sap.hana.di.procedure",
      "plugin_version": "11.1.0" },
    "<hdb_file_suffix_#>" : {
      "plugin_name" : "<plugin_NAME>",
      "plugin_version": "<plugin_VERSION>" }
  }
}
```

#### ➔ Tip

Add to the .hdiconfig file, the file-suffixes of the design-time artifacts deployed by your applications. The .hdiconfig file must be placed in the root folder of the database source-files, for example, app1/db/.hdbconfig.

- Configure the run-time name spaces for the objects added to the run-time container during application deployment.

In SAP HANA HDI, name-space rules are defined in one or more file resources named .hdinamespace, which must be located in the design-time folder to which the naming rules apply - or the root folder of a hierarchy to which the naming rules apply. The content of the .hdinamespace file is specified according to the JSON format with the following structure:

#### ➔ Tip

Change the path defined in name and, for subfolder, choose between append (add sub-folder name to object name space) and ignore (do **not** add sub-folder name to the object name space).

#### Sample Code

```
src/.hdinamespace

{
  "name"       : "com.sap.hana.example",
  "subfolder"  : "<[append | ignore]>"
```

```
}
```

## 6. Deploy the application.

The deployment operation pushes the specified application's database objects to the HDI container to which it is bound.

```
xs deploy
```

## Related Information

[The HDI Container Configuration File \[page 180\]](#)

[HDI Container Configuration File Syntax \[page 181\]](#)

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

### 5.1.2.1 HDI Containers

An SAP HANA HDI container consists of a design-time container and a corresponding run-time container.

SAP HANA HDI uses containers to store design-time artifacts and the corresponding deployed run-time (catalog) objects. The SAP HANA Deployment Infrastructure (HDI) strictly separates between design-time and run-time objects by introducing the following, distinct container types:

- **Design-time** container (DTC)  
An isolated environment for design-time files
- **Run-time** container (RTC)  
Stores the deployed objects built according to the specification stored in the corresponding design-time artifacts

#### ➔ Tip

HDI uses so-called Build Plug-ins to create run-time objects from the corresponding design-time files, for example, a procedure object in the database catalog from a design-time definition of a stored procedure.

## Design-Time Containers

In SAP HANA, the design-time container is used to store the design-time representations of the catalog objects that you want to create during the deployment process. The DTC does not provide direct access to its database storage; it contains meta-data that can only be accessed via the HDI application-programming interface (API). The HDI API enables access to design-time resources inside a DTC by means of the following file systems:

- work  
A read-write file system where developers can create, modify, and delete design-time resources. The resources in the **work** file system are not modified by HDI, so undeploying a resource from the **deployed**

file system leaves the corresponding resource in the work file system untouched. This is because deployed resources are handled by the **deployed** file system.

- **deployed**  
A read-only file system containing the design-time resources that have been **deployed**. The deployed file system can only be changed by HDI during the deployment and undeployment process.

### **Restriction**

For maximum isolation and increased security, content stored in a DTC is owned by a dedicated technical user.

Build Plug-ins are used to transform a certain type of design-time file into the corresponding run-time objects. For this reason, each design-time resource type must be mapped to a build plug-in for each container that is created. It is not possible to create a catalog object from a design-time resource without having a plug-in which can read and transform this source definition during deployment.

A design-time container is always attached to a corresponding **target** run-time container.

## Run-Time Containers

In SAP HANA, the database run-time container (RTC) stores the deployed objects built according to the specification stored in the corresponding design-time artifacts. During the deployment and undeployment operations, all interaction with the RTC is performed by build plug-ins, which are mapped to distinct design-time artifact types in the container-configuration file (`.hdiconfig`).

An RTC does not know about the concept of DTCs. As a consequence, the RTC can be identified with a standard database schema without any need for HDI meta-data.

The creation of an RTC results in a new technical database user and a schema with the same name as the one chosen for the name of the RTC itself. Access to RTC content is controlled by database privileges, for example, by granting SELECT or EXECUTE privileges.

## Build Plug-ins

The transformation of design-time resources into corresponding run-time (catalog) objects as well as the extraction of dependency information is performed by so-called build plug-ins. A newly created design-time container cannot access any plug-ins; this access must be defined in the HDI configuration file (`.hdiconfig`). In addition, since there is no default behavior configured, it is not possible to deploy any design-time files (except the two HDI container-configuration and name-space configuration files `.hdiconfig` and `.hdinamespace` respectively) without configuring a build plug-in first. To configure a DTC to use certain build plug-ins, you must set up and deploy a `.hdiconfig` container-configuration file.

### **Tip**

By default, a file suffix (for example, `.hdbprocedure`) is not bound to a specific version of a known build plug-in. Instead, the binding is established by the contents of the `.hdiconfig` file, which must be deployed into a container.

## Related Information

[The HDI Container Configuration File \[page 180\]](#)

[Maintaining HDI Containers \[page 173\]](#)

### 5.1.2.2 The HDI Container Configuration File

Bind design-time file types to a corresponding build plug-in.

A design-time container defines an isolated environment for design-time files. To create run-time objects from the corresponding design-time files (for example, a catalog object from a source definition of a stored procedure) HDI uses so-called Build Plug-ins. It is not possible to create a catalog object from a design-time resource without having a plug-in which can read and transform this source definition. For this reason, each design-time resource type must be mapped to a build plug-in for each container that is created. This mapping is configured in an HDI container-configuration file: a file with no name but the mandatory file suffix `.hdiconfig`.

#### i Note

The `.hdiconfig` file must be located in the root folder of the database module. However, you can include additional `.hdiconfig` files further down the source-file hierarchy, for example, if you want to change (expand/restrict) the definitions at a certain point in the hierarchy.

By default, a file suffix (for example, `.hdbprocedure`) is not bound to a specific version of a known build plug-in. Instead, the binding is established by the contents of the `.hdiconfig` file, which must be deployed into the container along with the artifacts it describes. An `.hdiconfig` file is a JSON file with the following structure:

#### Sample Code

```
{  
  "file_suffixes": {  
    "hdbsynonym": {  
      "plugin_name": "com.sap.hana.di.synonym",  
      "plugin_version": "11.1.0"},  
    "hdbview": {  
      "plugin_name": "com.sap.hana.di.view",  
      "plugin_version": "11.1.0"},  
    "hdbcalculationview": {  
      "plugin_name": "com.sap.hana.di.calculationview",  
      "plugin_version": "11.1.0"},  
    "hdbprocedure": {  
      "plugin_name": "com.sap.hana.di.procedure",  
      "plugin_version": "11.1.0"},  
    "<hdb_file_suffix_#>": {  
      "plugin_name": "<plugin_NAME><plugin_VERSION>",  
    }  
  }  
}
```

The plug-in **name** corresponds to the plugin name in reverse-URL notation, for example, `com.sap.hana.di.procedure` or `com.sap.hana.di.view`.

The plug-in **version** is described as follows: **major** version, **minor** version, and an optional **patch** version, for example, 11.1.0.

## Related Information

[Maintaining HDI Containers \[page 173\]](#)

[HDI Design-Time Resources and Build Plug-ins \[page 204\]](#)

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

### 5.1.2.3 HDI Container Configuration File Syntax

JSON syntax is used to format the content of the HDI container-configuration file (`.hdiconfig`).

#### Sample Code

```
{  
  "file_suffixes" : {  
    "<file_suffix_1>" : {  
      "plugin_name" : "<plugin_NAME>",  
      "plugin_version": "<plugin_VERSION>"  
    },  
    "<file_suffix_2>" : {  
      "plugin_name" : "<plugin_NAME>",  
      "plugin_version": "<plugin_VERSION>"  
    },  
    "<file_suffix_#>" : {  
      "plugin_name" : "<plugin_NAME>",  
      "plugin_version": "<plugin_VERSION>"  
    },  
    {  
      [...]  
    }  
  }  
}
```

#### file\_suffixes

Define one or more file suffixes which you want to bind to a build plug-in for deployment to an HDI container. The file suffix specifies the **type** of database artifact, for example, `.hdbsequence` or `.hdbview`)

#### Sample Code

```
{  
  "file_suffixes" : {  
    "<file_suffix_1>" : {  
      "plugin_name" : "<plugin_NAME>",  
      "plugin_version": "<plugin_VERSION>"  
    }  
  }  
}
```

```
    }  
}
```

You can use the asterisk character (\*) to define a **global** file suffix which binds all remaining, unbound file suffixes to a specific build plug-in, for example, to bind all unbound file suffixes to the Copy Only plug-in.

### **Restriction**

All "hdi \*" file suffixes (for example, .hdiconfig or hdinamespace) are reserved for use with HDI; these file suffixes cannot be (re)configured.

## **plugin\_name**

Use the `plugin_name` key to specify the name of the HDI build plug-in that you want to associate with a specific HDI artifact type.

### **Sample Code**

```
"plugin_name" : "<plugin_NAME>,"
```

For example, you can associate the artifact suffix for the HDB SQL view (.hdbview) with the build plug-in com.sap.hana.di.view.

### **Sample Code**

```
"plugin_name" : "com.sap.hana.di.view",
```

## **plugin\_version**

Use the `plugin_version` key to specify the version number of the HDI build plug-in that you want to associate with a specific HDI plug-in name.

### **Sample Code**

```
"plugin_version": "<Major_Version>.<Minor_Version>.<Patch_Version>"
```

The value of the `plugin_version` key is interpreted as a minimal required version of the build plug-in. Version numbers follow the format `<major>.<minor>.<patch>`. An installed build plug-in with the same major version is considered to be compatible if its minor version is greater than the requested minor version, or its minor version is equal to the requested version and the patch number is greater than (or equal to) the requested version.

### Sample Code

```
"plugin_version": "11.1.0"
```

#### Note

The plugin version corresponds to the version of the service-pack that includes it, for example, “11.1.0” (SPS 11).

## Related Information

[Set up an HDI Container \[page 176\]](#)

[HDI Design-Time Resources and Build Plug-ins \[page 204\]](#)

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 5.1.3 Define HDI Run-Time Name-Space Rules

Define rules for run-time name spaces in HDI containers.

## Prerequisites

For the definition and deployment of name space rules for run-time container objects, the following prerequisites apply:

- The design-time folder hierarchy contains application artifacts
- The design-time containers exist
- The corresponding run-time containers exist

## Context

The application objects deployed to an HDI container must also be assigned to a run-time name space, which is defined in the `.hdinamespace` file.

## Procedure

1. Check that the HDI service is available for your organizational space.

```
xs marketplace
```

2. Create an HDI container using the service broker.

```
xs create-service hana hdi-shared <myHDIcontainer>
```

3. Bind the new HDI container to the application that you want to use it.

```
xs bind-service <myApplication> <myHDIcontainer>
```

4. Configure the run-time name spaces for the objects added to the run-time container during application deployment.

In SAP HANA, name-space rules are defined in one or more file resources named `.hdinamespace`, which must be located in the design-time folder to which the naming rules apply - or the root folder of a hierarchy to which the naming rules apply. The content of the `.hdinamespace` file is specified according to the JSON format with the following structure:

### Tip

Change the path defined in `name` and, for `subfolder`, choose between `append` (add sub-folder name to object name space) and `ignore` (do **not** add sub-folder name to the object name space).

### Sample Code

```
src/.hdinamespace
```

```
{  
    "name"      : "com.sap.hana.example",  
    "subfolder" : "<[append | ignore]>"  
}
```

5. Deploy the name-space configuration to the container where the rules apply.

The `.hdinamespace` file must be located in the root folder of the database source-files, for example, `app1/hdb/.hdinamespace`.

```
xs deploy
```

## Related Information

[The HDI Name-Space Configuration File \[page 186\]](#)

[The HDI Name-space Configuration Syntax \[page 186\]](#)

### 5.1.3.1 HDI Run-Time Name spaces

HDI separates the naming of run-time objects from the organization of design-time files.

There is no one-to-one mapping between the file-system structure and the names of the resulting run-time objects. Instead, the names of run-time objects are determined by naming rules defined in HDI name-space files. If desired, the file-system structure can still be used to derive the run-time name space of an object. However, decoupling design-time file structure and run-time name-space names allows the file-system structure to reflect the organization of work, and it is easy to relocate design-time resources without changing run-time name spaces. Note that if design-time folder names do not contribute to run-time names, you avoid problems with limitations on name length.

In SAP HANA, name-space rules for run-time objects are defined in one or more files named `.hdinamespace`, which must be located in the folder to which the naming rules apply - or the root folder of a folder structure to which the naming rules apply. The name-space rules allow you to specify the following elements:

- A name-space prefix, for example, `com.acme.hana.example`
- The name of design-time folder(s) containing the deployed objects

#### Code Syntax

Run-Time Name Space with Appended Folder Name

```
com.acme.hana.example.<Design-Time_Folder_Name>:<Run-Time_Object_Name>
```

The following table illustrates how the rules specified in the name-space configuration file influence the way run-time objects are named. If the name-space rules specify that the name of the design-time sub-folder should be **appended** to the run-time name-space prefix, for example, `com.acme.hana.example`:

Table 13: Folder Names Appended to the Run-Time Name Space

Design-time Resource Path	Name of Run-Time Object (append)
/src/	com.acme.hana.example:<Object_Name>
/src/db	com.acme.hana.example.db:<Object_Name>
/src/data	com.acme.hana.example.data:<Object_Name>

## Related Information

[The HDI Name-Space Configuration File \[page 186\]](#)

[The HDI Name-space Configuration Syntax \[page 186\]](#)

## 5.1.3.2 The HDI Name-Space Configuration File

A JSON resource that defines naming rules for run-time objects.

In SAP HANA, name-space rules are defined in one or more file resource named `.hdinamespace`, which must be located in the folder to which the naming rules apply - or the root folder of a hierarchy to which the naming rules apply. The content of the `.hdinamespace` file is specified according to the JSON format with the following structure:

### Sample Code

```
src/.hdinamespace

{
  "name"      : "com.sap.hana.example",
  "subfolder" : "<[append | ignore]>"
}
```

Every folder can contain its own name-space file. For a name-space file to take effect, it has to be deployed in the same way as any other design-time file. After deployment, a name-space file provides run-time name-space information for all files in the same folder.

The `.hdinamespace` file provides name-space information for all sub-folders where no additional name-space file is present. For those sub-folders without a name-space configuration file, the name space depends on the value of the `subfolder` key in the name-space-configuration file in the parent folder.

- `append`  
Add the name of the sub-folder to the run-time name space for the deployed objects
- `ignore`  
Do **not** add the name of the sub-folder to the run-time name space for the deployed objects

### Related Information

[HDI Run-Time Name spaces \[page 185\]](#)

[The HDI Name-space Configuration Syntax \[page 186\]](#)

[Define HDI Run-Time Name-Space Rules \[page 183\]](#)

## 5.1.3.3 The HDI Name-space Configuration Syntax

The contents of the `.hdinamespace` file are formatted with the JSON syntax

### Sample Code

```
src/.hdinamespace

{
  "name"      : "com.sap.hana.example",
  "subfolder" : "<[append | ignore]>"
```

```
}
```

## name

Use the `name` property in the `.hdinamespace` configuration file to specify the name of run-time space to map to the name of (and objects in) the file-system folder where the `.hdinamespace` file is located.

### Sample Code

```
"name" : "com.sap.hana.example",
```

## subfolder

Use the `subfolder` key in the `.hdinamespace` configuration file to specify if the name of the file-system folder where the `.hdinamespace` file is located should be **appended** to the name of the corresponding run-time space or **ignored**.

### Sample Code

```
{
  "name" : "com.sap.hana.example",
  "subfolder" : "<[append | ignore]>"
}
```

For those sub-folders without a name-space configuration file, the name space depends on the value of the `subfolder` key in the name-space configuration file in the parent folder, for example: `append` or `ignore`.

- `"subfolder" : "ignore"`

The sub-folder inherits the name-space information from the parent folder. In this case, all sub-folders use the same run-time name space.

Table 14: Design-Time Folder Names Ignored in Run-Time Name Space

Design-time Resource Path	Name of Run-time Object (ignore)
/src/	com.sap.hana.example::<Object_Name>
/src/db	com.sap.hana.example::<Object_Name>
/src/db/proc	com.sap.hana.example::<Object_Name>
/src/data	com.sap.hana.example::<Object_Name>
/src/data/import	com.sap.hana.example::<Object_Name>

- "subfolder" : "append"

The run-time name-space for a sub-folder is derived by appending the sub-folder name to the name-space name of the parent folder.

Table 15: Design-Time Folder Names Appended to Run-time Name Space

Design-time Resource Path	Name of Run-Time Object (append)
/src/	com.sap.hana.example::<Object_Name>
/src/db	com.sap.hana.example.db::<Object_Name>
/src/db/proc	com.sap.hana.example.db.proc::<Object_Name>
/src/data	com.sap.hana.example.data::<Object_Name>
/src/data/import	com.sap.hana.example.data.import::<Object_Name>

## Related Information

[HDI Run-Time Name spaces \[page 185\]](#)

[The HDI Name-Space Configuration File \[page 186\]](#)

[Define HDI Run-Time Name-Space Rules \[page 183\]](#)

## 5.2 Configuring the HDI Deployer

Learn how to set up and use the Node.js-based SAP HANA Deployment Infrastructure (HDI) Deployer for XS advanced.

The HDI Deployer is a Node.js application (`@sap/hdi-deploy`) that is included in an XS advanced Multi-Target Application (MTA) and is used to deploy HDI design-time artifacts to the respective HDI containers. When an MTA is deployed, the HDI Deployer application is pushed first so that it can “prepare” the SAP HANA persistence; by the time defined services are started in XS Advanced, the HDI container is ready for use.

In the XS advanced HDI setup, each module of the MTA (for example `db/`, `js`, or `java`) is assigned its own technical database user for accessing the run-time schema of the HDI container.

The HDI Deployer is packaged into the `db/` module of the MTA. Consequently, in order to use a new HDI Deployer, you need to reference a new version of the HDI Deployer in the `db/` module's `package.json` file. The HDI Deployer supports HANA 1.0 SPS11, HANA 1.0 SPS12, and the current HANA 2.0 development codeline.

### i Note

The HDI Deployer assumes ownership of the `src/`, `cfg/`, and `lib/` folders in the bound target HDI container. It is not possible to bind more than one (1) instance of the HDI Deployer to the same HDI

container as the target container. For example, you cannot bind the `db/` modules of two different MTAs to the same HDI container as the target container; this results in undefined behavior.

## Related Information

[Integrate the HDI Deployer into an XS Advanced MTA Database Module \[page 189\]](#)

[Download and Consume SAP Node.js Packages \[page 516\]](#)

[Maintaining HDI Containers \[page 173\]](#)

### 5.2.1 Integrate the HDI Deployer into an XS Advanced MTA Database Module

Install the HDI Deployer for use by an XS advanced Multi-Target Application.

#### Prerequisites

The following prerequisite apply for the installation and use of the SAP HANA HDI Deployer for XS advanced:

- Node.js package manager (`npm`)

Ensure that `npm` is configured appropriately, for example, with a working URL to the package registry

#### Context

The HDI Deployer is an application that is included in a Multi-Target Application (MTA) and is used to deploy HDI design-time artifacts to the respective HDI containers. When an MTA is deployed, the HDI Deployer application is pushed first so that it can “prepare” the SAP HANA persistence; by the time the services defined in the MTA’s deployment descriptor are started in XS advanced, the HDI container is ready for use.

#### Procedure

1. Ensure that `npm` is configured appropriately.

The following example indicates some of the configuration parameters that might be required (not all indicated parameters are mandatory):

```
npm config set registry "http://npm.acme.com:8081/registry/content/groups/
build.milestones.npm/"
npm config rm proxy
```

```
npm config rm https-proxy  
npm config set strict-ssl false
```

2. Install the HDI Deployer for use by the XS advanced application's database module (db/).

You can install the HDI Deployer by including the Node.js application "@sap/hdi-deploy" in the package.json file that defines the dependencies inside your XS advanced application's db/ module, as illustrated in the following example:

#### Code Syntax

db/package.json

```
{  
  "name": "deploy",  
  "dependencies": {  
    "@sap/hdi-deploy": "2.1.0"  
  },  
  "scripts": {  
    "start": "node node_modules/@sap/hdi-deploy/"  
  }  
}
```

## Related Information

[Configuring the HDI Deployer \[page 188\]](#)

[Download and Consume SAP Node.js Packages \[page 516\]](#)

### 5.2.1.1 The XS Advanced Database Module's File Structure

The HDI Deployer expects the following file system structure for your the HDI content in your db/ module:

- src/ A folder containing the HDI source artifacts required by the XS advanced application
- cfg/ An optional folder containing HDI configuration artifacts
- package.json The configuration file used by the Node.js package manager (npm) to bootstrap and start the XS advanced application

#### Note

All other files in the root directory of the database module (db/) are ignored by the HDI Deployer (@sap/hdi-deploy).

## Related Information

[Integrate the HDI Deployer into an XS Advanced MTA Database Module \[page 189\]](#)

[Configuring the HDI Deployer \[page 188\]](#)

[Maintaining HDI Containers \[page 173\]](#)

### 5.2.1.2 HDI Delta Deployment and Undeploy Whitelist

The HDI Deployer implements a delta-based deployment strategy including an optional white list.

On start up, the HDI Deployer recursively scans the local `src/` and `cfg/` folders, processes configuration templates, checks the HDI container on the server-side, and calculates the set of added, modified, and deleted files based on the difference between the local file system state and the state of the deployed file system of the server-side HDI container.

In normal operation, the HDI Deployer will schedule only the set of added and modified files for deployment.

The set of deleted files is not scheduled for automatic undeployment. To undeploy deleted files, an application must include an undeploy “white list”, for example, by creating an `undeploy.json` file in the root directory of the `db/` module (alongside the `src/` and `cfg/` folders). The undeploy white list `undeploy.json` file is a JSON document with a top-level array of file names, as illustrated in the following example:

#### Code Syntax

Example `undeploy.json` File

```
[  
    "src/Table.hdbcds",  
    "src/Procedure.hdbprocedure"  
]
```

The `undeploy.json` file must list all artifacts to be undeployed. In addition, the file path specified for the artifacts to remove must be **relative** to the root directory of the `db/` module and must use the HDI file path forward-slash delimiter (/).

For interactive scenarios, it is possible to pass the “auto-undeploy” option to the HDI Deployer, as illustrated in the following example:

```
node deploy --auto-undeploy
```

In this case, the HDI Deployer ignores the undeploy white list defined in the `undeploy.json` file and schedules all deleted files in the `src/` and `cfg/` folders for undeployment.

## Related Information

[Integrate the HDI Deployer into an XS Advanced MTA Database Module \[page 189\]](#)

[The XS Advanced Database Module's File Structure \[page 190\]](#)

### 5.2.1.3 The Default Access Role for HDI Containers

An overview of the access role required for HDI containers.

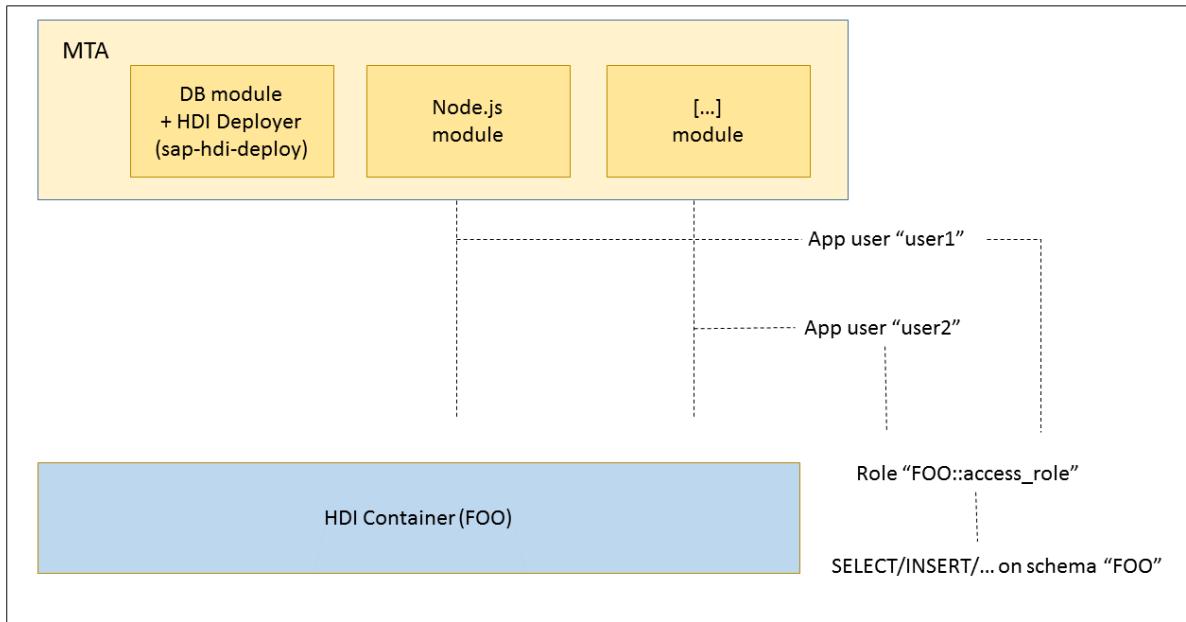
When an HDI container service instance is created by the SAP HANA Service Broker, for example, service instance "foo" with schema name "FOO", the broker creates an HDI container named "FOO" (consisting of the run-time schema "FOO", the HDI metadata and API schema "FOO#DI", and the object owner "FOO#OO") and, in addition, a global access role "FOO::access\_role" for the run-time schema. This access role is assigned a set of default permissions for the run-time schema: SELECT, INSERT, UPDATE, DELETE, EXECUTE, CREATE TEMPORARY TABLE, and SELECT CDS METADATA on the run-time schema "FOO".

Every time the service instance is bound to an application, the service broker creates two new users that are specific to this binding. The first user is the application user who is named user in the instance's credentials. This user is used by the application to access the HDI container's run-time schema "FOO". This user is assigned the service instance's global access role "FOO::access\_role". The second user is the HDI API user - named "hdi\_user" in the credentials. This user is equipped with privileges for the container's APIs in the "FOO#DI" schema.

The following diagram illustrates the binding-specific application users and the role of the global access role:

**i Note**

For the sake of simplicity, the HDI API users and the bindings for the HDI Deployer are not shown in this diagram.



To assign roles from the HDI content to the application-binding-specific users (the users specified in "user":), the HDI Deployer implements an automatic assignment of the "default\_access\_role" role if it is present in the deployed content.

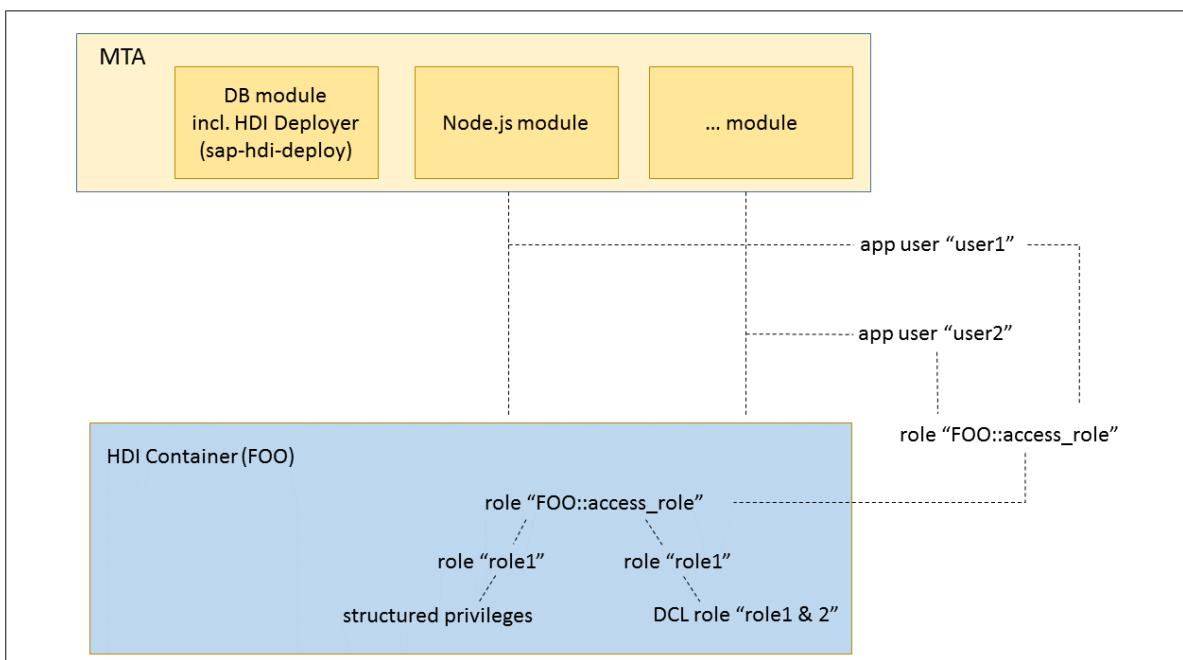
If a role definition file exists at the path `src/defaults/default_access_role.hdbrole`, and this file defines a role named "default\_access\_role", and this file is included in the deployment (for example, it is

not excluded by means of an include filter), then the HDI Deployer grants the deployed “default\_access\_role” role to the service instance’s global access role (for example, “FOO::access\_role”). In addition, the HDI Deployer revokes all default permissions SELECT, INSERT, UPDATE, DELETE, EXECUTE, CREATE TEMPORARY TABLE, and SELECT CDS METADATA on the run-time schema “FOO”) from the global access role.

### i Note

If you use a .hdinamespace file in src/ which defines a real name-space prefix for sub folders, then you need to put a .hdinamespace file with the empty name space “name”: ” at src/defaults/ to ensure that the role can be named default\_access\_role.

The following diagram illustrates the binding-specific application users, the role of the global access role, and the container-specific default access role:



### i Note

The “default\_access\_role” is assumed to be an “umbrella” role which aggregates other roles.

## Related Information

[Configuring the HDI Deployer \[page 188\]](#)

[Integrate the HDI Deployer into an XS Advanced MTA Database Module \[page 189\]](#)

[The XS Advanced Database Module’s File Structure \[page 190\]](#)

[HDI Delta Deployment and Undeploy Whitelist \[page 191\]](#)

## 5.2.1.4 Reusable Database Modules in XS Advanced Applications

Setup reusable components inside an application's own data persistence model.

To enable an application to use parts of the database persistence of a reusable component inside its own persistence model, the HDI Deployer allows you to automate the linking (include) of design-time files of reusable components in a consuming application. This automated mechanism is based on the Node.js package management mechanism for defining, publishing, and consuming reusable database modules, which also supports versioning based on the semantic versioning concepts.

A reusable database module is considered to have the same `src/` and `cfg/` folder structure as a normal database module. The `src/.hdiconfig` file is mandatory and used by the module mechanism as an indicator that the `src/` and `cfg/` folders belong to a consumable, reusable database module. In addition, the reusable database module needs to have a `package.json` file which defines the module's name, the module's version, and so on.

The following example illustrates the structure of a complete reusable database module:

```
/  
+-- src/  
|   +-- .hdiconfig  
|   +-- <source files ...>  
+-- cfg/  
|   +-- <optional configuration files ...>  
+-- package.json
```

The `package.json` file contains the module's name, description, version, repository URL, and the set of files which belong to the module, as illustrated in the following example:

### Code Syntax

`package.json`

```
{  
  "name": "module1",  
  "description": "A set of reusable database objects",  
  "version": "1.3.1",  
  "repository": {  
    "url": "git@github.com:modules/module1.git"  
  },  
  "files": [  
    "src",  
    "cfg",  
    "package.json"  
  ]  
}
```

Consumption of a reusable database module is done by adding a dependency in the consuming module's `package.json` file, for example, alongside the dependency to the HDI Deployer "`@sap/hdi-deploy`" as illustrated below, where the consuming module requires "`module1`" in version "`1.3.1`" and "`module2`" in version "`1.7.0`":

### Code Syntax

```
{
```

```
"name": "deploy",
"dependencies": {
  "@sap/hdi-deploy": "2.1.0",
  "module1": "1.3.1",
  "module2": "1.7.0"
},
"scripts": {
  "start": "node node_modules/@sap/hdi-deploy/"
}
}
```

When running `npm install` to download and install the dependencies which are listed in the dependencies section of the `package.json` file, `npm` will also download the reusable database modules and places them into the `node_modules/` folder of the consuming module. For each module a separate subfolder is created with the name of the module.

When the HDI Deployer is triggered to do the actual deployment of the (consuming) database module, it scans the `node_modules/` folder and virtually integrates the `src/` and `cfg/` folders of reusable database modules it finds into the (consuming) database module's `lib/` folder. Reusable database modules are identified by the mandatory `src/.hdiconfig` file. On successful deployment, the HDI container will contain the consumed modules below the root-level `lib/` folder, as shown in the following example:

```
/  
+-- src/  
+-- cfg/  
+-- lib/  
|   +-- module1/  
|   |   +-- src/  
|   |   +-- cfg/  
|   +-- module2/  
|       +-- src/  
|       +-- cfg/
```

#### i Note

It is not permitted to recursively include reusable database modules.

## Related Information

[Configuring the HDI Deployer \[page 188\]](#)

[Integrate the HDI Deployer into an XS Advanced MTA Database Module \[page 189\]](#)

### 5.2.1.5 Templates for HDI Configuration Files

The HDI Deployer implements a template mechanism for HDI configuration files.

The HDI Deployer enables you to set up a template mechanism for HDI configuration files, for example, configuration files for synonyms or projection views based on services which are bound to the `db/` module application. With this template mechanism, it is possible to configure synonyms, projection views, and similar artifacts. to point to the right database schema without knowing the schema name at development time.

On startup, the HDI Deployer recursively scans the local `cfg/` folder and picks up all files with a `*config` suffix, for example, all `.hdbsynonymconfig`, `.hdbvirtualtableconfig` files. For all collected files which contain `.configure` markers in their content, the HDI Deployer applies the configuration template mechanism and creates transient configuration files which are then deployed to the HDI container as described in the following examples.

For a synonym configuration file `cfg/LOCAL_TARGET.hdbsynonymconfig`:

#### Code Syntax

```
{  
  "LOCAL_TARGET" : {  
    "target" : {  
      "schema.configure" : "logical-external-service/schema",  
      "database.configure" : "logical-external-service/database",  
      "object" : "TARGET"  
    }  
  }  
}
```

The following section:

#### Code Syntax

```
"schema.configure" : "logical-external-service/schema",  
"database.configure" : "logical-external-service/database",  
"object" : "TARGET"
```

is transformed by the template mechanism into:

#### Code Syntax

```
"schema" : "THE_SCHEMA",  
"database" : "THE_DATABASE",  
"object" : "TARGET"
```

where “THE\_SCHEMA” and “THE\_DATABASE” are the values for the schema and database fields of the bound service “logical-external-service”, which are denoted by the path expressions “logical-external-service/schema” and “logical-external-service/database”.

If a field in the service is missing, it will not be configured and will be removed instead, for example, “database” in the code examples above might be optional. The names of the services are subject to the service replacements mechanism, which can be used to map a real service, for example, “real-external-service”, to a logical service name which is used in the configuration files, for example, “logical-external-service”. It is not always applicable to use “schema.configure”, “database.configure”, in the configuration template files. For this reason, the HDI Deployer provides a generic way of copying a set of properties from the bound service, for example, schema, database, remote source, etc. if they are present, although the template file doesn’t mention them. For the configuration file `cfg/LOCAL_TARGET.hdbsynonymconfig`, this could look as follows:

## Code Syntax

```
cfg/LOCAL_TARGET.hdbsynonymconfig  
  
{ "LOCAL_TARGET" : { "target" : { "*.configure" : "logical-external-service",  
"object" : "TARGET" } } }
```

When the HDI Deployer encounters a \*.configure entry, it copies all well-known fields which are present in the bound service into the configuration file. The well-known fields are currently “remote”, “database”, and “schema”.

The HDI Deployer also supports old-style .hdbsynonymtemplate template files. If a .hdbsynonymtemplate file is found in the cfg/ or src/ folder, it is processed as a configuration template file and a transient file with the suffix .hdbsynonymconfig is created. A field grantor is replaced with the schema value from the referenced service; in a configuration template file, a “grantor” field is equivalent to a “schema.configure” : “the-service/schema” entry.

## Related Information

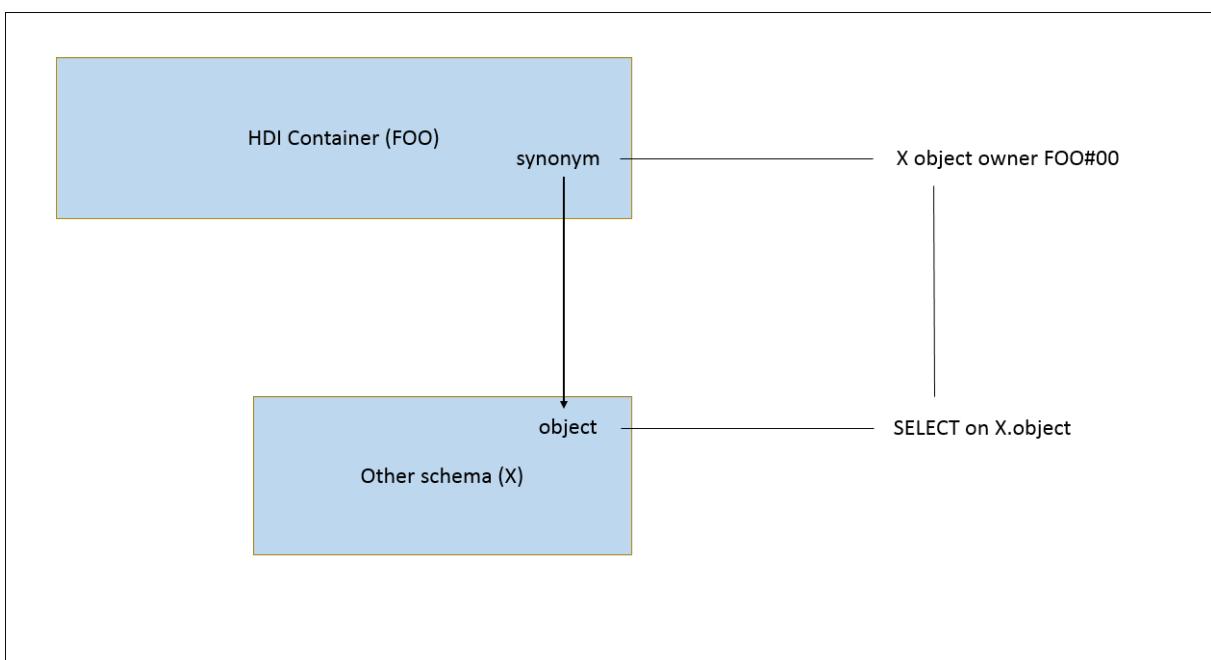
[Configuring the HDI Deployer \[page 188\]](#)

[Integrate the HDI Deployer into an XS Advanced MTA Database Module \[page 189\]](#)

### 5.2.1.6 Permissions for Container Objects

The owner of a container object needs additional privileges to the ones assigned by default.

By default, an HDI container is assigned very few database privileges. For example, the object owner (“#00” user) is only assigned the CREATE ANY privilege on the container’s run-time schema (schema “FOO” for an HDI container “FOO”). To access database objects inside other database schemata or other HDI containers, and to be able to deploy synonyms into the HDI container which point to objects outside the container, the object owner needs additional privileges. For example, for an object “object” in schema “X”, the SELECT privilege on “X.object” :



To assign privileges automatically to the object owner and (or) the application binding users, the HDI Deployer provides .hdbgrants files, which use a syntax that is similar to the .hdbrole artifact. The following example shows the structure of an .hdbgrants file:

### Code Syntax

```

granting-service.hdbgrants

{
  "granting-service": {
    "object_owner": {
      <privileges>
    },
    "application_user": {
      <privileges>
    }
  }
}

```

The top-level keys define the grantors; the names of the bound services which **grant** the privileges, “granting-service” in the code example above. The next level defines the users to whom the privileges are granted, these are the grantees: “object\_owner” is used to specify the HDI container’s object owner, and “application\_user” defines the application users who are bound to the application modules, for example, the Node.js module in an MTA. The third level defines the set of privileges to grant using a structure that is similar to the format used in a .hdbrole role-definition file.

#### Note

For backwards compatibility, the suffix .hdbsynonymgrantor is also supported.

On startup, the HDI Deployer looks for files with the .hdbgrants suffix and processes them in the following way: For each grantor specified in the .hdbgrants file, the HDI Deployer looks up a bound service with the

specified name (subject to service replacements), connects to the database with the service's credentials, and grants the specified privileges to the grantees.

### Code Syntax

Example `cfg/SYS-access.hdbgrants` File with Some Privileges

```
{  
  "SYS-access": {  
    "object_owner": {  
      "object_privileges": [  
        {  
          "schema": "SYS",  
          "name": "VIEWS",  
          "privileges": ["SELECT"]  
        },  
        {  
          "schema": "SYS",  
          "name": "TABLES",  
          "privileges": ["SELECT"]  
        }  
      ]  
    },  
    "application_user": {  
      "object_privileges": [  
        {  
          "schema": "SYS",  
          "name": "VIEWS",  
          "privileges": ["SELECT"]  
        },  
        {  
          "schema": "SYS",  
          "name": "TABLES",  
          "privileges": ["SELECT"]  
        }  
      ]  
    }  
  }  
}
```

### Related Information

[Configuring the HDI Deployer \[page 188\]](#)

[Integrate the HDI Deployer into an XS Advanced MTA Database Module \[page 189\]](#)

## 5.3 Creating the Data Persistence Artifacts in XS Advanced

The data persistence model comprises the underlying database objects you want to use to store and provide data for your application, for example: tables, views, data types.

As part of the process of defining the database persistence model for your application, you create database design-time artifacts such as tables and views, for example using Core Data Services (CDS). You can define

your data persistence model in one or more CDS documents. The syntax for the specific data artifacts is described in the following sections:

- CDS documents
- CDS entities (tables)
- CDS user-defined types
- CDS associations (between entities)
- CDS views
- CDS extensions

## Related Information

[Create the Data Persistence Artifacts in XS Advanced \[page 200\]](#)

[Create a CDS Document \(XS Advanced\) \[page 208\]](#)

[Create a CDS Entity in XS Advanced \[page 243\]](#)

[Create a CDS User-Defined Structure in XS Advanced \[page 264\]](#)

[Create a CDS Association in XS Advanced \[page 278\]](#)

[Create a CDS View in XS Advanced \[page 293\]](#)

[Create a CDS Extension \[page 316\]](#)

### 5.3.1 Create the Data Persistence Artifacts in XS Advanced

Define the artifacts that make up the data-persistence model.

#### Context

CDS artifacts are design-time definitions that are used to generate the corresponding run-time objects, when the CDS document that contains the artifact is deployed to the SAP HANA XS advanced model run time. In XS advanced, the CDS document containing the design-time definitions that you create using the CDS-compliant syntax must have the file extension `.hdbcards`, for example, `MyCDSTable.hdbcards`.

The HDI deployment tools deploy database artifacts to an HDI design-time container. Design-time database objects are typically located in the `db/` folder of the application design-time hierarchy, as illustrated in the following example:

#### Procedure

1. Create the infrastructure in your design-time file-system.

Design-time database objects are typically located in the `db/src/` folder of the application design-time hierarchy, as illustrated in the following example:

#### Sample Code

```
AppName
|- db/                                # Database deployment artifacts
  |- src/                               # Database artifacts: tables, views, etc.
```

2. Define the CDS artifacts.

The data-persistence model can be defined in one central CDS document (for example, `myCDSmodel.hdbc`) or split into separate CDS documents (`myCDSEntity.hdbc` and `myCDSDataType.hdbc`) which describe individual artifacts, as illustrated in the following example:

#### Sample Code

```
AppName
|- db/                                # Database deployment artifacts
  |- src/
    |- myCDSEntity.hdbc                # Database artifacts: tables, views, etc.
    |- myCDSDataType.hdbc              # Database design-time table definition
    \- myCDSmodel.hdbc                 # Database design-time model definition
```

3. Add a package descriptor for the XS advanced application.

The package descriptor (`package.json`) describes the prerequisites and dependencies that apply to the database module of an application in SAP HANA XS advanced. Add a `package.json` file to the root folder of your application's database module (`db/`).

#### Sample Code

```
AppName
|- db/                                # Database deployment artifacts
  |- package.json                      # Database details/dependencies
  |- src/
    |- myCDSEntity.hdbc                # Database artifacts: tables, views, etc.
    |- myCDSDataType.hdbc              # Database design-time table definition
    \- myCDSmodel.hdbc                 # Database design-time definition overview
```

The basic `package.json` file for your database module (`db/`) should look similar to the following example:

#### Sample Code

```
{
  "name": "deploy",
  "dependencies": {
    "@sap/hdi-deploy": "1.0.0"
  },
  "scripts": {
    "start": "node node_modules/@sap/hdi-deploy/deploy.js"
  }
}
```

- Add the container-configuration files required by the SAP HANA Deployment Infrastructure (HDI).

The following design-time artifacts are used to configure the HDI containers:

- Container deployment configuration (`.hdiconfig`)  
Mandatory: A JSON file containing a list of the bindings between database artifact types (for example, sequence, procedure, table) and the corresponding deployment plug-in (and version).
- Run-time container name space rules (`.hdinamespace`)  
Optional: A JSON file containing a list of design-time file suffixes and the naming rules for the corresponding runtime locations.

### Sample Code

```
AppName
|- db/
|  |- package.json
|  |- src/
|  |  |- .hdiconfig
|  |  |- .hdinamespace
|  |  |- myCDSentity.hdbcards
|  |  |- myCDSDataType.hdbcards
|  \- myCDSmodel.hdbcards
# Database deployment artifacts
# Database details/dependencies
# Database artifacts: tables, views, etc.
# HDI build plug-in configuration
# HDI run-time name-space configuration
# Database design-time table definition
# Database design-time table definition
# Database design-time definition overview
```

## Related Information

[Design-Time Database Resources in XS Advanced \[page 202\]](#)

[Maintaining HDI Containers \[page 173\]](#)

[Create the XS Advanced Application Package Descriptor \[page 652\]](#)

### 5.3.1.1 Design-Time Database Resources in XS Advanced

Design-time database resources reside in the database module of your multi-target application.

The design-time representations of your database resources must be collected in the database module of your multi-target application; the database module is a folder structure that you create, for example `/db/` that includes sub folder called `src/` in which you place all the source files. For database applications, the structure of the design-time resources should look something like the following example:

### Sample Code

Database module in an XS Advanced Application

```
<MyAppName>
|- db/
|  |- package.json
|  |- src/
|  |  |- .hdiconfig
|  |  |- .hdinamespace
|  |  |- myEntity.hdbcards
|  |  |- myDataType.hdbcards
|  \- myDoc.hdbcards
# Database deployment artifacts
# Database details/dependencies
# Database artifacts: tables, views, etc.
# HDI build plug-in configuration
# HDI run-time name-space configuration
# Database design-time table definition
# Database design-time type definition
# Database design-time model overview
```

```
| - web/
|   | - xs-app.json
|   \- resources/
|- js/
|   | - start.js
|   | - package.json
|   \- src/
|- security/
|   \- xs-security.json
\- mtad.yaml
```

As illustrated in the example above, the `/db/` folder contains all your design-time database artifacts, for example: tables, views, procedures, sequences, calculation views. In addition to the design-time database artifacts, the database content must also include the following components:

- `package.json`  
A file containing details of dependencies
- `.hdiconfig`  
The HDI container configuration
- `.hdinamespace`  
The run-time name-space configuration for the deployed objects

The HDI deployment tools deploy all database artifacts located in the `db/` folder to an HDI design-time container. At the same time, the HDI tools create the corresponding run-time container and populate the container with the specified catalog objects. The deployed objects can be referenced and consumed by your business applications.

### Caution

If the deployment tools establish that a database run-time artifact has no corresponding design-time definition, the run-time artifact (for example, a table) is dropped from the catalog.

## Deployment Configuration Artifacts

The following files are mandatory for the database component of your deployment. One instance of each configuration artifact is required in the `db/src/` folder, and this instance applies to all sub-folders unless another instance exists in a sub folder:

- `/db/src/.hdiconfig`  
Used to bind the database artifacts you want to deploy (determined by a file suffix, for example, `.hdbcards`) with the appropriate build plug-in. For example, to deploy a CDS data model to an HDI container, you must configure the CDS build plug-in and, in addition, any other plug-ins required.

### Sample Code

```
"hdbcards" : {
    "plugin_name" : "com.sap.hana.di.cds",
    "plugin_version" : "11.1.0" },
"hbdbprocedure" : {
    "plugin_name" : "com.sap.hana.di.procedure",
    "plugin_version": "11.1.0"
}
```

- /db/src/.hdinamespace

Defines naming rules for the run-time objects which are created in the catalog when the application is deployed

#### Sample Code

```
{
  "name"      : "com.sap.hana.db.cds",
  "subfolder" : "append"
}
```

#### Note

For a name-space and build-plugin configuration to take effect, the name-space and build-plugin configuration file must be deployed - in the same way as any other design-time file.

## Related Information

[Create the Data Persistence Artifacts in XS Advanced \[page 200\]](#)

[Defining the Data Model in XS Advanced \[page 172\]](#)

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

### 5.3.1.2 HDI Design-Time Resources and Build Plug-ins

In XS advanced, database design-time resources must be mapped to a build plug-in for deployment purposes.

In XS advanced, design-time artifacts are distinguished by means of a unique file suffix that must be mapped to an HDI build plug-in. The following example of an abbreviated HDI configuration file (.hdiconfig) file illustrates how the design-time artifact types .hdbcalsulationview and .hdbcds are mapped to their corresponding HDI build plug-in:

#### Code Syntax

```
.hdiconfig

  "hdbcalsulationview" : {
    "plugin_name" : "com.sap.hana.di.calculationview",
    "plugin_version": "12.1.0"
  }
  "hdbcds" : {
    "plugin_name" : "com.sap.hana.di.cds",
    "plugin_version": "12.1.0"
  }
```

The following table lists in alphabetical order the design-time artifacts you can develop and deploy with the SAP HANA Deployment Infrastructure (HDI) and describes the artifact's scope. The table also indicates which build plug-in is required to ensure successful deployment of the artifact. For more information about the individual artifact types and the configuration of the corresponding build plug-in, see *Related Links* below.

Table 16: Default File-Suffix to HDI Build Plug-in Mappings

Artifact Suffix	Description/Content	Build Plug-in
csv	Source data for a table-import operation (also hdbtabledata)	com.sap.hana.di.tabledata.source
.? (txt, copy only)	An arbitrary design-time resource	com.sap.hana.di.copyonly
hdbafllangprocedure	Definition of a language procedure for an application function library (AFL)	com.sap.hana.di.afllangprocedure
hdbanalyticprivilege	Definition of an XML-based analytic privilege	com.sap.hana.di.analyticprivilege
hdbcalsulationview	Definition of a calculation view	com.sap.hana.di.calculativew
hdbcds	A document that contains the specification of one or more database objects written with the SAP HANA Core Data Services syntax	com.sap.hana.di.cds
hdbconstraint	Transforms a design-time constraint into a constraint on database tables	com.sap.hana.di.constraint
hdbdropcreatetable	Transforms a design-time table resource into a table database object	com.sap.hana.di.dropcreate
hdbfulltextindex	Full text index definition	com.sap.hana.di.fulltextindex
hdbfunction	Database function definition	com.sap.hana.di.function
hdbgraphworkspace	Definition of a graph work space resource	com.sap.hana.di.graphworkspace
hdbindex	Table index definition	com.sap.hana.di.index
hdbmrjob	Definition of a Hadoop map-remote job	com.sap.hana.di.virtualfunctionpackage.hadoop
jar	Optional mapping, if you want direct access to Hadoop files	com.sap.hana.di.virtualfunctionpackage.hadoop
hdblibrary	A design-time library resource	com.sap.hana.di.library
hdbprocedure	Definition of a database procedure	com.sap.hana.di.procedure
hdbprojectionview	Definition of a projection view	com.sap.hana.di.projectionview

Artifact Suffix	Description/Content	Build Plug-in
hdbprojectionviewconfig	Configuration file for a projection view	com.sap.hana.di.projectionview.config
hdbpublicsynonym	Definition of a public database synonym	com.sap.hana.di.publicsynonym
hdbresultcache	Definition of a result cache	com.sap.hana.di.resultcache
hdbrole	Definition of a database roles	com.sap.hana.di.role
hdbroleconfig	Configuration of database privileges (and other roles) to be included in a database role	com.sap.hana.di.roleconfig
hdbsearchruleset	Definition of search configurations for built-in search procedure	com.sap.hana.di.searchruleset
hdbsequence	Definition of a database sequence	com.sap.hana.di.sequence
hdbsynonym	Database synonym definition	com.sap.hana.di.synonym
hdbsynonymconfig	Configuration file for a database synonym	com.sap.hana.di.synonym.config
hdbstatistics	Statistics definition file	com.sap.hana.di.statistics
hdbstructuredprivilege	Definition of analytic or structured privileges	com.sap.hana.di.structuredprivilege
hdbtable	Table operations	com.sap.hana.di.table
hdbtabledata	Definition of a data-import operation for a database table (also csv)	com.sap.hana.di.tabledata
hdbtabletype	Definition of a table type	com.sap.hana.di.tabletype
hdbtextconfig	Customization of the options used for text analysis	com.sap.hana.di.tabletype
hdbtextdict	Specification of the custom entity types and entity names to be used with text analysis	com.sap.hana.di.textdictionary
hdbtextrule	Specification of the rules (patterns) for extracting complex entities and relationships using text analysis	com.sap.hana.di.textrule

Artifact Suffix	Description/Content	Build Plug-in
hdbtextinclude	Definition of the rules to be used in one or more extraction rule sets for top-level, text analysis	com.sap.hana.di.textrule.include
hdbtextlexicon	Definition of the lists of words used in one or more top-level text analysis rule sets	com.sap.hana.di.textrule.lexicon
hdbtextminingconfig	Customization of the features and options used for text mining.	com.sap.hana.di.textmining.config
hdbtrigger	Database trigger definition	com.sap.hana.di.trigger
hdbview	View definition file	com.sap.hana.di.view
hdbvirtualfunction	Definition of a virtual database function	com.sap.hana.di.virtualfunction
hdbvirtualfunctionconfig	Configuration file for a virtual function	com.sap.hana.di.virtualfunction.config
hdbvirtualprocedure	Definition of a virtual database procedure	com.sap.hana.di.virtualprocedure
hdbvirtualprocedureconfig	Configuration file for the virtual database procedure definition	com.sap.hana.di.virtualprocedure.config
hdbvirtualtable	Definition of a virtual table	com.sap.hana.di.virtualtable
hdbvirtualtableconfig	Virtual table configuration file	com.sap.hana.di.virtualtable.config
properties tags	Properties file for a table-import operation	com.sap.hana.di.tabledata.properties

## Related Information

- [The HDI Container Configuration File \[page 180\]](#)
- [HDI Container Configuration File Syntax \[page 181\]](#)

## 5.3.2 Create a CDS Document (XS Advanced)

A CDS document is a design-time source file that contains definitions of the database objects you want to create in the SAP HANA catalog.

### Context

CDS documents are design-time source files that contain DDL code that describes a persistence model according to rules defined in Core Data Services. CDS documents have the file suffix .hdbc. Deploying the application with the database module that contains the CDS document creates the corresponding catalog objects in the corresponding schema.

**i** Note

The examples of CDS document elements included in this topic are incomplete [ . . . ]; it is intended for illustration purposes only.

### Procedure

1. Start the SAP HANA Web IDE for SAP HANA.

The SAP Web IDE for SAP HANA is available at the following URL:

`https://<HANA_HOST>:53075/`

➔ Tip

To display the URL for the SAP Web IDE for SAP HANA, open a command shell, log on to the XS advanced run time, and run the following command:

`xs app webide --urls`

2. Display the application project to which you want to add a CDS document.

In XS advanced, SAP Web IDE for SAP HANA creates an application within a context of a project. If you do not already have a project, there are a number of ways to create one, for example: by importing it, cloning it, or creating a new one from scratch.

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

Currently, there is only one type of project template available, namely: *Multi-Target Application Project*. Select *Multi-Target Application Project* and choose *Next*.

- c. Type a name for the new MTA project (for example, `myApp`) and choose *Next* to confirm.
- d. Specify details of the new MTA project and choose *Next* to confirm.
- e. Create the new MTA project; choose *Finish*.

3. Create the CDS document that defines the entity you want to create.

Database artifacts such as the ones defined in a CDS document belong in the MTA's database module.

➔ Tip

If you do not already have a database module, right-click the root folder of your new MTA project and, in the context menu, choose ► *New* > *HDB Module* ▶. Name the new database model **db**.

- a. Navigate to the `src/` folder in your application's database module `db/`.
- b. Right-click the folder `myworld/db/src/` and choose ► *New* > *CDS Artifact* ▶ in the context menu.
- c. Name the new CDS artifact **myCDSModel**.

The setup Wizard adds the mandatory suffix for CDS artifacts (`.hdbc`) to the new file name automatically.

4. Define the details of the CDS artifacts.

In the CDS document you just created, for example, `MyCDSModel.hdbc`, add the CDS-definition code to the file. The CDS code describes the CDS artifacts you want to add, for example, entity definitions, type definitions, view definitions. Note that the following code examples are provided for illustration purposes only.

- a. Add structured types, if required.

Use the `type` keyword to define a type artifact in a CDS document. In this example, you add the user-defined types and structured types to the top-level entry in the CDS document, the context `MyCDSModel`.

```
context MyCDSModel {  
    type BusinessKey : String(10);  
    type SString : String(40);  
    type MyStruct  
    {  
        aNumber : Integer;  
        aText : String(80);  
        anotherText : MyString80; // defined in a separate type  
    };  
<[...]>  
};
```

- b. Add a new context, if required.

Contexts enable you to group together related artifacts. A CDS document can only contain one top-level context, for example, `MyModel {}`. Any new context must be **nested** within the top-level entry in the CDS document, as shown in the following example.

```
context MyCDSModel {  
    type BusinessKey : String(10);  
    type SString : String(40);  
    type <[...]>  
    context MasterData {  
        <[...]>  
    };  
    context Sales {  
        <[...]>  
    };  
    context Purchases {  
        <[...]>  
    };  
};
```

- c. Add new entities.

You can add the entities either to the top-level entry in the CDS document (in this example, the context `MyCDSModel`) or to any other context, for example, `MasterData`, `Sales`, or `Purchases`. In this example, the new entities are column-based tables in the `MasterData` context.

```
context MyCDSModel {
    type BusinessKey : String(10);
    type SString : String(40);
    type <[...]>
    context MasterData {
        Entity Addresses {
            key AddressId: BusinessKey;
            City: SString;
            PostalCode: BusinessKey;
            <[...]>
        };
        Entity BusinessPartner {
            key PartnerId: BusinessKey;
            PartnerRole: String(3);
            <[...]>
        };
    };
    context Sales {
        <[...]>
    };
    context Purchases {
        <[...]>
    };
}
```

5. Save the CDS document.

### 5.3.2.1 CDS Editors

The SAP Web IDE for SAP HANA provides editing tools specially designed to help you create and modify CDS documents.

SAP Web IDE for SAP HANA includes dedicated editors that you can use to define data-persistence objects in CDS documents using the DDL-compliant Core Data Services syntax. SAP HANA XS advanced model recognizes the `.hdbcds` file extension required for CDS object definitions and, at deployment time, calls the appropriate plug-in to parse the content defined in the CDS document and create the corresponding run-time object in the catalog. If you right-click a file with the `.hdbcds` extension in the *Project Explorer* view of your application project, SAP Web IDE for SAP HANA provides the following choice of editors in the context-sensitive menu.

- [CDS Text Editor \[page 211\]](#)

View and edit DDL source code in a CDS document as text with the syntax elements highlighted for easier visual scanning.

Right-click a CDS document: [Open With](#) [Text Editor](#)

- [CDS Graphical Editor \[page 211\]](#)

View a graphical representation of the contents of a CDS source file, with the option to edit the source code as text with the syntax elements highlighted for easier visual scanning.

Right-click a CDS document: [Open With](#) [Graphical Editor](#)

## CDS Text Editor

SAP Web IDE for SAP HANA includes a dedicated editor that you can use to define data-persistence objects using the CDS syntax. SAP HANA recognizes the `.hdbcods` file extension required for CDS object definitions and calls the appropriate repository plug-in. If you double-click a file with the `.hdbcods` extension in the [Project Explorer](#) view, SAP Web IDE for SAP HANA automatically displays the selected file in the CDS text editor.

The CDS editor provides the following features:

- Syntax highlights  
The CDS DDL editor supports syntax highlighting, for example, for keywords and any assigned values. To customize the colors and fonts used in the CDS text editor, choose [Tools](#) [Preferences](#) [Code Editor](#) [Editor Appearance](#) and select a theme and font size.

### Note

The CDS DDL editor automatically inserts the keyword `namespace` into any new DDL source file that you create using the [New](#) [CDS Artifact](#) dialog.

The following values are assumed:

- `namespace = <ProjectName>.<ApplDBModuleName>`
- `context = <NewCDSFileName>`

- Keyword completion  
The editor displays a list of DDL suggestions that could be used to complete the keyword you start to enter. To change the settings, choose [Tools](#) [Code Completion](#) in the toolbar menu.
- Code validity  
The CDS text editor provides syntax validation, which checks for parser errors as you type. Semantic errors are only shown when you build the XS advanced application module to which the CDS artifacts belong; the errors are shown in the console tab.
- Comments  
Text that appears after a double forward slash (`//`) or between a forward slash and an asterisk (`/* ... */`) is interpreted as a comment and highlighted in the CDS editor (for example, `//this is a comment`).

## CDS Graphical Editor

The CDS graphical editor provides graphical modeling tools that help you to design and create database models using standard CDS artifacts with minimal or no coding at all. You can use the CDS graphical editor to create CDS artifacts such as entities, contexts, associations, structured types, and so on.

The built-in tools provided with the CDS Graphical Editor enable you to perform the following operations:

- Create CDS files (with the extension `.hdbcods`) using a file-creation wizard.
- Create standard CDS artifacts, for example: entities, contexts, associations (to internal and external entities), structured types, scalar types, ...
- Define technical configuration properties for entities, for example: indexes, partitions, and table groupings.
- Generate the relevant CDS source code in the text editor for the corresponding database model.

- Open in the CDS graphical editor data models that were created using the CDS text editor.

 **Tip**

The built-in tools included with the CDS Graphical Editor are context-sensitive; right-click an element displayed in the CDS Graphical editor to display the tool options that are available.

## Related Information

[Getting Started with the CDS Graphical Editor \[page 996\]](#)

### 5.3.2.2 CDS Documents in XS Advanced

CDS documents are design-time source files that contain DDL code that describes a persistence model according to rules defined in Core Data Services.

In general, CDS works in XS advanced (HDI) in the same way that it does in the SAP HANA XS classic Repository, which is described in the *SAP HANA Developer Guide for SAP HANA Studio* or the *SAP HANA Developer Guide for SAP HANA Web Workbench* listed in the related links below. For XS advanced, however, there are some incompatible changes and additions, which are described in the following sections:

- [General overview \[page 213\]](#)
- [Name Space \[page 214\]](#)
- [@<annotations> \[page 214\]](#)
- [Entity definitions \[page 216\]](#)
- [Structured Types \[page 219\]](#)

 **Note**

The following example of a CDS document for XS advanced is incomplete [...]; it is intended for illustration purposes only.

 **Sample Code**

```
context MyModel {
    type BusinessKey : String(10);
    type SString : String(40);
    type MyStruct
    {
        aNumber      : Integer;
        aText        : String(80);
        anotherText : MyString80; // defined in a separate type
    };
    table type Structure2 { ... };
    context MasterData {
        Entity Addresses {
            key AddressId: BusinessKey;
            City: SString;
            PostalCode: BusinessKey;
            <[...]>
        }
    }
}
```

```

} technical configuration {
    column store;
    index MyIndex1 on (a, b) asc;
    unique index MyIndex2 on (c, s) desc;
    partition by hash (id) partitions 2,
        range (a) (partition 1 <= values < 10, partition values = 10,
    partition others);
    group type Foo group subtype Bar group name Wheeeeezz;
    unload priority <integer_literal>;
};

context Purchases {
<[...]>
};

```

## General Overview

In XS advanced, CDS documents must have the file suffix .hdbcards, for example, `MyCDSDocument.hdbcards`. Each CDS document must contain the following basic elements:

- CDS artifact definitions  
The objects that make up your persistence model, for example: contexts, entities, structured types, and views

For XS advanced applications, the CDS document does not require a `namespace` declaration, and some of the `@<Annotations>` (for example, `@Schema` or `@Catalog`) are either not required or are no longer supported.

Instead, most of the features covered by the `@<Annotations>` in XS classic can now be defined in the `technical configuration` section of the entity definition or in the view definition.

### ➔ Tip

From SAP HANA 2.0 SPS 01, it is possible to define **multiple** top-level artifacts (for example, contexts, entities, etc.) in a single CDS document. For this reason, you can choose any name for the CDS source file; there is no longer any requirement for the name of the CDS source file to be the same as the name of the top-level artifact.

Multiple top-level artifacts are now allowed in a single CDS document, as illustrated in the following example:

### Sample Code

Multiple Top-Level Artifacts in a CDS Document

```

type T : Integer;
entity E {
    key id : Integer;
    a : String(20);
},
context ctx1 {
    ...
};
context ctx2 {
    ...
};

```

## Name Spaces

From SPS12, the declaration of a name space in a CDS document for XS advanced usage is optional.

### i Note

You can only omit the name-space declaration in a CDS document if no name space is defined in the corresponding HDI container-configuration file (`.hdinamespace`).

## @annotations in CDS Documents

When you are defining CDS models for XS advanced application, bear in mind that there are restrictions on the annotations that you can use. In XS advanced, the `technical configuration` section of the corresponding entity definition CDS document is used to define much of what previously was defined in annotations. This section indicates which annotations are (or are not) allowed in CDS documents in XS advanced.

### Supported CDS Annotations in XS Advanced

The following table lists the annotations that **are** supported in CDS models for XS advanced:

Table 17: Supported CDS Annotations in SAP HANA XS Advanced

CDS Annotation	XS Advanced (HDI)
<code>@OData.publish</code>	Expose a CDS context or nested context (and any artifacts the context includes) as an OData version 4 service.

### i Note

For more information about publishing CDS contexts as OData services, see *Related Links*.

## Annotations in Parameter Definitions in CDS Views

In a CDS view, parameter definitions can be annotated in the same way as any other artifact in CDS; the annotations must be prepended to the parameter name. Multiple annotations are separated either by whitespace or new-line characters.

### ➔ Tip

To improve readability and comprehension, it is recommended to include only one annotation assignment per line.

In the following example, the view `TargetWindowView` selects from the entity `TargetEntity`; the annotation `@positiveValuesOnly` is not checked; and the `targetId` is required for the `ON` condition in the entity `SourceEntity`.

## Sample Code

### Annotation Assignments to Parameter Definitions in CDS Views

```
annotation remark: String(100);

view TargetWindowView with parameters
  @remark: 'This is an arbitrary annotation'
  @positiveValuesOnly: true
  LOWER_LIMIT: Integer
as select from TargetEntity
{
  targetId,
  ...
} where targetId > :LOWER_LIMIT and targetId <= :LOWER_LIMIT + 10;
```

## Unsupported CDS Annotations in XS Advanced

The following table lists the annotations that are **not** supported in CDS models for XS advanced:

Table 18: Unsupported CDS Annotations in SAP HANA XS Advanced

CDS Annotation	XS Advanced (HDI)
@Catalog	To specify an index or table type in XS advanced, use the <code>technical configuration</code> section of the corresponding entity definition.
@nokey	In XS advanced, it is possible to define an entity without key elements without using an annotation.
@Schema	In XS advanced, schema handling is performed automatically by the HDI container.
@GenerateTableType	In SAP HANA XS advanced, no SAP HANA table type is generated for a structured type by default. To enforce the generation of an SAP HANA table type in SAP HANA XS advanced, use the keyword <code>table type</code> in a type definition instead of the keyword <code>type</code> , for example, <code>table type Structure2 { ... };</code>
@SearchIndex	To define a search index in XS Advanced, use the <code>technical configuration</code> section of the corresponding entity definition.
@WithStructuredPrivilegeCheck	In XS advanced, you define the privilege check in the view. <pre>view MyView as select from Foo {   &lt;select_list&gt; } &lt;where_groupBy_Having_OrderBy&gt; with structured privilege check;</pre>

## Entity Definitions

The definition of an entity can contain a section called `technical configuration`, which you use to define the elements listed in the following table:

Table 19: Technical Configuration Features for Entities in SAP HANA XS Versions

Configuration Element	XS Advanced
<a href="#">Storage type [page 216]</a>	✓
<a href="#">Indexes [page 217]</a>	✓
<a href="#">Full text indexes [page 217]</a>	✓
<a href="#">Partitioning [page 218]</a>	✓
<a href="#">Grouping [page 218]</a>	✓
<a href="#">Unload priority [page 218]</a>	✓

### i Note

The syntax in the technical configuration section is as close as possible to the corresponding clauses in the SAP HANA SQL `Create Table` statement. Each clause in the technical configuration must end with a semicolon.

## Storage Type

In XS advanced, the `@Catalog.tableType` annotation is not supported; you must use the technical configuration. In the technical configuration for an entity, you can use the `store` keyword to specify the storage type (“row” or “column”) for the generated table, as illustrated in the following example. If no store type is specified, a “column” store table is generated by default.

### Sample Code

```
entity MyEntity {  
    key id : Integer;  
    a : Integer;  
}  
technical configuration {  
    row store;  
};
```

The specification of table **type** is split into separate components. The storage type (`row store` or `column store`) is specified in the technical configuration section of the CDS entity definition; to generate a **temporary** table, use the keyword “`temporary entity`”, as illustrated in the following example:

### Sample Code

```
temporary entity MyEntity2 {  
    ...  
} technical configuration {
```

```
    row store;  
};
```

## Indexes

In XS advanced, the `@Catalog.index` annotation is not supported; you must use the technical configuration. In the technical configuration for an entity, you can use the `index` and `unique index` keywords to specify the index type for the generated table. For example: “`asc`” (ascending) or “`desc`” (descending) describes the index order, and `unique` specifies that the index is unique, where no two rows of data in the indexed entity can have identical key values.

### Sample Code

```
} technical configuration {  
    index MyIndex1 on (a, b) asc;  
    unique index MyIndex2 on (c, s) desc;  
};
```

## Full Text Indexes

In the technical configuration for an entity, you can use the `fulltext index` keyword to specify the full-text index type for the generated table, as illustrated in the following example.

### Sample Code

```
entity MyEntity {  
    key id : Integer;  
    t : String(100);  
    s {  
        u : String(100);  
    };  
} technical configuration {  
    fulltext index MyFTI1 on (t) <fulltext_parameter_list>;  
    fuzzy search index on (s.u);  
};
```

The `<fulltext_parameter_list>` is identical to the standard SAP HANA SQL syntax for `CREATE FULLTEXT INDEX`. A `fuzzy search index` in the technical configuration section of an entity definition corresponds to the `@SearchIndex` annotation in XS classic and the statement “`FUZZY SEARCH INDEX ON`” for a table column in SAP HANA SQL. It is not possible to specify both a full-text index and a fuzzy search index for the same element.

### ⚠ Restriction

In XS advanced, the `@SearchIndex` annotation is not supported; you must use the technical configuration to define which of the columns should be indexed for search capabilities. In XS classic, it is not possible to use both the `@SearchIndex` annotation **and** the technical configuration (for example, `fulltext index`) at the same time. In addition, the full-text parameters `CONFIGURATION` and `TEXT MINING CONFIGURATION` are not supported.

## Partitioning

In the technical configuration for an entity, you can use the `partition by` clause to specify the partitioning information for the generated table, as illustrated in the following example.

### ⚠ Restriction

The `partition by` clause is only supported in XS advanced.

### Sample Code

```
entity MyEntity {  
    key id : Integer;  
    a : Integer;  
} technical configuration {  
    partition by hash (id) partitions 2,  
        range (a) (partition 1 <= values < 10, partition values = 10,  
    partition others);  
};
```

The syntax in the `partition by` clause is identical to the standard SAP HANA SQL syntax for the `PARTITION BY` expression in the HANA SQL `CREATE TABLE` statement.

## Grouping

In the technical configuration for an entity, you can use the `group` clause to specify the partitioning information for the generated table, as illustrated in the following example.

### ⚠ Restriction

The `group` clause is only supported in XS advanced.

### Sample Code

```
entity MyEntity {  
    key id : Integer;  
    a : Integer;  
} technical configuration {  
    group type Foo group subtype Bar group name Wheeeeezz;  
};
```

The syntax in the `group` clause is identical to the standard SAP HANA SQL syntax for the `GROUP OPTION` expression in the HANA SQL `CREATE TABLE` statement.

## Unload Priority

In the technical configuration for an entity, you can use the `Unload Priority` clause to specify the priority for unloading the generated table from memory, as illustrated in the following example.

### ⚠ Restriction

The `unload priority` clause is only supported in XS advanced.

## Sample Code

```
entity MyEntity {  
    <element_list>  
} technical configuration {  
    unload priority <integer_literal>;  
};
```

The syntax in the `unload priority` clause is identical to the standard SAP HANA SQL syntax for the `UNLOAD PRIORITY` expression in the HANA SQL `CREATE TABLE` statement.

## Structured Types

In the SAP HANA XS classic model, for each structured CDS type, an SAP HANA table type is generated by default in the repository. For this reason, in XS classic, the generation of table types could be controlled explicitly by means of the `@GenerateTableType` annotation. In SAP HANA XS advanced, however, no SAP HANA table type is generated for a structured type by default.

In SAP HANA XS advanced, to enforce the generation of an SAP HANA table type you must use the keyword `table type` instead of the keyword `type`, as illustrated in the following example:

### Restriction

The `table type` keyword is only supported in SAP HANA XS advanced.

```
type Struc1 { ... };      // no table type is generated  
table type Struc2 { ... }; // creates a corresponding table type
```

You can define structured types that do not contain any elements, for example, using the keywords `type EmptyStruct { };`. In the example, below the generated table for entity "E" contains only one column: "a".

### Tip

It is not possible to generate an SAP HANA table type for an empty structured type.

```
type EmptyStruct { };  
entity E {  
    a : Integer;  
    s : EmptyStruct;  
};
```

## Related Information

[Create a CDS Document \(XS Advanced\) \[page 208\]](#)

[Create the Data Persistence Artifacts in XS Advanced \[page 200\]](#)

[Defining OData v4 Services for XS Advanced Java Applications \[page 463\]](#)

### 5.3.2.3 External CDS Artifacts in XS Advanced

You can define an artifact in one CDS document by referring to an artifact that is defined in another CDS document.

The CDS syntax enables you to define a CDS artifact in one document by basing it on an “external” artifact - an artifact that is defined in a separate CDS document. Each external artifact must be explicitly declared in the source CDS document with the `using` keyword, which specifies the location of the external artifact, its name, and where appropriate its CDS context.

#### ➔ Tip

The `using` declarations must be located in the header of the CDS document between the `namespace` declaration and the beginning of the top-level artifact, for example, the `context`.

The external artifact can be either a single object (for example, a type, an entity, or a view) or a context. You can also include an optional alias in the `using` declaration, for example, `ContextA.ContextAI as ic`. The alias (`ic`) can then be used in subsequent type definitions in the source CDS document.

#### ⚠ Restriction

For SAP HANA XS advanced deployments, the file name of a design-time CDS artifact must use the extension `.hdbcards`, for example, `ContextA.hdbcards` and `ContextB.hdbcards`.

```
//Filename = Pack1/Distributed/ContextB.hdbcards
namespace Pack1.Distributed;
using Pack1.Distributed::ContextA.T1;
using Pack1.Distributed::ContextA.ContextAI as ic;
using Pack1.Distributed::ContextA.ContextAI.T3 as ict3;
using Pack1.Distributed::ContextA.ContextAI.T3.a as a; // error, is not an
artifact
context ContextB {
    type T10 {
        a : T1;           // Integer
        b : ic.T2;        // String(20)
        c : ic.T3;        // structured
        d : type of ic.T3.b; // String(88)
        e : ict3;         // structured
        x : Pack1.Distributed::ContextA.T1; // error, direct reference not allowed
    };
    context ContextBI {
        type T1 : String(7); // hides the T1 coming from the first using declaration
        type T2 : T1;        // String(7)
    };
}
```

The CDS document `ContextB.hdbcards` shown above uses external artifacts (data types `T1` and `T3`) that are defined in the “target” CDS document `ContextA.hdbcards` shown below. Two `using` declarations are present in the CDS document `ContextB.hdbcards`; one with no alias and one with an explicitly specified alias (`ic`). The first `using` declaration introduces the scalar type `Pack1.Distributed::ContextA.T1`. The second `using` declaration introduces the context `Pack1.Distributed::ContextA.ContextAI` and makes it accessible by means of the explicitly specified alias `ic`.

### Note

If no explicit alias is specified, the last part of the fully qualified name is assumed as the alias, for example T1.

The `using` keyword is the only way to refer to an externally defined artifact in CDS. In the example above, the type `x` would cause an activation error; you cannot refer to an externally defined CDS artifact directly by using its fully qualified name in an artifact definition.

You can use the “`using`” keyword to reference the following SAP HANA artifacts in XS advanced:

- CDS DDL documents (specify which document is used)
- CDS DCL documents (CDS access policy documents)
- Calculation views (.hdbcalsulationview)
- SAP HANA entities (.hbcds)
- SAP HANA user role definitions (.hbrole)

```
//Filename = Pack1/Distributed/ContextA.hbcds
namespace Pack1.Distributed;
context ContextA {
    type T1 : Integer;
    context ContextAI {
        type T2 : String(20);
        type T3 {
            a : Integer;
            b : String(88);
        };
    };
}
```

### Note

Whether you use a single or multiple CDS documents to define your data-persistence model, each CDS document must contain only **one** top-level artifact, and the name of the top-level artifact must correspond to the name of the CDS document. For example, if the top-level artifact in a CDS document is `ContextA`, then the CDS document itself must be named `ContextA.hbcds`.

## Related Information

[Defining the Data Model in XS Advanced \[page 172\]](#)

[Create a CDS Document \(XS Advanced\) \[page 208\]](#)

## 5.3.2.4 CDS Naming Conventions in XS Advanced

Rules and restrictions apply to the names of CDS documents and the package in which the CDS document resides.

The rules that apply for naming CDS documents are the same as the rules for naming the packages in which the CDS document is located. When specifying the name of a package or a CDS document (or referencing the name of an existing CDS object, for example, within a CDS document), bear in mind the following rules:

- CDS source-file name  
From SAP HANA 2.0 SPS 01, it is possible to define **multiple** top-level artifacts (for example, contexts, entities, etc.) in a single CDS document. For this reason, you can choose any name for the CDS source file; there is no longer any requirement that the name of the CDS source file must be the same as the name of a top-level artifact.
- File suffix  
The file suffix differs according to SAP HANA XS version:
  - XS classic  
.hdbdd, for example, `MyModel.hdbdd`
  - XS advanced  
.hdbc, for example, `MyModel.hdbc`
- Permitted characters  
CDS object and package names can include only a subset of ASCII, namely, the following characters:
  - Lower or upper case letters (aA-zZ) and the underscore character (\_)
  - Digits (0-9)
- Forbidden characters  
The following restrictions apply to the characters that you can use (and their position) in the name of a CDS artifact or a package that contains CDS artifacts:
  - The hyphen (-) or the dot (.) are **not** permitted in the name of a CDS document, for example:  
`art-if.act.hdbc`
  - A digit (0-9) is **not** permitted as the first character in the name of either a CDS document or a package that contains CDS documents, for example:
    - XS classic  
`2CDSobjectname.hdbdd`
    - XS advanced  
`acme.com.1package.hdbc`
  - The CDS parser does not recognize either CDS document names or package names that consist **exclusively** of digits, for example:
    - XS classic  
`1234.hdbdd`
    - XS advanced  
`acme.com.999.hdbc`
- Special considerations for artifact names in HDI/XS advanced  
Additional restrictions apply to the names of files and folders in the HDI virtual file system. For more details, see *Related Information*.

### Caution

Although it is possible to use quotation marks ("") to wrap a name that includes forbidden characters, as a general rule, it is recommended to follow the naming conventions for CDS documents specified here in order to avoid problems during activation in the catalog.

## Related Information

[Create a CDS Document \(XS Advanced\) \[page 208\]](#)

[CDS Documents in XS Advanced \[page 212\]](#)

[Naming Conventions in HDI \[page 175\]](#)

### 5.3.2.5 Accessing CDS Metadata in HDI

CDS metadata is available as standard SQL monitoring views and table functions.

As of SP11, any CDS catalog metadata, which was created by the CDS plug-in of the HANA Deployment Infrastructure (HDI), `com.sap.hana.di.cds`, is available as standard SQL monitoring views (`CDS_*`) and one table function (`CDS_ARTIFACT_DEFINITION`). In the same way as any SAP HANA SQL metadata, the views listing CDS metadata are available in schema SYS. The exposure of CDS metadata by means of SQL views enables consumers to combine CDS and database metadata with the SQL JOIN command.

The CDS catalog metadata enables you to reconstruct a CDS source; if the reconstructed source is processed again by the CDS compiler, it produces the same metadata even if the recreated CDS source is not identical to the original source. The result set structure of the main CDS metadata access point, for example, the table function “`CDS_ARTIFACT_DEFINITION`” is minimal. If necessary, it can be joined with database catalog system views and views containing other CDS related metadata in order to enrich the result set with additional, more specific metadata.

### Note

It is very important to control the run time access to the CDS metadata. Although the user has access to the CDS metadata SQL objects (in schema SYS), instance-based access control ensures that the metadata content exposed to a user is only that which the requesting user is authorized to see.

The instance based CDS metadata access control is managed at the HDI run time container schema level by means of a newly introduced schema level object privilege “`SELECT CDS METADATA`”. Since the HDI run time container schema is created and owned by an HDI internal technical user, the CDS metadata privilege cannot be assigned using the usual GRANT command. Instead, the granting must be performed using an built-in administrative stored procedure “`GRANT_CONTAINER_SCHEMA_PRIVILEGES`”; the stored procedure is provided by SAP HANA HDI in a schema which is related to the actual HDI run time container schema (for example, named `<HDI_RT_Container_Name>#DI`).

## ➔ Tip

You can also use HDI container tools to granting the privilege “SELECT CDS METADATA” to an HDI run time container schema.

## Related Information

[CDS Catalog Reader API for HDI \[page 224\]](#)

[Maintaining HDI Containers \[page 173\]](#)

### 5.3.2.6 CDS Catalog Reader API for HDI

The new SQL-based CDS metadata access API in schema SYS is only available CDS content created for (and deployed in) the SAP HANA HDI.

With SPS 11, any CDS catalog metadata that is created by the CDS plugin of the HANA Deployment Infrastructure (HDI), `com.sap.hana.di.cds`, is available as standard SQL monitoring views (`CDS_*`) and one table function (`CDS_ARTIFACT_DEFINITION`). In the same way as HANA SQL metadata, these views are available in schema SYS. The exposure of CDS catalog metadata in SQL views enables consumers to combine CDS and database metadata by means of the SQL JOIN command.

Table 20: CDS Catalog Metadata Views

View Name	Description
<code>CDS_ARTIFACT_DEFINITION</code>	Retrieves CDS metadata for a given CDS artifact name
<code>CDS_ARTIFACT_NAMES</code>	Exposes all CDS artifact names and their kind for all schemas
<code>CDS_ANNOTATION_VALUES</code>	Exposes multiple rows, if a single CDS artifact is annotated with multiple annotation values
<code>CDS_ANNOTATION_ASSIGNMENTS</code>	Returns a flat list of annotation assignments
<code>CDS_ASSOCIATIONS</code>	Expose association metadata and definitions
<code>CDS_ENTITIES</code>	Expose entity (table) metadata and definitions
<code>CDS_VIEWS</code>	Expose view metadata and definitions

### `CDS_ARTIFACT_DEFINITION`

`CDS_ARTIFACT_DEFINITION` is intended to be the main entry point to retrieve CDS metadata for a given CDS artifact name. If no CDS artifact name is known, the `CDS_ARTIFACT_NAMES` view can be used to retrieve all

artifact names for a given schema. The other CDS\_\* monitoring views can be accessed individually, too, but are actually intended for use by joining with the result of CDS\_ARTIFACT\_DEFINITION, to enrich the result with additional, more detailed CDS metadata.

The `CDS_ARTIFACT_DEFINITION` TableFunction expects exactly two mandatory **input** parameters; the parameters specify the requested CDS artifact name in a certain schema; that is, the corresponding HDI runtime container. The input parameters are as follows:

- `SCHEMA_NAME` (NVARCHAR(256))
- `ARTIFACT_NAME` (NVARCHAR(127))

Table 21: Result-Set Structure

Column Name	Type	Description
<code>SCHEMA_NAME</code>	NVARCHAR(256)	The HDI runtime container schema name. The returned schema name always matches the schema name requested when calling the TableFunction. Since HDI allows you to deploy the same CDS artifact name into the same namespace in two different runtime container schemas, it is necessary to specify the schema name as part of the ON-condition of SQL JOINS, if additional metadata is selected from additional monitoring views.
<code>ARTIFACT_NAME</code>	NVARCHAR(127)	The fully qualified CDS artifact name, including the namespace, separated by a double colon “::”, for example, <code>namespaceComponent1.namespaceComponent1::ContextName.EntityName</code>
<code>ELEMENT_NAME</code>	NVARCHAR(127)	CDS element name is populated with the element name, if <code>ARTIFACT_KIND</code> is “ELEMENT”. The name can also be a path with the name components separated by a double colon “::”, where flattened elements of (nested) structures are included.
<code>ARTIFACT_KIND</code>	VARCHAR(32)	The CDS artifact kind, for example, one of the following enumeration values: <ul style="list-style-type: none"> <li>• ANNOTATION</li> <li>• ARRAY_TYPE</li> <li>• ASSOCIATION</li> <li>• ASSOCIATION_ELEMENT</li> <li>• CONTEXT</li> <li>• DERIVED_TYPE</li> <li>• ELEMENT</li> <li>• ENTITY</li> <li>• ENUM</li> <li>• STRUCTURED_TYPE</li> <li>• VIEW</li> </ul>

Column Name	Type	Description
PARENT_ARTIFACT_NAME	NVARCHAR(127)	<p>The name of the parent CDS artifact. NULL for top-level root artifacts.</p> <p><b>► Tip</b></p> <p>The parent relationship is meant in terms of structural definition; not the usage dimension.</p>
PARENT_ARTIFACT_ELEMENT	NVARCHAR(127)	<p>The name of the parent CDS element. The PARENT_ARTIFACT_ELEMENT column has a value in the following cases:</p> <ol style="list-style-type: none"> <li>Anonymous, non-primitive type definitions “behind” elements have the element as parent artifact, for example: “elem: array of String(3);”. The array type definition has the “elem” artifact as parent. The row representing the element definition refers to the anonymous array type definition in the USED_ARTIFACT_NAME column.</li> <li>For flattened elements, non-top-level elements refer to their parent elements, which were also created as part of the flattening process.</li> </ol>
IS_ANONYMOUS	VARCHAR(5)	<p>Specifies whether the artifact definition is anonymous: TRUE or FALSE (Default). Artifact definitions in CDS can be anonymous in the sense that there is no identifier specified for them in the source.</p> <p><b>i Note</b></p> <p>“anonymous” artifact definitions are internally still supplied with a unique artifact name, which is constructed by the convention: Parent artifact name, concatenated with “.” and the relative local component name.</p> <p>Elements are always syntactically named in the CDS source and are therefore always represented as non-anonymous, even if an enclosing structured type definition itself might be anonymous. The same is true for the foreign key elements of managed associations, which are represented with artifact kind ASSOCIATION_ELEMENT.</p>
USED_ARTIFACT_SCHEMA	NVARCHAR(256)	<p>The schema name of the used CDS artifact. (See also column SCHEMA_NAME).</p> <p>The triplet USED_ARTIFACT_SCHEMA, NAME, and KIND is similar to OBJECT_SCHEMA, NAME, and TYPE in SYS.SYNONYMS; it defines a reference to another CDS artifact definition, which is used. These three columns are only filled for usage relationships, not for nested definitions.</p>

Column Name	Type	Description
USED_ARTIFACT_NAME	NVARCHAR(127)	<p>The name of the used CDS artifact. If CDS or SAP HANA database (built-in) primitive types are used, the USED_ARTIFACT_NAME is filled and the kind is PRIMITIVE_TYPE. No value is given for the SCHEMA because CDS built-in types are virtual and do not exist in a actual database schema.</p> <p><b>i Note</b></p> <p>The list of available CDS primitive types is described in the <i>SAP HANA Core Data Services (CDS) Reference</i> available on the SAP Help Portal.</p> <p>CDS built-in primitive types are prefixed with namespace "sap.cds::". SAP HANA types that are usable but deprecated within CDS are located in the top-level context "hana" (prefixed with "hana"). If explicitly defined custom CDS artifacts are used, the USED_ARTIFACT_KIND values will be one of the following: DERIVED_TYPE, ANNOTATION, ARRAY, CONSTANT, or ENUM.</p>
USED_ARTIFACT_KIND	VARCHAR(32)	The kind of CDS artifact used
ORDINAL_NUMBER	INTEGER	<p>The ordinal position of the CDS artifact definition within the returned artifact tree for the requested artifact name. The ordinal number starts with 0 and is increased in pre-order manner during depth-first traversal of the requested artifact tree.</p> <p><b>i Note</b></p> <p>The CDS ordinal number must not be confused with the column order of tables at the database level. The user of the API can JOIN the POSITION information of the SYS.COLUMNS view to prefix the column position in the generated database object with the CDS ORDINAL_NUMBER.</p>
SQL_DATA_TYPE_NAME	VARCHAR(16)	The SQL data type name, which is only filled for elements with built-in CDS primitive types. See also column DATA_TYPE_NAME in SYS.COLUMNS view.
TYPE_PARAM_1	INTEGER	<p>The number of characters for character types; maximum number of digits for numeric types; number of bytes for LOB types; SRID value for GIS (geospatial) types.</p> <p><b>i Note</b></p> <p>Internal values for time and timestamp data type are not exposed. If such information is required, the user of the CDS metadata SQL API has to JOIN the CDS element row with the SYS.COLUMNS view.</p>

Column Name	Type	Description
TYPE_PARAM_2	INTEGER	The numeric types: the maximum number of digits to the right of the decimal point
IS_NULLABLE	VARCHAR(5)	Specifies whether the database column generated for this CDS element is allowed to accept null values. Set to TRUE (default) or FALSE. See also column IS_NULLABLE in SYS.COLUMNS view.
IS_KEY	VARCHAR(5)	Specifies whether the column is part of the primary key, in order of the definition in the source. Set to TRUE or FALSE (default). The order of the columns in the key is given by the order of the elements in the Entity. The key information is also available for the flattened elements, for structured-type usage for elements that are part of the key.
VALUE	NVARCHAR(5000)	Default value in case of ARTIFACT_KIND = 'ELEMENT' and constant value in case of ARTIFACT_KIND = 'CONSTANT'.
AUX_ELEMENT_INFO	NVARCHAR(5000)	This column exposes the following auxiliary element related properties: <ul style="list-style-type: none"> <li>Original name path if used with "as" for foreign key element definitions of managed associations (ARTIFACT_KIND = ASSOCIATION_ELEMENT)</li> <li>The period element kind in case of series entities: PERIOD or ALTERNATE_PERIOD</li> </ul>

## CDS\_ARTIFACT\_NAMES

This view exposes all CDS artifact names and their kind for all schemas the user has the `SELECT_CDS_METADATA` privilege for. It is intended for use with a “WHERE” condition, for example, to restrict the result set to **all** artifacts defined within one single schema (HDI runtime container) or CDS namespace.

Table 22: Result-Set Structure

Column Name	Type	Description
SCHEMA_NAME	NVARCHAR(256)	The name of the SAP HANA HDI run time container schema
ARTIFACT_NAME	NVARCHAR(127)	The fully qualified CDS artifact name, including the namespace, separated by a double colon “::”

Column Name	Type	Description
ARTIFACT_KIND	VARCHAR(32)	<p>The artifact kind, for example, one of the following enumeration values:</p> <ul style="list-style-type: none"> <li>• ANNOTATION</li> <li>• ARRAY_TYPE</li> <li>• ASSOCIATION</li> <li>• CONTEXT</li> <li>• DERIVED_TYPE</li> <li>• ENTITY</li> <li>• ENUM</li> <li>• STRUCTURED_TYPE</li> <li>• VIEW</li> </ul>

## CDS\_ANNOTATION\_VALUES

The view `SYS_RT.CDS_ANNOTATION_VALUES` exposes multiple rows if a single CDS artifact is annotated with multiple annotation values. This allows you to select all artifact names that are annotated with a certain annotation. The values of CDS internal annotations that are prefixed with namespace “`sap.cds::`” are not exposed with this view.

Table 23: Result-Set Structure

Column Name	Type	Description
SCHEMA_NAME	NVARCHAR(256)	The name of the SAP HANA HDI run time container schema
ARTIFACT_NAME	NVARCHAR(127)	The name of the annotated CDS artifact
ELEMENT_NAME	NVARCHAR(127)	Element name, if an element is annotated
ANNOTATION_SCHEMA_NAME	NVARCHAR(256)	Schema name of annotation definition
ANNOTATION_NAME	NVARCHAR(127)	Name of the annotation definition <div style="background-color: #ffffcc; padding: 10px; margin-top: 10px;"> <b>i Note</b> <p>For built-in CDS annotations, no entry can be found in <code>SYS.CDS_ANNOTATIONS</code>. This is because the definitions of built-in CDS annotations are not delivered as CDS catalog content; they are defined during compiler initialization.</p> </div>
VALUE	NCLOB	The annotation value tree, serialized as JSON, for example: <code>{ "value": "&lt;SomeAnnotationValue&gt;" }</code>

## CDS\_ANNOTATION\_ASSIGNMENTS

This view in the CDS catalog API returns a flat list of annotation assignments:

### Sample Code

```
select * from SYS.CDS_ANNOTATION_ASSIGNMENTS
```

The view `SYS_RT.CDS_ANNOTATION_VALUES` exposes multiple rows if a single CDS artifact is annotated with multiple annotation values. This allows you to select all artifact names that are annotated with a certain annotation. The values of CDS internal annotations that are prefixed with namespace “`sap.cds::`” are not exposed with this view.

Table 24: Result-Set Structure

Column Name	Description
SCHEMA_NAME	The schema of the annotated object
ARTIFACT_NAME	The name of the annotated CDS object
COMPONENT_NAME	The name of the annotated element (if the annotation is assigned to an element of an object)
ANNOTATION_NAME	The name of the annotation
EXTENSION_PACKAGE_NAME	The name of the extension package, if the assignment happens in an extension
FORMAT	The format of the annotation value; possible values for FORMAT are: <code>Enum</code> , <code>Boolean</code> , <code>String</code> , <code>Number</code> , <code>Array</code> , and <code>Other</code> .
VALUE	<p>The annotation value</p> <p><b>i Note</b> Values for array-like annotations are always returned as “blob”; flattening stops at arrays.</p>
VALIDATION_RESULT	Possible values for validation results are: <ul style="list-style-type: none"><li>• <b>CORRECT</b> A definition exists for the annotation, and the assignment matches the definition.</li><li>• <b>MISMATCHED</b> A definition exists for the annotation, and the assignment <b>does not</b> match the definition.</li><li>• <b>UNCHECKED</b> No definition exists for the annotation.</li></ul>
DEFINITION_SCHEMA_NAME	The schema of the annotation definition (if a definition exists)

Column Name	Description
DEFINITION_NAME	The name of annotation definition (if a definition exists)

The following is an example of a CDS source:

```
annotation KnownAnno {
    a : Integer;
    b : String(100);
    c : Integer;
    t : UTCTimestamp;
};

@KnownAnno : { a:1 , b:'some value' , c:'no int' , t:'2017-03-02 10:33'}
type T : Integer;
@ArrayAnno : [1,2,3]
type S {
    x : Integer;
    @UnknownAnno : { c:3 , d:'hi there' , e:true}
    y : Integer;
}
```

The following table shows the resulting annotation assignments:

### i Note

The contents of the columns SCHEMA\_NAME and DEFINITION\_SCHEMA\_NAME are not shown.  
SCHEMA\_NAME always shows the name of the schema of the HDI container where the model is deployed.  
This is also true for DEFINITION\_SCHEMA\_NAME, if the annotation has a definition.

Table 25: Annotation Assignments 1

ARTIFACT_NAME	COMPONENT_NAME	ANNOTATION_NAME	FORMAT
S		ArrayAnno	Array
S	y	UnknownAnno.c	Number
S	y	UnknownAnno.d	String
S	y	UnknownAnno.e	Boolean
T		KnownAnno.a	Number
T		KnownAnno.b	String
T		KnownAnno.t	String
T		KnownAnno.c	String

Table 26: Annotation Assignments 2

ARTIFACT_NAME	VALUE	VALIDATION_RESULT	DEFINITION_NAME
S	[1,2,3]	UNCHECKED	

ARTIFACT_NAME	VALUE	VALIDATION_RESULT	DEFINITION_NAME
S	3	UNCHECKED	
S	hi there	UNCHECKED	
S	true	UNCHECKED	
T	1	CORRECT	KnownAnno
T	some value	CORRECT	KnownAnno
T	2017-03-02 10:33	CORRECT	KnownAnno
T	no int	MISMATCHED	KnownAnno

## CDS\_ASSOCIATIONS

Association definitions are already exposed as part of the `CDS_ARTIFACT_DEFINITION` result set, represented by `ARTIFACT_KIND` “ASSOCIATION”. Foreign key elements are also available with kind `ASSOCIATION_ELEMENT`, which allows them to be distinguished from “normal” elements.

The CDS association metadata can be retrieved with the following SQL SELECT statement:

### Sample Code

```
SELECT * FROM CDS_ARTIFACT_DEFINITION('<SomeSchemaName>',
'<SomeCdsArtifactName>') def
LEFT OUTER JOIN SYS.CDS_ASSOCIATIONS assoc
ON def.SCHEMA_NAME = assoc.SCHEMA_NAME AND def.ARTIFACT_NAME =
assoc.ASSOCIATION_NAME
WHERE def.ARTIFACT_KIND = 'ASSOCIATION' OR def.ARTIFACT_KIND =
'ASSOCIATION_ELEMENT';
```

Table 27: Result-Set Structure

Column Name	Type	Description
SCHEMA_NAME	NVARCHAR(256)	The name of the SAP HANA HDI run time container schema
ASSOCIATION_NAME	NVARCHAR(127)	The name of the CDS entity
ASSOCIATION_KIND	NVARCHAR(32)	One of the following supported association-specific enumeration values: <ul style="list-style-type: none"><li>• UNMANAGED</li><li>• FOREIGN_KEY_EXPLICIT</li><li>• FOREIGN_KEY_IMPLICIT</li></ul>

Column Name	Type	Description
TARGET_ARTIFACT_SCHEMA_NAME	NVARCHAR(127)	Name of the schema for the associated CDS artifact.
TARGET_ARTIFACT_NAME	NVARCHAR(127)	Name of the associated CDS artifact <b>⚠ Restriction</b> Only CDS entities are allowed as the target of an association.
TARGET_CARDINALITY_MIN	INTEGER	Minimum cardinality of association target. Default is 0.
TARGET_CARDINALITY_MAX	INTEGER	Maximum cardinality of association target. Default is 1. -1 represents unlimited.
JOIN_CONDITION	NCLOB	The join condition for unmanaged associations (ASSOCIATION_KIND = UNMANAGED)

## CDS\_ENTITIES

Table 28: Result-Set Structure

Column Name	Type	Description
SCHEMA_NAME	NVARCHAR(256)	The name of the SAP HANA HDI run time container schema
ENTITY_NAME	NVARCHAR(127)	The name of the CDS entity
SERIES_KIND	NVARCHAR(32)	One of the following supported series-kind enumeration values: <ul style="list-style-type: none"> <li>• NO_SERIES</li> <li>• NOT_EQUIDISTANT, EQUIDISTANT</li> <li>• EQUIDISTANT_PIECEWISE</li> </ul>

## CDS\_VIEWS

Table 29: Result-Set Structure

Column Name	Type	Description
SCHEMA_NAME	NVARCHAR(256)	The name of the SAP HANA HDI run time container schema
VIEW_NAME	NVARCHAR(127)	The name of the CDS view

Column Name	Type	Description
DEFINITION	NCLOB	The view definition string. Relative names are resolved to absolute names already and constant expressions to concrete values.

## Related Information

[Accessing CDS Metadata in HDI \[page 223\]](#)

### 5.3.2.7 CDS Contexts in XS Advanced

You can define multiple CDS-compliant elements (for example, entities or views) in a single file by assigning them to a [context](#).

The following example illustrates how to assign two simple entities to a context using the CDS-compliant syntax; you store the context-definition file with a specific name and the file extension .hdbcards, for example, MyContext.hdbcards.

#### Tip

From SAP HANA 2.0 SPS 01, it is possible to define **multiple** top-level artifacts (for example, contexts, entities, etc.) in a single CDS document. For this reason, you can choose any name for the CDS source file; there is no longer any requirement that the name of the CDS source file must be the same as the name of a top-level artifact.

In the example below, you must save the context definition “Books” in the CDS document Books.hdbcards. In XS advanced, the name space declaration is optional.

The following code example illustrates how to use the CDS syntax to define multiple design-time entities in a context named Books.

```
namespace com.acme.myapp1;
@OData.publish : true      //OData v4 only
context Books {
    entity Book {
        key AuthorID : String(10);
        key BookTitle : String(100);
        ISBN : Integer not null;
        Publisher : String(100);
    } technical configuration {
        column store;
        unique index MYINDEX1 on (ISBN) desc;
    };
    entity Author {
        key AuthorName : String(100);
        AuthorNationality : String(20);
        AuthorBirthday : String(100);
        AuthorAddress : String(100);
    } technical configuration {
        column store;
        unique index MYINDEX1 on (AuthorNationality) desc;
    };
}
```

```
    } ;  
} ;
```

Activation of the file `Books.hdbcds` containing the context and entity definitions creates the catalog objects “Book” and “Author”.

### i Note

The namespace specified at the start of the file, for example, `com.acme.myapp1` corresponds to the location of the entity definition file (`Books.hdbcds`) in the application-package hierarchy. The namespace is not required in XS advanced scenarios; it is optional and provided only for backwards compatibility with XS classic.

This section includes details of the following concepts and features:

- [Nested Contexts \[page 235\]](#)
- [Name Resolution Rules \[page 236\]](#)
- [OData Services \[page 237\]](#)

## Nested Contexts

The following code example shows you how to define a nested context called `InnerCtx` in the parent context `MyContext`. The example also shows the syntax required when making a reference to a user-defined data type in the nested context, for example, `(field6 : type of InnerCtx.CtxType.b;)`.

The `type of` keyword is only required if referencing an element in an entity or in a structured type; types in another context can be referenced directly, without the `type of` keyword. The nesting depth for CDS contexts is restricted by the limits imposed on the length of the database identifier for the name of the corresponding SAP HANA database artifact (for example, table, view, or type); this is currently limited to 126 characters (including delimiters).

### i Note

The context itself does not have a corresponding artifact in the SAP HANA catalog; the context only influences the names of SAP HANA catalog artifacts that are generated from the artifacts defined in a given CDS context, for example, a table or a structured type.

```
namespace com.acme.myapp1;  
context MyContext {  
    // Nested contexts  
    context InnerCtx {  
        Entity MyEntity {  
            key id : Integer;  
            // <...>  
        } ;  
        Type CtxType {  
            a : Integer;  
            b : String(59);  
        } ;  
    } ;  
    type MyType1 {  
        field1 : Integer;  
        field2 : String(40);  
    } ;
```

```

        field3 : Decimal(22,11);
        field4 : Binary(11);
    };
    type MyType2 {
        field1 : String(50);
        field2 : MyType1;
    };
    type MyType3 {
        field1 : UTCTimestamp;
        field2 : MyType2;
    };
    entity MyEntity1 {
        key id : Integer;
        field1 : MyType3 not null;
        field2 : String(24);
        field3 : LocalDate;
        field4 : type of field3;
        field5 : type of MyType1.field2;
        field6 : type of InnerCtx.CtxType.b; // refers to nested context
        field7 : InnerCtx.CtxType;           // more context references
    } technical configuration {
        unique index IndexA on (field1) asc;
    };
};

```

## Name Resolution Rules

The sequence of definitions inside a block of CDS code (for example, `entity` or `context`) does not matter for the scope rules; a binding of an artifact type and name is valid within the confines of the smallest block of code containing the definition, except in inner code blocks where a binding for the same identifier remains valid. This rules means that the definition of `nameX` in an inner block of code hides any definitions of `nameX` in outer code blocks.

### Note

An identifier may be used before its definition without the need for forward declarations.

```

context OuterCtx
{
    type MyType1 : Integer;
    type MyType2 : LocalDate;
    context InnerCtx
    {
        type Use1 : MyType1;           // is a String(20)
        type Use2 : MyType2;          // is a LocalDate
        type MyType1 : String(20);    //
    };
    type invalidUse : Use1;         // invalid: Use1 is not
                                    // visible outside of InnerCtx
    type validUse   : InnerCtx.Use1; // ok
};

```

No two artifacts (including namespaces) can be defined whose absolute names are the same or are different only in case (for example, `MyArtifact` and `myartifact`), even if their artifact type is different (`entity` and `view`). When searching for artifacts, CDS makes no assumptions which artifact kinds can be expected at certain source positions; it simply searches for the artifact with the given name and performs a final check of the artifact type.

The following example demonstrates how name resolution works with multiple nested contexts. Inside context `NameB`, the local definition of `NameA` shadows the definition of the context `NameA` in the surrounding scope. This means that the definition of the identifier `NameA` is resolved to `Integer`, which does not have a sub-component `T1`. The result is an error, and the compiler does not continue the search for a “better” definition of `NameA` in the scope of an outer (parent) context.

```
context OuterCtx
{
    context NameA
    {
        type T1 : Integer;
        type T2 : String(20);
    };
    context NameB
    {
        type NameA : Integer;
        type Use   : NameA.T1; // invalid: NameA is an Integer
        type Use2  : OuterCtx.NameA.T2; // ok
    };
}
```

## Publishing OData Services

You can use the Boolean `@OData.publish` annotation at the context level to indicate that the annotated CDS Context should be exposed (`true`) as an OData service.

### Restriction

The `@OData.publish` feature only works with OData version 4.

The `@OData.publish` annotation cannot be used to publish individual CDS entities, or CDS views. In addition, the annotation `@OData.publish` annotation cannot be used to publish a CDS context that includes a subcontext.

All contexts defined in a CDS document and annotated with `@OData.publish` are published as OData v4 services. There is no restriction on the number of CDS contexts that can be annotated as `OData.publish : true`. In the following example, the CDS artifacts `MyEntity1` and `MyEntity2` defined in the CDS context `ContextA` are exposed for consumption by OData.

### Sample Code

Publish a CDS Context (and Contents) as an OData v4 Service

```
namespace acme.com;
@OData.publish : true
context ContextA {
    MyEntity1 {
        key ID : Integer;
    };
    MyEntity2 {
        key ID : Integer;
        type Address1
    };
};
```

## Note

Although an annotated context that includes a nested (sub) context cannot be published as an OData service, it is possible to publish the individual nested contexts. However, any nested context annotated for OData publication must not include any nested contexts of its own.

In the following example, the CDS artifacts defined in the subcontexts `SubContextA1` and `SubContextA2` are exposed as OData v4 services.

## Sample Code

### Publish CDS Subcontexts an OData v4 Services

```
namespace acme.com;
context ContextA {
    @OData.publish : true
    context ContextA1 {
        MyEntity3 {
            key ID : Integer;
            name : String(80);
            number : Integer;
        };
        MyEntity4 {
            key ID : Integer;
            name : String(80);
        };
    };
    @OData.publish : true
    context ContextA2 {
        MyEntity5 {
            key ID : Integer;
            field1 : Address1;
        };
        type Address1 {
            a : Integer;
            b : String(40);
        };
    };
}
```

## Namespaces in OData v4 Service URLs

The name of the OData service created from a CDS context in a specified name space is:

`<name.space>._.<context>`. In the OData query, the name space and the context name (which is also the OData service name) are separated by the character combination “`_.`” (dot underscore dot). If no name space is declared in the CDS document, the name space and separator “`_.`” are omitted from the service name; the service name is the same as the context name it is derived from. For example, assuming the context `ContextA` is published as an OData service, the OData service name used in the query is as follows:

- Name space specified (“`acme.com`”) in the CDS document (no nested contexts)  
OData Service Name: `http://<BaseURL>/java/odata/v4/<acme.com>._.<ContextA>/$metadata`
- Name space **not** specified in the CDS document (no nested contexts)  
OData Service Name: `http://<BaseURL>/java/odata/v4/<ContextA>/$metadata`

If the CDS document includes a nested **subcontext** that is annotated for publication as an OData v4 service, you can access the exposed artifacts defined in the nested **subcontext** directly, by specifying the complete context path (separated by a ".") in the query URL, as illustrated in the following examples:

- Subcontext ContextA1  
`https://<BaseUrl>/java/odata/v4/<acme.com>._.<ContextA>.<ContextA1>/$metadata`
- Subcontext ContextA2  
`https://<BaseUrl>/java/odata/v4/<acme.com>._.<ContextA>.<ContextA2>/$metadata`

## Related Information

[Create a CDS Document \(XS Advanced\) \[page 208\]](#)

[Defining OData v2 Services for XS Advanced JavaScript Applications \[page 408\]](#)

[Tutorial: Create OData v4 Services for XS Advanced Java Applications \[page 464\]](#)

### 5.3.2.8 CDS Comment Types

The Core Data Services (CDS) syntax enables you to insert comments into object definitions.

#### Example

##### Comment Formats in CDS Object Definitions

```
namespace com.acme.myapp1;

/**
 * multi-line comment,
 * for doxygen-style,
 * comments and annotations
 */
type Type1 {
    element Fstr:      String( 5000 ); // end-of-line comment
                      Flstr: LargeString;
    /*inline comment*/ Fbin:      Binary( 4000 );
    element Flbin:    LargeBinary;
    Fint:           Integer;
    element Fint64:   Integer64;
    Ffixdec:        Decimal( 34, 34 /* another inline comment */ );
    element Fdec:     DecimalFloat;
    Fflt:           BinaryFloat;
    //complete line comment element Flomdat: LocalDate; LocalDate
temporarily switched off
    //complete line comment          Floctim: LocalTime;
    element Futcdatim: UTCDateTime;
                      Futctstmp: UTCTimestamp;
};
```

## Overview

You can use the forward slash (/) and the asterisk (\*) characters to add comments and general information to CDS object-definition files. The following types of comment are allowed:

- In-line comment
- End-of-line comment
- Complete-line comment
- Multi-line comment

### In-line Comments

The in-line comment enables you to insert a comment into the middle of a line of code in a CDS document. To indicate the start of the in-line comment, insert a forward-slash (/) followed by an asterisk (\*) before the comment text. To signal the end of the in-line comment, insert an asterisk followed by a forward-slash character (/\*) after the comment text, as illustrated by the following example::

```
element Flomdat: /*comment text*/ LocalDate;
```

### End-of-Line Comment

The end-of-line comment enables you to insert a comment at the end of a line of code in a CDS document. To indicate the start of the end-of-line comment, insert two forward slashes (//) before the comment text, as illustrated by the following example::

```
element Flomdat: LocalDate; // Comment text
```

### Complete-Line Comment

The complete-line comment enables you to tell the parser to ignore the contents of an entire line of CDS code. The comment out a complete line, insert two backslashes (//) at the start of the line, as illustrated in the following example:

```
// element Flomdat: LocalDate; Additional comment text
```

### Multi-Line Comments

The multi-line comment enables you to insert comment text that extends over multiple lines of a CDS document. To indicate the start of the multi-line comment, insert a forward-slash (/) followed by an asterisk

(\*) at the start of the group of lines you want to use for an extended comment (for example, `/*`). To signal the end of the multi-line comment, insert an asterisk followed by a forward-slash character (`*/`). Each line between the start and end of the multi-line comment must start with an asterisk (\*), as illustrated in the following example:

```
/*
 *  multiline,
 *  doxygen-style
 *  comments and annotations
 */
```

### 5.3.2.9 CDS Extensions Artifacts

Extend existing artifact definitions with properties stored in an additional, external artifact.

The CDS extension mechanism allows you to add properties to existing artifact definitions without modifying the original source files. In this way, you can split the definition of an artifact across multiple files each of which can have a different life cycle and code owner. For example, a customer can add a new element to an existing entity definition by the following statement:

#### Sample Code

CDS Artifact Extension Syntax

```
extend EntityE with {
    newElement: Integer;
}
```

In the example above, the code illustrated shows how to define a new **element** inside an existing entity (`EntityE`) artifact.

#### i Note

The `extend` statement changes an existing artifact; it does not define any additional artifact.

It is essential to ensure that additional element definitions specified in custom extensions do not break the existing definitions of the base application. This is achieved by adapting the name-search rules and by additional checks for the `extend` statements. For the definition of these rules and checks, it is necessary to define the relationship between an `extend` statement and the artifact definitions, as well as the relationship between an `extend` statement and any additional `extend` statements.

## Organization of Extensions

When you extend an SAP application, you typically add new elements to entities or views; these additional elements usually work together and can, themselves, require additional artifacts, for example, “types” used as element “types”. To facilitate the process, we define an extension package (or package for short), which is a set of `extend` statements, normal artifact definitions (for example, “types” which are used in an `extend`

declaration), and extension relationships (also known as “dependencies”). Each CDS source file belongs to exactly one package; all the definitions in this file contribute to that one (single) package. However, a “package” typically contains contributions from multiple CDS source files.

#### ➔ Tip

It is also possible to use a package to define a clear structure for an application, even if no extensions are involved.

## Package Hierarchies

The extension mechanism can be used by developers as well as SAP industry solutions, partners, and customers. A productive system is likely to have more than one package; some packages might be independent from each other; some packages might depend on other packages. With such a model, we get an acyclic directed graph, with the base application and the extension packages as nodes and the dependencies as edges. This induces a partial order on the packages with the base application as lowest package (for simplicity we also call the base application a package). There is not necessarily a single top package (here: the final customer extension).

It is essential to ensure that which package is semantically self-contained and self-explanatory; avoid defining “micro” packages which can be technically applied individually but have no independent business value.

#### ⚠ Restriction

Circular dependencies between extension packages are not allowed.

## Package Definition

It is necessary to specify which `extend` statements and normal artifact definitions belong to which package and, in addition, on which other packages a package depends. A package is considered to be a normal CDS artifact; it has a name, and a corresponding definition, and its use can be found in the CDS Catalog. An extension package is defined by a special CDS source file with the file suffix `.package.hdbc`.

#### ℹ Note

The full stop (.) **before** the extension-package file name is mandatory.

## Related Information

[Create a CDS Document \(XS Advanced\) \[page 208\]](#)

[Create a CDS Extension \[page 316\]](#)

[The CDS Extension Descriptor \[page 321\]](#)

### 5.3.3 Create a CDS Entity in XS Advanced

Define a design-time **entity** (or table) using the Core Data Services (CDS) syntax.

#### Prerequisites

To complete this task successfully, note the following prerequisites:

- You must have access to an SAP HANA system.
- You must have access to the SAP Web IDE for SAP HANA

##### **i** Note

The permissions defined in the XS advanced role collection `XS_AUTHORIZATION_USER` must be assigned to the user who wants to access the tools included in the SAP Web IDE for SAP HANA.

- You must have already created a development workspace and a multi-target application (MTA) project.
- You must already have created a database module for your MTA application project.
- You must already have set up an HDI container for the CDS artifacts

##### **i** Note

A container setup file (`.hdiconfig`) is required to define which plug-ins to use to create the corresponding catalog objects from your design-time artifacts when the multi-target application (or just the database module) is deployed.

- To view run-time objects, you must have access to the SAP HANA XS advanced run-time tools that enable you to view the contents of the catalog.

##### **i** Note

The permissions defined in the XS advanced role collection `XS_AUTHORIZATION_USER` must be assigned to the user who wants to access the SAP HANA run-time tools.

#### Context

In the SAP HANA database, as in other relational databases, a CDS entity is a table with a set of data elements that are organized using columns and rows. SAP HANA Extended Application Services for XS advanced (XS advanced) enables you to use the CDS syntax to create a database entity as a design-time file. Deploying the

database module that contains the CDS entity creates the corresponding table in the corresponding schema. To create a design-time CDS entity-definition file, perform the following steps:

## Procedure

1. Start the SAP HANA Web IDE for SAP HANA.

The SAP Web IDE for SAP HANA is available at the following URL:

`https://<HANA_HOST>:53075/`

➔ Tip

To display the URL for the SAP Web IDE for SAP HANA, open a command shell, log on to the XS advanced run time, and run the following command:

`xs app webide --urls`

2. Open the application project to which you want to add your CDS entity.
3. Create the CDS entity-definition file.

Browse to the folder in the database module in your application's project workspace, for example, `<MyApp1>/HDB/src` where you want to create the new CDS entity-definition file and perform the following steps:

- a. Right-click the folder where you want to save the CDS entity-definition file and choose **New > CDS Artifact** in the context-sensitive pop-up menu.
  - b. Enter the name of the entity-definition file in the *File Name* box, for example, `MyEntity`.
2. Define the structure of the CDS entity.

If the new entity-definition file is not automatically displayed by the file-creation wizard, double-click the entity-definition file you created in the previous step, for example, `MyEntity.hdbc`, and add the entity-definition code to the file:

Note

The following code example is provided for illustration purposes only.

```
entity MyEntity {
    key Author      : String(100);
    key BookTitle   : String(100);
    ISBN          : Integer not null;
```

```
        Publisher : String(100);
} technical configuration {
    column store;
    unique index MYINDEX1 on (ISBN) desc;
};
```

5. Save the CDS entity-definition file.

Saving the definition persists the file in your local workspace; it does not create any objects in the database catalog.

6. Activate the changes in the catalog.

To activate the new entity definition and generate a corresponding table in the catalog, use the *Build* feature.

- Right-click the new database module in your application project.
- In the context-sensitive pop-up menu, choose ► *Build* ▶.

➔ Tip

You can follow progress of the build in the console at the bottom of the CDS editor.

7. Check that the new entity has been successfully created in the catalog.

➔ Tip

A selection of run-time tools is available in the *Database Explorer* perspective of SAP web IDE for SAP HANA at the following location:

► *Tools* ► *Database Explorer* ▶

In XS advanced, your database run-time objects are located in the HDI container created for your multi-target application's database module; you need to locate and bind to this application-specific container to view its contents. The container name contains the name of the user logged into the SAP Web IDE for SAP HANA, the name of the database module containing the CDS design-time entities, and the string *-hdi-container*, for example:

<XS\_UserName>-ctetig24[...]-<DB\_Module>-*hdi-container*

To bind to the HDI container, in the SAP HANA run-time *Catalog* tool, right-click *Catalog* in the catalog list, and in the *Search HDI Containers* dialog, locate the container to which you want to bind, and choose *Bind*.

## Related Information

[Set up an HDI Container \[page 176\]](#)

[Create a CDS Document \(XS Advanced\) \[page 208\]](#)

### 5.3.3.1 CDS Entities in XS Advanced

In the SAP HANA database, as in other relational databases, a CDS entity is a table with a set of data elements that are organized using columns and rows.

A CDS entity has a specified number of columns, defined at the time of entity creation, but can have any number of rows. Database entities also typically have meta-data associated with them; the meta-data might include constraints on the entity or on the values within particular columns. SAP HANA Extended Application Services (SAP HANA XS) enables you to create a database entity as a design-time file. All design-time files, including your CDS entity definition, can be transported to other SAP HANA systems and, when deployed, used to generate the same catalog objects. You can define the entity using CDS-compliant DDL.

#### i Note

In XS classic, the delivery unit is the medium SAP HANA provides to enable you to assemble all your application-related repository artifacts together into an archive that can be easily exported to other systems. In XS advanced, you add your artifacts to application modules (for example, a database module); the modules are used to define and deploy a multi-target application (MTA).

The following code illustrates an example of a single design-time entity definition using CDS-compliant DDL. The name of the top-level artifact, in the following example, the entity `MyTable` must match the name of the CDS artifact. In the example below, the CDS document must be named `MyTable.hdbcards`. In XS advanced, an optional name space can be declared; it indicates the repository package in which the object the document defines is located.

#### ➔ Tip

From SAP HANA 2.0 SPS 01, it is possible to define **multiple** top-level artifacts (for example, contexts, entities, etc.) in a single CDS document. For this reason, you can choose any name for the CDS source file; there is no longer any requirement for the name of the CDS source file to be the same as the name of the top-level artifact.

#### Code Syntax

CDS Entity definition in XS advanced (`MyTable.hdbcards`)

```
namespace com.acme.myapp1;
entity MyTable {
    key Author : String(100);
    key BookTitle : String(100);
    ISBN : Integer not null;
    Publisher : String(100);
} technical configuration {
    column store;
    unique index MYINDEX1 on (ISBN) desc;
};
```

If you want to create a CDS-compliant database entity definition as a design-time file in SAP HANA XS advanced model, you must create the entity as a flat file and save the file containing the DDL entity dimensions with the suffix `.hdbcards`, for example, `MyTable.hdbcards`. The

### Note

On deployment of the CDS design-time artifact, the file suffix, for example, .hdbcds, is used to determine which runtime plug-in to call during the activation process. The plug-in reads the repository file selected for activation, in this case a CDS-compliant entity, parses the object descriptions in the file, and creates the appropriate runtime objects.

When a CDS document is deployed (or activated), the deployment process generates a corresponding catalog object for each of the artifacts defined in the document; the location in the catalog is determined by the type of object generated. For example, the corresponding database table for a CDS entity definition is generated in the following catalog location:

► <SID> ► Catalog ► <SCHEMA\_NAME> ► Tables ▶

## Entity Element Definition

You can expand the definition of an entity element beyond the element's name and type by using element **modifiers**. For example, you can specify if an entity element is the primary key or **part** of the primary key. The following entity element modifiers are available:

- **key**

Defines if the specified element is the **primary** key or **part** of the primary key for the specified entity.

### Note

Structured elements can be part of the key, too. In this case, all table fields resulting from the flattening of this structured field are part of the primary key.

- **null**

Defines if an entity element can (**null**) or cannot (**not null**) have the value NULL. If neither **null** nor **not null** is specified for the element, the default value **null** applies (except for the **key** element).

- **default <literal\_value>**

Defines the default value for an entity element in the event that no value is provided during an INSERT operation. The syntax for the literals is defined in the primitive data-type specification.

- **generated always as <expression>**

Defines a value that is computed as specified in **<expression>**, for example, **a=b**.

You can also use the SAP HANA SQL clause **generated always as identity** in CDS entity definitions to define a field in the database table that is present in the persistence and has a value that is computed as specified in the expression.

```
entity MyEntity {
    key   MyKey    : Integer;
    key   MyKey2   : Integer null;      // illegal combination
    key   MyKey3   : Integer default 2;
    elem2 : String(20) default 'John Doe';
    elem3 : String(20) default 'John Doe' null;
    elem4 : String default 'Jane Doe' not null;
    elem5 : Integer;
    elem6 : Integer;
    elem7 : Integer generated always as elem5+elem6;
};
```

## Spatial Data

CDS entities support the use of spatial data types such as `hana.ST_POINT` or `hana.ST_GEOMETRY` to store geo-spatial coordinates. Spatial data is data that describes the position, shape, and orientation of objects in a defined space; the data is represented as two-dimensional geometries in the form of points, line strings, and polygons.

## Related Information

[Create a CDS Entity in XS Advanced \[page 243\]](#)

[Entity Element Modifiers \[page 248\]](#)

[CDS Entity Syntax Options in XS Advanced \[page 253\]](#)

### 5.3.3.2 Entity Element Modifiers

Element **modifiers** enable you to expand the definition of an entity element beyond the element's name and type. For example, you can specify if an entity element is the primary key or **part** of the primary key.

#### Example

```
entity MyEntity {
    key MyKey : Integer;
    elem2 : String(20) default 'John Doe';
    elem3 : String(20) default 'John Doe' null;
    elem4 : String default 'Jane Doe' not null;
};

entity MyEntity1 {
    key id : Integer;
    a : integer;
    b : integer;
    c : integer generated always as a+b;
};

entity MyEntity2 {
    autoId : Integer generated [always|by default] as identity ( start with 10
increment by 2 );
    name : String(100);
};
```

#### key

```
key MyKey : Integer;
key MyKey2 : Integer null; // illegal combination
key MyKey3 : Integer default 2;
```

You can expand the definition of an entity element beyond the element's name and type by using element **modifiers**. For example, you can specify if an entity element is the primary key or **part** of the primary key. The following entity element modifiers are available:

- **key**

Defines if the element is the **primary** key or **part** of the primary key for the specified entity. You **cannot** use the **key** modifier in the following cases:

- In combination with a **null** modifier. The **key** element is **non null** by default because NULL cannot be used in the **key** element.

**i Note**

Structured elements can be part of the key, too. In this case, all table fields resulting from the flattening of this structured field are part of the primary key.

## null

```
elem3 : String(20) default 'John Doe' null;
elem4 : String default 'Jane Doe' not null;
```

**null** defines if the entity element **can** (**null**) or **cannot** (**not null**) have the value NULL. If neither **null** nor **not null** is specified for the element, the default value **null** applies (except for the **key** element), which means the element **can** have the value NULL. If you use the **null** modifier, note the following points:

**⚠ Caution**

The keywords **nullable** and **not nullable** are no longer valid; they have been replaced for SPS07 with the keywords **null** and **not null**, respectively. The keywords **null** and **not null** must appear at the end of the entity element definition, for example, `field2 : Integer null;`.

- The **not null** modifier can only be added if the following is true:
  - A default is also defined
  - no null data is already in the table
- Unless the table is empty, bear in mind that when adding a new **not null** element to an existing entity, you must declare a default value because there might already be existing rows that do not accept NULL as a value for the new element.
- **null** elements with default values are permitted
- You cannot combine the element **key** with the element modifier **null**.
- The elements used for a unique index must have the **not null** property.

```
entity WithNullAndNotNull
{
    key id : Integer;
    field1 : Integer;
    field2 : Integer null; // same as field1, null is default
    field3 : Integer not null;
};
```

## default

```
default <literal_value>
```

For each scalar element of an entity, a default value can be specified. The `default` element identifier defines the default value for the element in the event that no value is provided during an `INSERT` operation.

### i Note

The syntax for the literals is defined in the primitive data-type specification.

```
entity WithDefaults
{
    key id : Integer;
    field1 : Integer      default -42;
    field2 : Integer64    default 9223372036854775807;
    field3 : Decimal(5, 3) default 12.345;
    field4 : BinaryFloat  default 123.456e-1;
    field5 : LocalDate    default date'2013-04-29';
    field6 : LocalTime    default time'17:04:03';
    field7 : UTCDateTime  default timestamp'2013-05-01 01:02:03';
    field8 : UTCTimestamp default timestamp'2013-05-01 01:02:03';
    field9 : Binary(32)   default x'0102030405060708090a0b0c0d0e0[...]';
    field10 : String(10)  default 'foo';
};
```

## generated always as <expression>

```
entity MyEntity1 {
    key id : Integer;
    a : integer;
    b : integer;
    c : integer generated always as a+b;
};
```

The SAP HANA SQL clause `generated always as <expression>` is available for use in CDS entity definitions; it specifies the expression to use to generate the column value at run time. An element that is defined with `generated always as <expression>` corresponds to a field in the database table that is present in the persistence and has a value that is computed as specified in the expression, for example, “`a+b`”.

“Generated” fields and “calculated” field differ in the following way. **Generated** fields are physically present in the database table; values are computed on `INSERT` and need not be computed on `SELECT`. **Calculated** fields are not actually stored in the database table; they are computed when the element is “selected”. Since the value of the **generated** field is computed on `INSERT`, the expression used to generate the value must not contain any non-deterministic functions, for example: `current_timestamp`, `current_user`, `current_schema`, and so on.

### ⚠ Restriction

The `generated always as <expression>` clause is only supported for column tables.

## generated [always | by default] as identity

```
entity MyEntity2 {  
    autoID : Integer generated always as identity ( start with 10 increment by  
2 );  
    name : String(100);  
};
```

The SAP HANA SQL clause `generated as identity` is available for use in CDS entity definitions; it enables you to specify an identity column. An element that is defined with `generated as identity` corresponds to a field in the database table that is present in the persistence and has a value that is computed as specified in the sequence options defined in the `identity` expression, for example, `( start with 10 increment by 2 )`.

In the example illustrated here, the name of the generated column is `autoID`, the first value in the column is "10"; the `identity` expression `( start with 10 increment by 2 )` ensures that subsequent values in the column are incremented by 2, for example: 12, 14, and so on.

### ⚠ Restriction

The `generated as identity` clause is only supported for column tables.

You can use either `always` or `by default` in the clause `generated as identity`, as illustrated in the examples in this section. If `always` is specified, then values are **always** generated; if `by default` is specified, then values are generated **by default**.

```
entity MyEntity2 {  
    autoID : Integer generated by default as identity ( start with 10 increment  
by 2 );  
    name : String(100);  
};
```

### ⚠ Restriction

CDS does not support the use of reset queries, for example, `RESET BY <subquery>`.

## Column Migration Behavior

The following table shows the migration strategy that is used for modifications to any given column; the information shows which actions are performed and what strategy is used to preserve content. During the migration, a comparison is performed on the column **type**, the generation **kind**, and the expression, if available. From an end-user perspective, the result of a column modification is either a preserved or new value. The aim of any modification to an entity (table) is to cause as little loss as possible.

- Change to the column **type**  
In case of a column type change, the content is converted into the new type. HANA conversion rules apply.
- Change to the expression clause  
The expression is re-evaluated in the following way:
  - “early”

- As part of the column change
    - “late”
  - As part of a query
  - Change to a calculated column
    - A calculated column is transformed into a plain column; the new column is initialized with NULL.
- Technically, columns are either dropped and added or a completely new “shadow” table is created into which the existing content is copied. The shadow table will then replace the original table.

Table 30:

Before column/ After row	Plain	As <expr>	generated always as <expr>	generated always as identity <expr>	generated by default as identity <expr>
<b>Plain</b>	Migrate	Drop/add	Drop/add	Migrate	Migrate
	Keep content	Evaluate on select	Evaluate on add	Keep content	Keep content
<b>generated by default as identity &lt;expr&gt;</b>	Migrate	Drop/add	Drop/add	Migrate	Migrate
	Keep content	Evaluate on select	Evaluate on add	Keep content	Keep content
<b>generated always as identity &lt;expr&gt;</b>	Migrate	Drop/add	Drop/add	Migrate	Migrate
	Keep content	Evaluate on select	Evaluate on add	Keep content	Keep content
<b>generated always as &lt;expr&gt;</b>	Drop/add	Drop/add	Drop/add	Drop/add	Migrate
	NULL	Evaluate on select	Evaluate on add	Keep content	Keep content
<b>as &lt;expr&gt;</b>	Drop/add	Drop/add	Drop/add	Drop/add	Migrate
	NULL	Evaluate on select	Evaluate on add	Keep content	Keep content

## Related Information

[SAP HANA SQL and System Views Reference \(CREATE TABLE\)](#)

### 5.3.3.3 CDS Entity Syntax Options in XS Advanced

The CDS syntax specifies a number of options you can use to define an **entity** (table) in a design-time artifact.

#### Example

##### Note

This example is not a working example; it is intended for illustration purposes only.

```
namespace Pack1."pack-age2";
context MyContext {
    entity MyEntity1 {
        key id : Integer;
        name : String(80);
    };
    entity MyEntity2 {
        key id : Integer;
        x : Integer;
        y : Integer;
        a : Integer;
        field7 : Decimal(20,10) = power(ln(x)*sin(y), a);
    } technical configuration {
        column store;
        unique index Index1 on (x, y) desc;
        index Index2 on (x, a) desc;
        index Index3 on (y asc, a desc);
        partition by <partition_clause>;
        group <grouping_clause>;
        unload priority 0;
        no auto merge;
    };
    entity MyEntity {
        key id : Integer;
        a : Integer;
        b : Integer;
        c : Integer;
        s {
            m : Integer;
            n : Integer;
        };
    } technical configuration {
        row store;
        index MyIndex1 on (a, b) asc;
        unique index MyIndex2 on (c, s) desc;
        fulltext index MYFTI1 on (t)
        FUZZY SEARCH INDEX off;
    };
    temporary entity MyTempEntity {
        a : Integer;
        b : String(20);
    } technical configuration {
        column store;
    };
};
context MySpatialContext {
    entity Address {
        key id : Integer;
        street_number : Integer;
        street_name : String(100);
        zip : String(10);
        city : String(100);
        loc : hana.ST_POINT(4326);
    };
};
```

```

    };
};

context MySeriesContext {
    entity MySeriesEntity {
        key setId : Integer;
        t : UTCTimestamp;
        value : Decimal(10,4);
        series (
            series key (setId)
            period for series (t)
            equidistant increment by interval 0.1 second
            equidistant piecewise //increment or piecewise; not both
        )
    };
}

```

### Restriction

For series data, you can use either `equidistant` or `equidistant piecewise`, but not both at the same time. The example above is for illustration purposes only.

## Overview

Entity definitions resemble the definition of structured types, but with the following additional features:

- [Key definition \[page 255\]](#)
- [Calculated Fields \[page 255\]](#)
- [Technical Configuration \[page 256\]](#)  
Includes: index, unique index, full-text index, table type (row/column), partitioning, grouping, unload priority, and auto-merge options.
- [Spatial data \[page 262\]](#) \*
- [Series Data \[page 262\]](#) \*

On deployment in the SAP HANA XS advanced, each entity definition in CDS is used to generate a database table.

### Tip

From SAP HANA 2.0 SPS 01, it is possible to define **multiple** top-level artifacts (for example, contexts, entities, etc.) in a single CDS document. For this reason, you can choose any name for the CDS source file; there is no longer any requirement for the name of the CDS source file to be the same as the name of the top-level artifact.

The name of the table generated for each entity definition is built according to the same rules as for table types, for example, `Pack1.Pack2::MyModel.MyContext.MyTable`. In addition, the CDS name is restricted by the limits imposed on the length of the database identifier for the name of the corresponding SAP HANA database artifact (for example, table, view, or type); this is currently limited to 126 characters (including delimiters).

### Note

As of SPS 12, the name-space definition is optional.

## Key Definition

```
type MyStruc2
{
    field1 : Integer;
    field2 : String(20);
};

entity MyEntity2
{
    key id : Integer;
    name   : String(80);
    key str : MyStruc2;
};
```

Usually an entity must have a key; you use the keyword `key` to mark the respective elements. The key elements become the primary key of the generated SAP HANA table and are automatically flagged as `not null`. Key elements are also used for managed associations. Structured elements can be part of the key, too. In this case, all table fields resulting from the flattening of this structured element are part of the primary key.

## Calculated Fields

The definition of an entity can contain calculated fields, as illustrated in type “z” the following example:

### Sample Code

```
entity MyCalcField {
    a : Integer;
    b : Integer;
    c : Integer = a + b;
    s : String(10);
    t : String(10) = upper(s);
    x : Decimal(20,10);
    y : Decimal(20,10);
    z : Decimal(20,10) = power(ln(x)*sin(y), a);
};
```

The calculation expression can contain arbitrary expressions and SQL functions. The following restrictions apply to the expression you include in a calculated field:

- The definition of a calculated field must not contain other calculated fields, associations, aggregations, or subqueries.
- A calculated field cannot be key.
- No index can be defined on a calculated field.
- A calculated field cannot be used as foreign key for a managed association.

In a query, calculated fields can be used like ordinary elements.

### Note

In SAP HANA tables, you can define columns with the additional configuration “GENERATED ALWAYS AS”. These columns are physically present in the table, and all the values are stored. Although these columns behave for the most part like ordinary columns, their value is computed upon insertion rather than specified

in the `INSERT` statement. This is in contrast to calculated field, for which no values are actually stored; the values are computed upon `SELECT`.

## technical configuration

The definition of an entity can contain a section called `technical configuration`, which you use to define the elements listed in the following table:

- Storage type (row/column) [page 256]
- Indexes [page 257]
- Full text indexes [page 257]
- Partitioning [page 258]
- Grouping [page 260]
- Unload priority [page 261]
- Auto merge option [page 261]

### i Note

The syntax in the technical configuration section is as close as possible to the corresponding clauses in the SAP HANA SQL `Create Table` statement. Each clause in the technical configuration must end with a semicolon (";").

## Storage Type and Table Types

In the technical configuration for an entity, you can use the `store` keyword to specify the storage type ("row" or "column") for the generated table, as illustrated in the following example. If no store type is specified, a "column" store table is generated by default.

### Sample Code

#### Storage Type

```
entity MyEntity {  
    key id : Integer;  
    a : Integer;  
    b : Integer;  
    t : String(100);  
    s {  
        u : String(100);  
    };  
} technical configuration {  
    row store;  
};
```

To specify a table of type "global temporary", use the `temporary entity` keywords in the CDS entity definition, as illustrated in the following example. To specify a "global temporary table" with the type **column**, use the `temporary entity` keywords and, in addition, define the storage type as `column`, as illustrated in the following examples.

## Sample Code

### Temporary Table Types

```
context MyContext1 {
    temporary entity MyEntity3 {
        ID : Integer;
        name : String(30);
    };
    temporary entity MyTempEntity {
        a : Integer;
        b : String(20);
    } technical configuration {
        column store;
    };
};
```

## Indexes

In the technical configuration for an entity, you can use the `index` and `unique index` keywords to specify the index type for the generated table. For example, `unique` generates an index where no two sets of data in the indexed entity can have identical key values. You can use the keywords “`asc`” (ascending) or “`desc`” (descending) to specify the order of the index.

## Sample Code

```
entity MyEntity {
    key id : Integer;
    a : Integer;
    b : Integer;
    t : String(100);
    s {
        u : String(100);
    };
} technical configuration {
    index MyIndex1 on (a, b) asc;
    unique index MyIndex2 on (c, s) desc;
    index MyIndex3 on (b asc, t desc);
};
```

### Note

You specify the index order (ascending or descending) for individual rows or columns, for example, `(b asc, t desc, s.u desc)`.

## Full text indexes

In the technical configuration for an entity, you can use the `fulltext index` keyword to specify the full-text index type for the generated table, as illustrated in the following example.

## Sample Code

```
entity MyEntity {
    key id : Integer;
    a : Integer;
    b : Integer;
    t : String(100);
```

```

        s {
            u : String(100);
        };
    } technical configuration {
        row store;
        index MyIndex1 on (a, b) asc;
        unique index MyIndex2 on (a, b) asc;
        fulltext index MYFTI1 on (t)
            LANGUAGE COLUMN t
            LANGUAGE DETECTION ('de', 'en')
            MIME TYPE COLUMN s.u
            FUZZY SEARCH INDEX off
            PHRASE INDEX RATIO 0.721
            SEARCH ONLY off
            FAST PREPROCESS off
            TEXT ANALYSIS off;
        fuzzy search index on (s.u);
    };

```

The `fulltext index` is identical to the standard SAP HANA SQL syntax for `CREATE FULLTEXT INDEX`. A `FUZZY SEARCH INDEX` in the technical configuration section of an entity definition corresponds to the `@SearchIndex` annotation in XS classic and the statement "`FUZZY SEARCH INDEX ON`" for a table column in SAP HANA SQL. It is not possible to specify both a full-text index and a fuzzy search index for the same element.

## Partitioning

In the technical configuration of an entity definition, you can specify the partitioning information for the generated table, as illustrated in the following example:

### Sample Code

#### Partition Specification

```

entity MyEntity {
    key id : Integer;
    a : Integer;
    b : Integer;
    t : String(100);
    s {
        u : String(100);
    };
} technical configuration {
    <partition_clause>;
};

```

The code required in the `<partition_clause>` is identical to the corresponding clause in the standard HANA SQL `CREATE TABLE` statement, as illustrated in the following example:

### Sample Code

#### Partition Specification

```

entity MyEntity {
    key id : Integer;
    a : Integer;
    b : Integer;
    t : String(100);
} technical configuration {
    partition by hash (id) partitions 2,

```

```
        range (a) (partition 1 <= values < 10,
                    partition values = 10,
                    partition others);
    };
```

The `partition by` clause defines the rules to use to partition the corresponding table, for example, using hash, range, or round-robin partition rules.

### Note

You can use the `partition by` clause to ensure any partitions added after activation of a CDS entity (for example, with an SQL statement `ALTER TABLE <MyEntity> PARTITION BY RANGE("id")`) are retained on reactivation of the original CDS entity.

Normally, any external changes to a table that was originally created by activating a CDS entity definition are lost on reactivation; the partitions defined after the entity activation are lost. With the `keeping existing layout` option, you can use the `partition by` clause to preserve an existing partitioning, if possible. It is not possible to preserve the added partitions on reactivation of the CDS entity, if the fields used in the external partition specification have changed in the entity definition in a way that no longer fits the partition.

### Sample Code

Retain existing partitions on reactivation

```
entity MyEntity {
    ...
} technical configuration {
    partition by keeping existing layout;
};
```

### Migration Disabled

If the technical configuration of an entity contains the clause “`migration disabled`”, the activation of the CDS source is only allowed if changes in the entity definition do not lead to a migration of the table. If a migration is required, the activation fails and the changes need to be made manually.

### Sample Code

```
entity MyEntity {
    key id : Integer;
    name : String(100);
    value : Decimal(10,2);
} technical configuration {
    migration disabled;
};
```

If the entity defined in the previous example is changed in a way that the corresponding table can be adapted via `ALTER` statements, the activation of the modified table will succeed. This is typically the case for adding or removing elements (as long they are not “`key`” elements) or adding and removing indexes, as illustrated in the following example.

### Sample Code

```
entity MyEntity {
```

```
key id : Integer;
name : String(120);
name2 : String(80);
someTime: LocalTime;
} technical configuration {
migration disabled;
};
```

Changing an element **type** in the way illustrated in the following example is not allowed; activating the following CDS document fails because the change of type in elements “name” and“name2” mean that a migration is required, which is explicitly forbidden by the `(migration disabled` clause.

### Sample Code

```
entity MyEntity {
key id : Integer;
name : String(120);
name2 : String(80);
someTime: LocalTime;
} technical configuration {
migration disabled;
};
```

## Grouping

In the `technical configuration` of an entity definition, you can specify the grouping information for the generated table, as illustrated in the following example:

### Sample Code

#### Grouping Specification

```
entity MyEntity {
key id : Integer;
a : Integer;
b : Integer;
t : String(100);
s {
    u : String(100);
};
} technical configuration {
<grouping_option>;
};
```

The code required in the `<grouping_option>` is identical to the corresponding clause in the standard HANA SQL `CREATE TABLE` statement, as illustrated in the following example:

### Sample Code

#### Grouping Specification

```
entity MyEntity {
key id : Integer;
a : Integer;
b : Integer;
t : String(100);
} technical configuration {
group type Foo group subtype Bar group name Wheeeeezz;
```

```
};
```

You must set the `group type`, the `group subtype`, and the `group name`.

## Unload Priority

In the `technical configuration` of an entity definition, you can specify the unload priority for the generated table, as illustrated in the following example:

### Sample Code

#### Unload Priority Specification

```
entity MyEntity {
    key id : Integer;
    a : Integer;
    b : Integer;
    t : String(100);
    s {
        u : String(100);
    };
} technical configuration {
    unload priority <integer_literal>;
};
```

`unload priority` specifies the priority with which a table is unloaded from memory. The priority can be set between 0 (zero) and 9 (nine), where 0 means “cannot be unloaded” and 9 means “earliest unload”.

## Auto-Merge Option

In the `technical configuration` of an entity definition, you can specify any automatic-merge options for the generated table, as illustrated in the following example:

### Sample Code

#### Auto-Merge Option

```
entity MyEntity {
    key id : Integer;
    a : Integer;
    b : Integer;
    t : String(100);
    s {
        u : String(100);
    };
} technical configuration {
    [no] auto merge;
};
```

### Note

`auto merge;` triggers an automatic delta merge; `no auto merge;` disables the automatic delta merge operation.

## Spatial Types \*

The following example shows how to use the spatial type `ST_POINT` in a CDS entity definition. In the example entity `Person`, each person has a home address and a business address, each of which is accessible via the corresponding associations. In the `Address` entity, the geo-spatial coordinates for each person are stored in element `loc` using the spatial type `ST_POINT`(\*).

### Sample Code

```
context SpatialData {
    entity Person {
        key id : Integer;
        name : String(100);
        homeAddress : Association[1] to Address;
        officeAddress : Association[1] to Address;
    };
    entity Address {
        key id : Integer;
        street_number : Integer;
        street_name : String(100);
        zip : String(10);
        city : String(100);
        loc : hana.ST_POINT(4326);
    };
    view CommuteDistance as select from Person {
        name,
        homeAddress.loc.ST_Distance(officeAddress.loc) as distance
    };
};
```

## Series Data \*

CDS enables you to create a table to store series data by defining an entity that includes a `series()` clause as an table option and then defining the appropriate parameters and options.

### Note

The `period` for `series` must be unique and should not be affected by any shift in timestamps.

### Sample Code

```
context SeriesData {
    entity MySeriesEntity1 {
        key setId : Integer;
        t : UTCTimestamp;
        value : Decimal(10,4);
        series(
            series key (setId)
            period for series (t)
            equidistant increment by interval 0.1 second
        );
    };
    entity MySeriesEntity2 {
        key setId : Integer;
```

```

        t : UTCTimestamp;
        value : Decimal(10, 4);
        series (
            series key (setId)
            period for series (t)
            equidistant piecewise
        );
    };
}

```

CDS also supports the creation of a series table called `equidistant piecewise` using Formula-Encoded Timestamps (FET). This enables support for data that is not loaded in an order that ensures good compression. There is no a-priori restriction on the timestamps that are stored, but the data is expected to be well approximated as piecewise linear with some jitter. The timestamps do not have a single slope/offset throughout the table; rather, they can change within and among series in the table.

### **Restriction**

The `equidistant piecewise` specification can only be used in CDS; it cannot be used to create a table with the SQL command `CREATE TABLE`.

When a series table is defined as `equidistant piecewise`, the following restrictions apply:

1. The `period` includes one column (instant); there is no support for interval periods.
2. There is no support for `missing elements`. These could logically be defined if the period includes an interval start and end. Missing elements then occur when we have adjacent rows where the end of the interval **does not** equal the start of the interval.
3. The type of the period column must map to the one of the following types: `DATE`, `SECONDDATE`, or `TIMESTAMP`.

### **Caution**

(\*) 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 options is available in SAP Help Portal. 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.

## 5.3.4 Create a CDS User-Defined Structure in XS Advanced

Define a design-time custom **structured type** using the Core Data Services (CDS) syntax.

### Prerequisites

To complete this task successfully, note the following prerequisites:

- You must have access to an SAP HANA system.
- You must have access to the SAP Web IDE for SAP HANA

#### i Note

The permissions defined in the XS advanced role collection `XS_AUTHORIZATION_USER` must be assigned to the user who wants to access the tools included in the SAP Web IDE for SAP HANA.

- You must have already a development workspace and a multi-target application (MTA) project.
- You must already have created a database module for your MTA application project.
- You must already have set up an HDI container for the CDS artifacts

#### i Note

A container setup file (`.hdiconfig`) is required to define which plug-ins to use to create the corresponding catalog objects from the design-time artifacts when the multi-target application (or just the database module) is deployed.

- You must have access to the SAP HANA XS advanced run-time tools that enable you to view the contents of the catalog.

#### i Note

The permissions defined in the XS advanced role collection `XS_AUTHORIZATION_USER` must be assigned to the user who wants to access the SAP HANA run-time tools.

### Context

A structured type is a data type comprising a list of attributes, each of which has its own data type. SAP HANA Extended Application Services for XS advanced model (XS advanced) enables you to use the CDS syntax to create a user-defined structured type as a design-time file. Repository files are transportable. Deploying the

CDS document containing the type definition creates the corresponding types in the database catalog. To create a CDS document that defines one or more structured types, perform the following steps:

## Procedure

1. Start the SAP HANA Web IDE for SAP HANA.

The SAP Web IDE for SAP HANA is available at the following URL:

`https://<HANA_HOST>:53075/`

➔ Tip

To display the URL for the SAP Web IDE for SAP HANA, open a command shell, log on to the XS advanced run time, and run the following command:

`xs app webide --urls`

2. Open the application project to which you want to add a CDS-compliant, user-defined structured type.
3. Create the CDS entity-definition file.

Browse to the folder in the database module in your project workspace, for example, `<MyApp1>/HDB/src` where you want to create the new CDS type-definition file and perform the following steps:

- a. Right-click the folder where you want to save the CDS type-definition file and choose **New > CDS Artifact** in the context-sensitive pop-up menu.
  - b. Enter the name of the entity-definition file in the **File Name** box, for example, `MyStructuredType`.
- c. Choose **Finish** to save the new CDS type-definition file in the database module of your local project workspace.
  4. Define the details of the new CDS structured type.

If the new type-definition file is not automatically displayed by the file-creation wizard, double-click the type-definition file you created in the previous step, for example, `MyStructuredType.hdbc`, and add the type-definition code to the file:

Note

The following code example is provided for illustration purposes only.

```
table type MyStructuredType
{
    aNumber : Integer;
    someText : String(80);
    otherText : String(80);
```

```
};
```

5. Save the CDS type-definition file.

Saving the definition persists the file in your local workspace; it does not create any objects in the database catalog.

6. Activate the changes in the catalog.

To activate the new entity definition and generate a corresponding table in the catalog, use the *Build* feature.

- a. Right-click the new database module in your application project.
- b. In the context-sensitive pop-up menu, choose ► *Build* ▶.

➔ Tip

You can follow progress of the build in the console at the bottom of the CDS editor.

7. Check that the new custom structured type has been successfully created in the catalog.

➔ Tip

A selection of run-time tools is available in the *Database Explorer* perspective of SAP web IDE for SAP HANA at the following location:

► *Tools* ► *Database Explorer* ▶

In XS advanced, your database run-time objects are located in the HDI container created for your multi-target application's database module; you need to locate and bind to this application-specific container if you want to view its contents. The container name contains the name of the user logged into the SAP Web IDE for SAP HANA, the name of the database module containing the CDS design-time entities, and the string *-hdi-container*, for example:

<XS\_UserName>-ctetig24[...]-<DB\_Module>-*hdi-container*

To bind to the HDI container, in the SAP HANA run-time *Catalog* tool, right-click *Catalog* in the catalog list, and in the *Search HDI Containers* dialog, locate the container to which you want to bind, and choose *Bind*.

## Related Information

[Set up an HDI Container \[page 176\]](#)

[Create the Data Persistence Artifacts in XS Advanced \[page 200\]](#)

## 5.3.4.1 CDS User-Defined Data Types in XS Advanced

User-defined data types reference existing structured types (for example, user-defined) or the individual types (for example, field, type, or context) used in another data-type definition.

You can use the `type` keyword to define a new data type in CDS-compliant DDL syntax. You can define the data type in the following ways:

- Using allowed structured types (for example, user-defined)
- Referencing another data type

In the following example, the element definition `field2 : MyType1;` specifies a new element `field2` that is based on the specification in the user-defined data type `MyType1`.

### i Note

If you are using a CDS document to define a single CDS-compliant user-defined data type, the name of the CDS document must match the name of the top-level data type defined in the CDS document. In the following example, you must save the data-type definition “`MyType1`” in the CDS document `MyType1.hdbcds`.

```
namespace com.acme.myapp1;
type MyType1 {
    field1 : Integer;
    field2 : String(40);
    field3 : Decimal(22,11);
    field4 : Binary(11);
};
```

In the following example, you must save the data-type definition “`MyType2`” in the CDS document `MyType2.hdbcds`; the document contains a using directive pointing to the data-type “`MyType1`” defined in CDS document `MyType1.hdbdd`.

```
namespace com.acme.myapp1;
using com.acme.myapp1::MyType1;
type MyType2 {
    field1 : String(50);
    field2 : MyType1;
};
```

In the following example, you must save the data-type definition “`MyType3`” in the CDS document `MyType3.hdbcds`; the document contains a using directive pointing to the data-type “`MyType2`” defined in CDS document `MyType2.hdbdd`.

```
namespace com.acme.myapp1;
using com.acme.myapp1::MyType2;
type MyType3 {
    field1 : UTCTimestamp;
    field2 : MyType2;
};
```

The following code example shows how to use the `type of` keyword to define an element using the definition specified in another user-defined data-type field. For example, `field4 : type of field3;` indicates that, like `field3`, `field4` is a `LocalDate` data type.

```
namespace com.acme.myapp1;
```

```

using com.acme.myapp1::MyType1;
entity MyEntity1 {
    key id : Integer;
    field1 : MyType3;
    field2 : String(24);
    field3 : LocalDate;
    field4 : type of field3;
    field5 : type of MyType1.field2;
    field6 : type of InnerCtx.CtxType.b; // context reference
};

```

You can use the `type of` keyword in the following ways:

- Define a new element (`field4`) using the definition specified in another user-defined element `field3`:  
`field4 : type of field3;`
- Define a new element `field5` using the definition specified in a **field** (`field2`) that belongs to another user-defined data type (`MyType1`):  
`field5 : type of MyType1.field2;`
- Define a new element (`field6`) using an existing field (`b`) that belongs to a data type (`CtxType`) in another **context** (`InnerCtx`):  
`field6 : type of InnerCtx.CtxType.b;`

The following code example shows you how to define nested contexts (`MyContext.InnerCtx`) and refer to data types defined by a user in the specified context.

```

namespace com.acme.myapp1;
context MyContext {
    // Nested contexts
    context InnerCtx {
        Entity MyEntity {
            ...
        };
        Type CtxType {
            a : Integer;
            b : String(59);
        };
    };
    type MyType1 {
        field1 : Integer;
        field2 : String(40);
        field3 : Decimal(22,11);
        field4 : Binary(11);
    };
    type MyType2 {
        field1 : String(50);
        field2 : MyType1;
    };
    type MyType3 {
        field1 : UTCTimestamp;
        field2 : MyType2;
    };
    entity MyEntity1 {
        key id : Integer;
        field1 : MyType3 not null;
        field2 : String(24);
        field3 : LocalDate;
        field4 : type of field3;
        field5 : type of MyType1.field2;
        field6 : type of InnerCtx.CtxType.b; // refers to nested context
        field7 : InnerCtx.CtxType; // more context references
    } technical configuration {
        unique index IndexA on (field1) asc;
    };
}

```

## Restrictions

CDS name resolution does not distinguish between CDS elements and CDS types. If you define a CDS element based on a CDS data type that has the same name as the new CDS element, CDS displays an error message and the deployment of the objects defined in the CDS document fails.

### ⚠ Caution

In an CDS document, you cannot define a CDS element using a CDS type of the same name; you must specify the **context** where the target type is defined, for example, `MyContext.doobidoo`.

The following example defines an association between a CDS element and a CDS data type both of which are named `doobidoo`. The result is an error when resolving the names in the CDS document; CDS expects a type named `doobidoo` but finds an CDS entity element with the same name that is **not** a type.

```
context MyContext2 {
    type doobidoo : Integer;
    entity MyEntity {
        key id : Integer;
        doobidoo : doobidoo; // error: type expected; doobidoo is not a type
    };
}
```

The following example works, since the explicit reference to the context where the type definition is located (`MyContext.doobidoo`) enables CDS to resolve the definition target.

```
context MyContext {
    type doobidoo : Integer;
    entity MyEntity {
        key id : Integer;
        doobidoo : MyContext.doobidoo; // OK
    };
}
```

### ℹ Note

To prevent name clashes between artifacts that **are** types and those that **have** a type assigned to them, make sure you keep to strict naming conventions. For example, use an **uppercase** first letter for `MyEntity`, `MyView` and `MyType`; use a lowercase first letter for elements `myElement`.

## Related Information

[Create a CDS User-Defined Structure in XS Advanced \[page 264\]](#)

## 5.3.4.2 CDS Structured Type Definition in XS Advanced

A structured type is a data type comprising a list of attributes, each of which has its own data type. The attributes of the structured type can be defined manually in the structured type itself and reused either by another structured type or an entity.

SAP HANA Extended Application Services advanced model (XS advanced) enables you to create a database structured type as a design-time file. All design-time files including your structured-type definition can be transported to other SAP HANA systems and deployed there to create the same catalog objects as those created in the original SAP HANA system. You can define the structured type using CDS-compliant DDL.

When a CDS document is deployed as part of a database module, the deployment process generates a corresponding catalog object for each of the artifacts defined in the CDS document; the location in the catalog is determined by the type of object generated. For example, the corresponding table type for a CDS type definition is generated in the following catalog location:

► <SID> > Catalog > <SCHEMA\_NAME> > Procedures > Table Types ▶

### Structured User-Defined Types

In a structured user-defined type, you can define original types (`aNumber` in the following example) or reference existing types defined elsewhere in the same type definition or another, separate type definition (`MyString80`). If you define multiple types in a single CDS document, for example, in a parent context, each structure-type definition must be separated by a semi-colon (;`).`

The type `MyString80` is defined in the following CDS document:

```
namespace Package1.Package2;
type MyString80: String(80);
```

A `using` directive is required to resolve the reference to the data type specified in `otherText : MyString80;`, as illustrated in the following example:

```
namespace Package1.Package2;
using Package1.Package2::MyString80; //contains definition of MyString80
type MyStruct
{
  aNumber : Integer;
  someText : String(80);
  otherText : MyString80; // defined in a separate type
};
```

#### i Note

If you are using a CDS document to specify a single CDS-compliant data type, the name of the CDS document (`MyStruct.hdbcds`) must match the name of the top-level data type defined in the CDS document, for example, with the `type` keyword.

## Nested Structured Types

Since user-defined types can make use of other user-defined types, you can build nested structured types, as illustrated in the following example:

```
namespace Package1.Package2;
using Package1.Package2::MyString80;
using Package1.Package2::MyStruct;
@Schema: 'MYSCHHEMA'
context NestedStructs {
    type MyNestedStruct
    {
        name : MyString80;
        nested : MyStruct; // defined in a separate type
    };
    type MyDeepNestedStruct
    {
        text : LargeString;
        nested : MyNestedStruct;
    };
    type MyOtherInt      : type of MyStruct.aNumber; // => Integer
    type MyOtherStruct : type of MyDeepNestedStruct.nested.nested; // => MyStruct
};
```

The example above shows how you can use the `type of` keyword to define a type based on an existing type that is already defined in another user-defined structured type.

## Generated Table Types

For each structured type, a SAP HANA table type is generated, whose name is built by concatenating the following elements of the CDS document containing the structured-type definition and separating the elements by a dot delimiter (.):

- the name space (for example, Pack1.Pack2)
- the names of all artifacts that enclose the type (for example, MyModel)
- the name of the type itself (for example, MyNestedStruct)

```
create type "Pack1.Pack2::MyModel.MyNestedStruct" as table(
    name          nvarchar(80),
    nested.aNumber integer,
    nested.someText nvarchar(80),
    nested.otherText nvarchar(80)
);
```

The columns of the table type are built by flattening the elements of the type. Elements with structured types are mapped to one column per nested element, with the column names built by concatenating the element names and separating the names by dots ".".

Table types are only generated for direct structure definitions; in the following example, this would include: MyStruct, MyNestedStruct, and MyDeepNestedStruct. No table types are generated for **derived** types that are based on structured types; in the following example, the derived types include: MyS, MyOtherInt, MyOtherStruct.

## Example

```
namespace Pack1."pack-age2";
context MyModel
{
    type MyInteger : Integer;
    type MyString80 : String(80);
    type MyDecimal : Decimal(10,2);
    type MyStruct
    {
        aNumber : Integer;
        someText : String(80);
        otherText : MyString80; // defined in example above
    };
    type MyS : MyStruct;
    type MyOtherInt : type of MyStruct.aNumber;
    type MyOtherStruct : type of MyDeepNestedStruct.nested.nested;
    type MyNestedStruct
    {
        name : MyString80;
        nested : MyS;
    };
    type MyDeepNestedStruct
    {
        text : LargeString;
        nested : MyNestedStruct;
    };
};
```

## Related Information

[Create a CDS User-Defined Structure in XS Advanced \[page 264\]](#)

[CDS User-Defined Data Types in XS Advanced \[page 267\]](#)

### 5.3.4.3 CDS Structured Types in XS Advanced

A structured type is a data type comprising a list of attributes, each of which has its own data type. The attributes of the structured type can be defined manually in the structured type itself and reused either by another structured type or an entity.

## Example

```
namespace examples;
context StructuredTypes {
    type MyOtherInt : type of MyStruct.aNumber; // => Integer
    type MyOtherStruct : type of MyDeepNestedStruct.nested.nested; // =>
MyStruct
    type EmptyStruct { };
    type MyStruct
    {
        aNumber : Integer;
        aText : String(80);
        anotherText : MyString80; // defined in a separate type
    };
};
```

```

};

entity E {
    a : Integer;
    s : EmptyStruct;
};

type MyString80 : String(80);
type MyS : MyStruct;
type MyNestedStruct
{
    name : MyString80;
    nested : MyS;
};
type MyDeepNestedStruct
{
    text : LargeString;
    nested : MyNestedStruct;
};

```

## type

In a structured user-defined type, you can define original types (`aNumber` in the following example) or reference existing types defined elsewhere in the same type definition or another, separate type definition, for example, `MyString80` in the following code snippet. If you define multiple types in a single CDS document, each structure definition must be separated by a semi-colon (;).

```

type MyStruct
{
    aNumber      : Integer;
    aText        : String(80);
    anotherText  : MyString80; // defined in a separate type
};

```

You can define structured types that do not contain any elements, for example, using the keywords `type EmptyStruct { };`. In the example, below the generated table for entity "E" contains only one column: "a".

### Restriction

It is not possible to generate an SAP HANA table type for an empty structured type.

```

type EmptyStruct { };
entity E {
    a : Integer;
    s : EmptyStruct;
};

```

## type of

You can define a type based on an existing type that is already defined in another user-defined structured type, for example, by using the `type of` keyword, as illustrated in the following example:

```
Context StructuredTypes
{
    type MyOtherInt      : type of MyStruct.aNumber;           // => Integer
    type MyOtherStruct   : type of MyDeepNestedStruct.nested.nested; // =>
MyStruct
};
```

## Related Information

[Create a CDS User-Defined Structure in XS Advanced \[page 264\]](#)

[CDS User-Defined Data Types in XS Advanced \[page 267\]](#)

[CDS Structured Type Definition in XS Advanced \[page 270\]](#)

### 5.3.4.4 CDS Primitive Data Types

In the Data Definition Language (DDL), primitive (or core) data types are the basic building blocks that you use to define *entities* or *structure types* with DDL.

When you are specifying a design-time table (entity) or a view definition using the CDS syntax, you use data types such as *String*, *Binary*, or *Integer* to specify the type of content in the entity columns. CDS supports the use of the following primitive data types:

- [DDL data types \[page 274\]](#)
- [Native SAP HANA data types \[page 276\]](#)

The following table lists all currently supported simple DDL primitive data types. Additional information provided in this table includes the SQL syntax required as well as the equivalent SQL and EDM names for the listed types.

Table 31: Supported SAP HANA DDL Primitive Types

Name	Description	SQL Literal Syntax	SQL Name	EDM Name
String (n)	Variable-length Unicode string with a specified maximum length of n=1-1333 characters (5000 for SAP HANA specific objects). Default = maximum length. String length (n) is mandatory.	'text with "quote"'	NVARCHAR	String
LargeString	Variable length string of up to 2 GB (no comparison)	'text with "quote"'	NCLOB	String

Name	Description	SQL Literal Syntax	SQL Name	EDM Name
Binary(n)	Variable length byte string with user-defined length limit of up to 4000 bytes. Binary length (n) is mandatory.	x'01Cafe', X'01Cafe'	VARBINARY	Binary
LargeBinary	Variable length byte string of up to 2 GB (no comparison)	x'01Cafe', X'01Cafe'	BLOB	Binary
Integer	Respective container's standard signed integer. Signed 32 bit integers in 2's complement, -2**31 .. 2**31-1. Default=NULL	13, -1234567	INTEGER	Int64
Integer64	Signed 64-bit integer with a value range of -2^63 to 2^63-1. Default=NULL.	13, -1234567	BIGINT	Int64
Decimal( p, s )	Decimal number with fixed precision (p) in range of 1 to 34 and fixed scale (s) in range of 0 to p. Values for precision and scale are mandatory.	12.345, -9.876	DECIMAL( p, s )	Decimal
DecimalFloat	Decimal floating-point number (IEEE 754-2008) with 34 mantissa digits; range is roughly ±1e-6143 through ±9.99e+6144	12.345, -9.876	DECIMAL	Decimal
BinaryFloat	Binary floating-point number (IEEE 754), 8 bytes (roughly 16 decimal digits precision); range is roughly ±2.2207e-308 through ±1.7977e +308	1.2, -3.4, 5.6e+7	DOUBLE	Double
LocalDate	Local date with values ranging from 0001-01-01 through 9999-12-31	date'1234-12-31'	DATE	DateTimeOffset Combines date and time; with time zone must be converted to offset
LocalTime	Time values (with seconds precision) and values ranging from 00:00:00 through 24:00:00	time'23:59:59', time'12:15'	TIME	Time For duration/ period of time (==xsd:duration). Use DateTimeOffset if there is a date, too.

Name	Description	SQL Literal Syntax	SQL Name	EDM Name
UTCDateTime	UTC date and time (with seconds precision) and values ranging from 0001-01-01 00:00:00 through 9999-12-31 23:59:59	timestamp'2011-12-31 23:59:59'	SECONDDATE	Date DateTimeOffset Values ending with "Z" for UTC. Values before 1753-01-01T00:00:00 are not supported; transmitted as NULL.
UTCTimestamp	UTC date and time (with a precision of 0.1 microseconds) and values ranging from 0001-01-01 00:00:00 through 9999-12-31 23:59:59.9999999, and a special initial value	timestamp'2011-12-31 23:59:59.7654321'	TIMESTAMP	Date DateTimeOffset With Precision = "7"
Boolean	Represents the concept of binary-valued logic	true, false, unknown (null)	BOOLEAN	Boolean

The following table lists all the **native** SAP HANA primitive data types that CDS supports. The information provided in this table also includes the SQL syntax required (where appropriate) as well as the equivalent SQL and EDM names for the listed types.

### i Note

\* In CDS, the name of SAP HANA data types are prefixed with the word "hana", for example, `hana.ALPHANUM`, or `hana.SMALLINT`, or `hana.TINYINT`.

Table 32: Supported Native SAP HANA Data Types

Data Type *	Description	SQL Literal Syntax	SQL Name	EDM Name
ALPHANUM	Variable-length character string with special comparison	-	ALPHANUMERIC	-
SMALLINT	Signed 16-bit integer	-32768, 32767	SMALLINT	Int16
TINYINT	Unsigned 8-bit integer	0, 255	TINYINT	Byte
REAL	32-bit binary floating-point number	-	REAL	Single
SMALLDECIMAL	64-bit decimal floating-point number	-	SMALLDECIMAL	Decimal
VARCHAR	Variable-length ASCII character string with user-definable length limit n	-	VARCHAR	String

Data Type *	Description	SQL Literal Syntax	SQL Name	EDM Name
CLOB	Large variable-length ASCII character string, no comparison	-	CLOB	String
BINARY	Byte string of fixed length n	-	BINARY	Blob
ST_POINT	0-dimensional geometry representing a single location	-	-	-
ST_GEOMETRY	Maximal supertype of the geometry type hierarchy; includes ST_POINT	-	-	-

The following example shows the **native** SAP HANA data types that CDS supports; the code example also illustrates the mandatory syntax.

### i Note

Support for the geo-spatial types ST\_POINT and ST\_GEOMETRY is limited: these types can only be used for the definition of elements in types and entities. It is not possible to define a CDS view that selects an element based on a geo-spatial type from a CDS entity.

```
@nokey
entity SomeTypes {
    a : hana.ALPHANUM(10);
    b : hana.SMALLINT;
    c : hana.TINYINT;
    d : hana.SMALLDECIMAL;
    e : hana.REAL;
    h : hana.VARCHAR(10);
    i : hana.CLOB;
    j : hana.BINARY(10);
    k : hana.ST_POINT;
    l : hana.ST_GEOMETRY;
};
```

## Related Information

[Create a CDS User-Defined Structure in XS Advanced \[page 264\]](#)

## 5.3.5 Create a CDS Association in XS Advanced

You create associations in a CDS entity definition, which is a design-time file in SAP HANA.

### Prerequisites

To complete this task successfully, note the following prerequisites:

- You must have access to an SAP HANA system.
- You must have access to the SAP Web IDE for SAP HANA

#### **i** Note

The permissions defined in the XS advanced role collection `XS_Authorization_User` must be assigned to the user who wants to access the tools included in the SAP Web IDE for SAP HANA.

- You must have already a development workspace and a multi-target application (MTA) project.
- You must already have created a database module for your MTA project.
- You must already have set up an HDI container for the CDS artifacts

#### **i** Note

A container setup file (`.hdiconfig`) is required to define which plug-ins to use to create the corresponding catalog objects from the design-time artifacts when the multi-target application (or just the database module) is deployed.

### Context

Associations define relationships between entities (tables). SAP HANA Extended Application Services for XS advanced (XS advanced) enables you to use the CDS syntax to create associations between entities. The association is defined in the entity definition itself. To create an association between CDS entities, perform the following steps:

### Procedure

1. Start the SAP HANA Web IDE for SAP HANA.

The SAP Web IDE for SAP HANA is available at the following URL:

`https://<HANA_HOST>:53075/`

## ➔ Tip

To display the URL for the SAP Web IDE for SAP HANA, open a command shell, log on to the XS advanced run time, and run the following command:

```
xs app webide --urls
```

2. Open the application project to which you want to add your CDS association.
3. Create the CDS entity-definition file that will contain the associations you define.

Browse to the folder in the database module in your application's project workspace where you want to create the new CDS entity-definition file, for example, <MyApp1>/HDB/src, and perform the following steps:

- a. Right-click the folder where you want to save the CDS entity-definition file and choose ➔ *New* ➔ *CDS Artifact* in the context-sensitive pop-up menu.
- b. Enter the name of the entity-definition file in the *File Name* box, for example, MyEntity.

## ➔ Tip

If you use the available setup Wizards to create your design-time artifacts, the correct file extensions is added automatically. The file extension is used to determine which plug-in to use to create the corresponding run-time object during deployment. CDS artifacts have the file extension .hdbc, for example, MyEntity.hdbc.

- c. Choose *Finish* to save the new CDS entity-definition file in the database module of your local project workspace.

4. Define the underlying CDS entities and structured types.

If the new entity-definition file is not automatically displayed by the file-creation wizard, double-click the entity-definition file you created in the previous step, for example, MyEntity.hdbc, and add the entity-definition code to the file:

## ℹ Note

The following code example is provided for illustration purposes only.

```
econtext MyEntity1 {
    type StreetAddress {
        name : String(80);
        number : Integer;
    };
    type CountryAddress {
        name : String(80);
        code : String(3);
    };
    entity Address {
        key id : Integer;
        street : StreetAddress;
        zipCode : Integer;
        city : String(80);
        country : CountryAddress;
        type : String(10); // home, office
    };
    entity Person
    {
        key id : Integer;
        addressId : Integer;
```

```
};  
};
```

5. Define a one-to-**one** association between CDS entities.

In the same entity-definition file you edited in the previous step, for example, `MyEntity.hdbc`, add the code for the one-to-one association between the entity `Person` and the entity `Address`, as illustrated below:

 **Note**

This example does not specify cardinality or foreign keys, so the cardinality is set to the default 0..1, and the target entity's primary key (the element `id`) is used as foreign key.

```
entity Person  
{  
    key id : Integer;  
    address1 : Association to Address;  
    addressId : Integer;  
};
```

6. Define an unmanaged association with cardinality one-to-**many** between CDS entities.

In the same entity-definition file you edited in the previous step, for example, `MyEntity.hdbc`, add the code for the one-to-many association between the entity `Address` and the entity `Person`. The code should look something like the bold text illustrated in the following example:

```
entity Address {  
    key id : Integer;  
    street : StreetAddress;  
    zipCode : Integer;  
    city : String(80);  
    country : CountryAddress;  
    type : String(10); // home, office  
    inhabitants : Association[*] to Person on inhabitants.addressId = id;  
};
```

7. Save the CDS document.

Saving the definition persists the file in your local workspace; it does not create any objects in the database catalog.

8. Activate the changes in the catalog.

To activate the new entity definition and generate a corresponding table in the catalog, use the *Build* feature.

- Right-click the new database module in your application project.
- In the context-sensitive pop-up menu, choose  **Build**.

 **Tip**

You can follow progress of the build in the console at the bottom of the CDS editor.

## Related Information

[Set up an HDI Container \[page 176\]](#)

[Create a CDS Document \(XS Advanced\) \[page 208\]](#)

### 5.3.5.1 CDS Associations

Associations define relationships between entities.

Associations are specified by adding an element to a source entity with an association **type** that points to a target entity, complemented by optional information defining cardinality and which keys to use.

**i** Note

CDS supports both managed and unmanaged associations.

SAP HANA Extended Application Services (SAP HANA XS) enables you to use associations in CDS entities or CDS views. The syntax for **simple** associations in a CDS document is illustrated in the following example:

```
namespace samples;
@Schema: 'MYSHEMA'           // XS classic *only*
context SimpleAssociations {
    type StreetAddress {
        name : String(80);
        number : Integer;
    };
    type CountryAddress {
        name : String(80);
        code : String(3);
    };
    entity Address {
        key id : Integer;
        street : StreetAddress;
        zipCode : Integer;
        city : String(80);
        country : CountryAddress;
        type : String(10); // home, office
    };
    entity Person
    {
        key id : Integer;
        // address1,2,3 are to-one associations
        address1 : Association to Address;
        address2 : Association to Address { id };
        address3 : Association[1] to Address { zipCode, street, country };
        // address4,5,6 are to-many associations
        address4 : Association[0..*] to Address { zipCode };
        address5 : Association[*] to Address { street.name };
        address6 : Association[*] to Address { street.name AS streetName,
                                                country.name AS countryName };
    };
}
```

## Cardinality in Associations

When using an association to define a relationship between entities in a CDS document, you use the **cardinality** to specify the type of relation, for example, one-to-one (to-one) or one-to-many (to-n); the relationship is with respect to both the source and the target of the association.

The target cardinality is stated in the form of [ min .. max ], where max=\* denotes infinity. If no cardinality is specified, the default cardinality setting [ 0..1 ] is assumed. It is possible to specify the maximum cardinality of the source of the association in the form [ maxs, min .. max], too, where maxs = \* denotes infinity.

### Tip

The information concerning the maximum cardinality is only used as a hint for optimizing the execution of the resulting JOIN.

The following examples illustrate how to express cardinality in an association definition:

```
namespace samples;
@Schema: 'MYSCHHEMA'           // XS classic *only*
context AssociationCardinality {
    entity Associations {
        // To-one associations
        assoc1 : Association[0..1]      to target; // has no or one target instance
        assoc2 : Association           to target; // as assoc1, uses the default
[min is 0]
        assoc3 : Association[1]        to target; // as assoc1; the default for
instance
        assoc4 : Association[1..1]      to target; // association has one target
        // To-many associations
        assoc5 : Association[0..*]     to target{id1};
        assoc6 : Association[]         to target{id1}; // as assoc4, [] is short
[for 0..*]
        assoc7 : Association[2..7]     to target{id1}; // any numbers are
possible; user provides
        assoc8 : Association[1, 0..*]   to target{id1}; // additional info. about
source cardinality
    };
    // Required to make the example above work
    entity target {
        key id1 : Integer;
        key id2 : Integer;
    };
}
```

## Target Entities in Associations

You use the `to` keyword in a CDS view definition to specify the target entity in an association, for example, the name of an entity defined in a CDS document. A qualified entity name is expected that refers to an existing entity. A target entity specification is mandatory; a default value is **not** assumed if no target entity is specified in an association relationship.

The entity `Address` specified as the target entity of an association could be expressed in any of the ways illustrated the following examples:

```
address1 : Association to Address;
address2 : Association to Address { id };
address3 : Association[1] to Address { zipCode, street, country };
```

## Filter Conditions and Prefix Notation

When following an association (for example, in a view), it is now possible to apply a filter condition; the filter is merged into the `ON`-condition of the resulting `JOIN`. The following example shows how to get a list of customers and then filter the list according to the sales orders that are currently “open” for each customer. In the example, the infix filter is inserted after the association orders to get only those orders that satisfy the condition `[status='open']`.

### Sample Code

```
view C1 as select from Customer {
    name,
    orders[status='open'].id as orderId
};
```

The association `orders` is defined in the entity definition illustrated in the following code example:

### Sample Code

```
entity Customer {
    key id : Integer;
    orders : Association[*] to SalesOrder on orders.cust_id = id;
    name : String(80);
};

entity SalesOrder {
    key id : Integer;
    cust_id : Integer;
    customer: Association[1] to Customer on customer.id = cust_id;
    items : Association[*] to Item on items.order_id = id;
    status: String(20);
    date : LocalDate;
};

entity Item {
    key id : Integer;
    order_id : Integer;
    salesOrder : Association[1] to SalesOrder on salesOrder.id = order_id;
    descr : String(100);
    price : Decimal(8,2);
};
```

### Tip

For more information about filter conditions and prefixes in CDS views, see *CDS Views* and *CDS View Syntax Options*.

## Foreign Keys in Associations

For **managed** associations, the relationship between source and target entity is defined by specifying a set of elements of the target entity that are used as a foreign key. If no foreign keys are specified explicitly, the elements of the target entity's designated primary key are used. Elements of the target entity that reside inside substructures can be addressed via the respective path. If the chosen elements do not form a unique key of the target entity, the association has cardinality to-many. The following examples show how to express foreign keys in an association.

```
namespace samples;
using samples::SimpleAssociations.StreetAddress;
using samples::SimpleAssociations.CountryAddress;
using samples::SimpleAssociations.Address;
@Schema: 'MYSCHHEMA'           // XS classic *only*
context ForeignKeys {
    entity Person {
        key id : Integer;
        // address1,2,3 are to-one associations
        address1 : Association to Address;
        address2 : Association to Address { id };
        address3 : Association[1] to Address { zipCode, street, country };
        // address4,5,6 are to-many associations
        address4 : Association[0..*] to Address { zipCode };
        address5 : Association[*] to Address { street.name };
        address6 : Association[*] to Address { street.name AS streetName,
                                                country.name AS countryName };
    };
    entity Header {
        key id : Integer;
        toItems : Association[*] to Item on toItems.head.id = id;
    };
    entity Item {
        key id : Integer;
        head : Association[1] to Header { id };
        // <...>
    };
}
```

- **address1**  
No foreign keys are specified: the target entity's primary key (the element `id`) is used as foreign key.
- **address2**  
Explicitly specifies the foreign key (the element `id`); this definition is similar to `address1`.
- **address3**  
The foreign key elements to be used for the association are explicitly specified, namely: `zipcode` and the structured elements `street` and `country`.
- **address4**  
Uses only `zipcode` as the foreign key. Since `zipcode` is not a unique key for entity `Address`, this association has cardinality "to-many".
- **address5**  
Uses the subelement `name` of the structured element `street` as a foreign key. This is not a unique key and, as a result, `address4` has cardinality "to-many".
- **address6**  
Uses the subelement `name` of both the structured elements `street` and `country` as foreign key fields. The names of the foreign key fields must be unique, so an alias is required here. The foreign key is not unique, so `address6` is a "to-many" association.

You can use foreign keys of managed associations in the definition of other associations. In the following example, the appearance of association head in the ON condition is allowed; the compiler recognizes that the field head.id is actually part of the entity Item and, as a result, can be obtained without following the association head.

### Sample Code

```
entity Header {
    key id : Integer;
    toItems : Association[*] to Item on toItems.head.id = id;
};

entity Item {
    key id : Integer;
    head : Association[1] to Header { id };
    ...
};
```

## Restrictions

CDS name resolution does not distinguish between CDS associations and CDS entities. If you define a CDS association with a CDS entity that has the same name as the new CDS association, CDS displays an error message and the activation of the CDS document fails.

### Caution

In an CDS document, to define an association with a CDS entity of the same name, you must specify the **context** where the target entity is defined, for example, MyContext.Address3.

The following code shows some examples of associations with a CDS entity that has the same (or a similar) name. Case sensitivity ("a", "A") is important; in CDS documents, address is not the same as Address. In the case of Address2, where the association name and the entity name are identical, the result is an error; when resolving the element names, CDS expects an entity named Address2 but finds a CDS association with the same name instead. MyContext.Address3 is allowed, since the target entity can be resolved due to the absolute path to its location in the CDS document.

```
context MyContext {
    entity Address {...}
    entity Address1 {...}
    entity Address2 {...}
    entity Address3 {...}
    entity Person {
        key id : Integer;
        address : Association to Address;           // OK: "address" ≠ "Address"
        address1 : Association to Address1;          // OK: "address1" ≠ "Address1"
        Address2 : Association to Address2;          // Error: association name =
    entity name
        Address3 : Association to MyContext.Address3; //OK: full path to Address3
    };
};
```

 Example**Complex (One-to-Many) Association**

The following example shows a more complex association (to-many) between the entity "Header" and the entity "Item".

```
namespace samples;
@Schema: 'MYSCHHEMA'          // XS classic *only*
context ComplexAssociation {
    Entity Header {
        key PurchaseOrderId: BusinessKey;
        Items: Association [0..*] to Item on
        Items.PurchaseOrderId=PurchaseOrderId;
        "History": HistoryT;
        NoteId: BusinessKey null;
        PartnerId: BusinessKey;
        Currency: CurrencyT;
        GrossAmount: AmountT;
        NetAmount: AmountT;
        TaxAmount: AmountT;
        LifecycleStatus: StatusT;
        ApprovalStatus: StatusT;
        ConfirmStatus: StatusT;
        OrderingStatus: StatusT;
        InvoicingStatus: StatusT;
    } technical configuration {
        column store;
    };
    Entity Item {
        key PurchaseOrderId: BusinessKey;
        key PurchaseOrderItem: BusinessKey;
        ToHeader: Association [1] to Header on
        ToHeader.PurchaseOrderId=PurchaseOrderId;
        ProductId: BusinessKey;
        NoteId: BusinessKey null;
        Currency: CurrencyT;
        GrossAmount: AmountT;
        NetAmount: AmountT;
        TaxAmount: AmountT;
        Quantity: QuantityT;
        QuantityUnit: UnitT;
        DeliveryDate: SDate;
    } technical configuration {
        column store;
    };
    define view POView as SELECT from Header {
        Items.PurchaseOrderId as poId,
        Items.PurchaseOrderItem as poItem,
        PartnerId,
        Items.ProductId
    };
    // Missing types from the example above
    type BusinessKey: String(50);
    type HistoryT: LargeString;
    type CurrencyT: String(3);
    type AmountT: Decimal(15, 2);
    type StatusT: String(1);
    type QuantityT: Integer;
    type UnitT: String(5);
    type SDate: LocalDate;
};
```

## Related Information

[Create a CDS Association in XS Advanced \[page 278\]](#)

### 5.3.5.2 CDS Association Syntax Options

Associations define relationships between entities.

#### Example

##### Managed Associations

```
Association [ <cardinality> ] to <targetEntity> [ <forwardLink> ]
```

#### Example

##### Unmanaged Associations

```
Association [ <cardinality> ] to <targetEntity> <unmanagedJoin>
```

## Overview

Associations are specified by adding an element to a source entity with an association **type** that points to a target entity, complemented by optional information defining cardinality and which keys to use.

#### i Note

CDS supports both managed and unmanaged associations.

SAP HANA Extended Application Services (SAP HANA XS) enables you to use associations in the definition of a CDS entity or a CDS view. When defining an association, bear in mind the following points:

- [`<Cardinality>` \[page 288\]](#)  
The relationship between the source and target in the association, for example, one-to-one, one-to-many, many-to-one
- [`<targetEntity>` \[page 289\]](#)  
The target entity for the association
- [`<forwardLink>` \[page 290\]](#)  
The foreign keys to use in a managed association, for example, element names in the target entity
- [`<unmanagedJoin>` \[page 291\]](#)  
**Unmanaged** associations only; the `ON` condition specifies the elements of the source and target elements and entities to use in the association

## Association Cardinality

When using an association to define a relationship between entities in a CDS view; you use the **cardinality** to specify the type of relation, for example:

- one-to-one (to-one)
- one-to-many (to-n)

The relationship is with respect to both the source and the target of the association. The following code example illustrates the syntax required to define the cardinality of an association in a CDS view:

```
[ [ ( maxs | * ) , ]           // source cardinality
  [ min .. ] ( max | * )       // target cardinality
]
```

In the most simple form, only the target cardinality is stated using the syntax `[ min .. max ]`, where `max=*` denotes infinity. Note that `[]` is short for `[ 0..* ]`. If no cardinality is specified, the default cardinality setting `[ 0..1 ]` is assumed. It is possible to specify the maximum cardinality of the source of the association in the form `[ maxs, min .. max ]`, where `maxs = *` denotes infinity.

The following examples illustrate how to express cardinality in an association definition:

```
namespace samples;
@Schema: 'MYSCHHEMA'           // XS classic *only*
context AssociationCardinality {
    entity Associations {
        // To-one associations
        assoc1 : Association[0..1]      to target;
        assoc2 : Association          to target;
        assoc3 : Association[1]        to target;
        assoc4 : Association[1..1]     to target; // association has one target
    }
    instance
        // To-many associations
        assoc5 : Association[0..*]    to target{id1};
        assoc6 : Association[]       to target{id1}; // as assoc4, [] is short
    for [0..*]
        assoc7 : Association[2..7]    to target{id1}; // any numbers are
    possible; user provides
        assoc8 : Association[1, 0..*] to target{id1}; // additional info. about
    source cardinality
    };
    // Required to make the example above work
    entity target {
        key id1 : Integer;
        key id2 : Integer;
    };
}
```

The following table describes the various cardinality expressions illustrated in the example above:

Table 33: Association Cardinality Syntax Examples

Association	Cardinality	Explanation
assoc1	[0..1]	The association has no or one target instance
assoc2		Like assoc1, this association has no or one target instance and uses the default [0..1]
assoc3	[1]	Like assoc1, this association has no or one target instance; the default for min is 0

Association	Cardinality	Explanation
assoc4	[1..1]	The association has one target instance
assoc5	[0..*]	The association has no, one, or multiple target instances
assoc6	[]	Like assoc4, [] is short for [0..*] (the association has no, one, or multiple target instances)
assoc7	[2..7]	Any numbers are possible; the user provides
assoc8	[1, 0..*]	The association has no, one, or multiple target instances and includes additional information about the source cardinality

When an infix filter effectively reduces the cardinality of a “to-N” association to “to-1”, this can be expressed explicitly in the filter, for example:

```
assoc[1: <cond> ]
```

Specifying the cardinality in the filter in this way enables you to use the association in the `WHERE` clause, where “to-N” associations are not normally allowed.

### Sample Code

```
namespace samples;
@Schema: 'MYSCHHEMA' // XS classic *only*
context CardinalityByInfixFilter {
    entity Person {
        key id : Integer;
        name : String(100);
        address : Association[*] to Address on address.personId = id;
    };
    entity Address {
        key id : Integer;
        personId : Integer;
        type : String(20); // home, business, vacation, ...
        street : String(100);
        city : String(100);
    };
    view V as select from Person {
        name
    } where address[1: type='home'].city = 'Accra';
};
```

## Association Target

You use the `to` keyword in a CDS view definition to specify the target entity in an association, for example, the name of an entity defined in a CDS document. A qualified entity name is expected that refers to an existing entity. A target entity specification is mandatory; a default value is **not** assumed if no target entity is specified in an association relationship.

```
Association[ <cardinality> ] to <targetEntity> [ <forwardLink> ]
```

The target entity `Address` specified as the target entity of an association could be expressed as illustrated the following examples:

```
address1 : Association to Address;
address2 : Association to Address { id };
address3 : Association[1] to Address { zipCode, street, country };
```

## Association Keys

In the relational model, associations are mapped to foreign-key relationships. For **managed** associations, the relation between source and target entity is defined by specifying a set of elements of the target entity that are used as a foreign key, as expressed in the `forwardLink` element of the following code example:

```
Association[ <cardinality> ] to <targetEntity> [ <forwardLink> ]
```

The `forwardLink` element of the association could be expressed as follows:

```
<forwardLink> = { <foreignKeys> }
<foreignKeys> = <targetKeyElement> [ AS <alias> ] [ , <foreignKeys> ]
<targetKeyElement> = <elementName> ( . <elementName> )*
```

If no foreign keys are specified explicitly, the elements of the target entity's designated primary key are used. Elements of the target entity that reside inside substructures can be addressed by means of the respective path. If the chosen elements do not form a unique key of the target entity, the association has cardinality to-many. The following examples show how to express foreign keys in an association.

```
entity Person
{
    key id : Integer;
    // address1,2,3 are to-one associations
    address1 : Association to Address;
    address2 : Association to Address { id };
    address3 : Association[1] to Address { zipCode, street, country };
    // address4,5,6 are to-many associations
    address4 : Association[0..*] to Address { zipCode };
    address5 : Association[*] to Address { street.name };
    address6 : Association[*] to Address { street.name AS streetName,
                                              country.name AS countryName };
}
```

Table 34: Association Syntax Options

Association	Keys	Explanation
address1		No foreign keys are specified: the target entity's primary key (the element <code>id</code> ) is used as foreign key.
address2	{ <code>id</code> }	Explicitly specifies the foreign key (the element <code>id</code> ); this definition is identical to <code>address1</code> .
address3	{ <code>zipCode</code> , <code>street</code> , <code>country</code> }	The foreign key elements to be used for the association are explicitly specified, namely: <code>zipcode</code> and the structured elements <code>street</code> and <code>country</code> .

Association	Keys	Explanation
address4	{ zipcode }	Uses only zipcode as the foreign key. Since zipcode is not a unique key for entity Address, this association has cardinality "to-many".
address5	{ street.name }	Uses the sub-element name of the structured element street as a foreign key. This is not a unique key and, as a result, address4 has cardinality "to-many".
address6	{ street.name AS streetName, country.name AS countryName }	Uses the sub-element name of both the structured elements street and country as foreign key fields. The names of the foreign key fields must be unique, so an alias is required here. The foreign key is not unique, so address6 is a "to-many" association.

You can now use foreign keys of managed associations in the definition of other associations. In the following example, the compiler recognizes that the field toCountry.cid is part of the foreign key of the association toLocation and, as a result, physically present in the entity Company.

### Sample Code

```
namespace samples;
@Schema: 'MYSCHHEMA'           // XS classic *only*
context AssociationKeys {
    entity Country {
        key c_id : String(3);
        // <...>
    };
    entity Region {
        key r_id : Integer;
        key toCountry : Association[1] to Country { c_id };
        // <...>
    };
    entity Company {
        key id : Integer;
        toLocation : Association[1] to Region { r_id, toCountry.c_id };
        // <...>
    };
}
```

## Unmanaged Associations

**Unmanaged** associations are based on existing elements of the source and target entity; no fields are generated. In the ON condition, only elements of the source or the target entity can be used; it is not possible to use other associations. The ON condition may contain any kind of expression - all expressions supported in views can also be used in the ON condition of an unmanaged association.

### i Note

The names in the ON condition are resolved in the scope of the source entity; elements of the target entity are accessed through the association itself .

In the following example, the association `inhabitants` relates the element `id` of the source entity `Room` with the element `officeId` in the target entity `Employee`. The target element `officeId` is accessed through the name of the association itself.

```
namespace samples;
@Schema: 'MYSCHHEMA' // XS classic *only*
context UnmanagedAssociations {
    entity Employee {
        key id : Integer;
        officeId : Integer;
        // <...>
    };
    entity Room {
        key id : Integer;
        inhabitants : Association[*] to Employee on inhabitants.officeId = id;
        // <...>
    };
    entity Thing {
        key id : Integer;
        parentId : Integer;
        parent : Association[1] to Thing on parent.id = parentId;
        children : Association[*] to Thing on children.parentId = id;
        // <...>
    };
}
```

The following example defines two related **unmanaged** associations:

- `parent`  
The unmanaged association `parent` uses a cardinality of `[1]` to create a relation between the element `parentId` and the target element `id`. The target element `id` is accessed through the name of the association itself.
- `children`  
The unmanaged association `children` creates a relation between the element `id` and the target element `parentId`. The target element `parentId` is accessed through the name of the association itself.

```
entity Thing {
    key id : Integer;
    parentId : Integer;
    parent : Association[1] to Thing on parent.id = parentId;
    children : Association[*] to Thing on children.parentId = id;
    ...
};
```

## Constants in Associations

The usage of constants is no longer restricted to annotation assignments and default values for entity elements. With SPS 11, you can use constants in the “ON”-condition of unmanaged associations, as illustrated in the following example:

### Sample Code

```
context MyContext {
    const MyIntConst : Integer = 7;
    const MyStringConst : String(10) = 'bright';
    const MyDecConst : Decimal(4,2) = 3.14;
    const MyDateTimeConst : UTCDateTime = '2015-09-30 14:33';
    entity MyEntity {
        key id : Integer;
        a : Integer;
        b : String(100);
```

```

c : Decimal(20,10);
d : UTCDateTime;
your : Association[1] to YourEntity on your.a - a < MyIntConst;
};
entity YourEntity {
    key id : Integer;
    a : Integer;
};
entity HerEntity {
    key id : Integer;
    t : String(20);
};
view MyView as select from MyEntity
    inner join HerEntity on locate (b, :MyStringConst) > 0
{
    a + :MyIntConst as x,
    b || ' is ' || :MyStringConst as y,
    c * sin(:MyDecConst) as z
} where d < :MyContext.MyDateTimeConst;
};

```

## Related Information

[Create a CDS Association in XS Advanced \[page 278\]](#)

[CDS Associations \[page 281\]](#)

## 5.3.6 Create a CDS View in XS Advanced

Define a design-time **view** using the Core Data Services (CDS) syntax.

## Prerequisites

To complete this task successfully, note the following prerequisites:

- You must have access to an SAP HANA system.
- You must have access to the SAP Web IDE for SAP HANA

### **i** Note

The permissions defined in the XS advanced role collection XS\_Authorization\_User must be assigned to the user who wants to access the tools included in the SAP Web IDE for SAP HANA.

- You must have already a development workspace and a multi-target application (MTA) project.
- You must already have created a database module for your MTA application project.
- You must already have set up an HDI container for the CDS artifacts

### **i** Note

A container setup file (.hdiconfig) is required to define which plug-ins to use to create the corresponding catalog objects from the design-time artifacts when the multi-target application (or just the database module) is deployed.

- You must have access to the SAP HANA XS advanced run-time tools that enable you to view the contents of the catalog.

### **i** Note

The permissions defined in the XS advanced role collection XS\_Authorization\_USER must be assigned to the user who wants to access the SAP HANA run-time tools.

## Context

A view is a virtual table based on the dynamic results returned in response to an SQL statement. SAP HANA Extended Application Services for XS advanced model (XS advanced) enables you to use CDS syntax to create a database view as a design-time file. You can use this design-time view definition to generate the corresponding catalog object when you deploy the application that contains the view-definition artifact (or just the application's database module).

### **i** Note

The code examples provided are for illustration purposes only.

## Procedure

1. Start the SAP HANA Web IDE for SAP HANA.

The SAP Web IDE for SAP HANA is available at the following URL:

[https://<HANA\\_HOST>:53075](https://<HANA_HOST>:53075)

### ➔ Tip

To display the URL for the SAP Web IDE for SAP HANA, open a command shell, log on to the XS advanced run time, and run the following command:

```
xs app webide --urls
```

2. Open the application project to which you want to add your CDS entity.
3. Create the CDS document that will contain the view-definition.

Browse to the folder in the database module in your application's project workspace, for example, <MyApp1>/HDB/src where you want to create the new CDS document with the view-definition file and perform the following steps:

- a. Right-click the folder where you want to save the CDS entity-definition file and choose ► **New** ► **CDS Artifact** in the context-sensitive pop-up menu.
- b. Enter the name of the view-definition file in the **File Name** box, for example, `MyViewContext`.

**→ Tip**

If you use the available setup Wizards to create your design-time artifacts, the correct file extensions is added automatically. The file extension is used to determine which plug-in to use to create the corresponding run-time object during deployment. CDS artifacts have the file extension `.hdbc`, for example, `MyViewContext.hdbc`.

- c. Choose **Finish** to save the new CDS view-definition file in the database module of your application's local project workspace.
4. Define the underlying CDS entities and structured types for the SQL view.

If the new CDS document is not automatically displayed by the file-creation wizard, double-click the CDS file you created in the previous step, for example, `MyViewContext.hdbc`, and add the code that defines the underlying table entities and structured types:

```
context MyViewContext {
    type StreetAddress {
        name : String(80);
        number : Integer;
    };
    type CountryAddress {
        name : String(80);
        code : String(3);
    };
    entity Address {
        key id : Integer;
        street : StreetAddress;
        zipCode : Integer;
        city : String(80);
        country : CountryAddress;
        type : String(10); // home, office
    } technical configuration {
        column store;
    };
}
```

5. Define an SQL view as a projection of a CDS entity.

In the same CDS document you edited in the previous step, for example, `MyViewContext.hdbc`, add the code for the view `AddressView` to the end of the document below the entity `Address`.

**i Note**

In CDS, a view is an entity without an its own persistence; it is defined as a projection of other entities.

```
view AddressView as select from Address
{
    id,
    street.name,
    street.number
};
```

6. Save the CDS document with the view-definition.

Saving the definition persists the file in your local workspace; it does not create any objects in the database catalog.

7. Deploy the new view (and corresponding tables and types) in the catalog.

To activate the CDS artifacts defined in the CDS document and generate the corresponding objects in the catalog, use the [Build](#) feature.

- a. Right-click the new database module in your application project.
- b. In the context-sensitive pop-up menu, choose  [Build](#).

 **Tip**

You can follow the build progress in the console at the bottom of the CDS editor.

8. Check that the new table, type, and view objects have been successfully created in the catalog.

 **Tip**

A selection of run-time tools is available in the [Database Explorer](#) perspective of SAP web IDE for SAP HANA at the following location:

In XS advanced, your database run-time objects are located in the HDI container created for your multi-target application's database module; you need to locate and bind to this application-specific container to view its contents. The container name contains the name of the user logged into the SAP Web IDE for SAP HANA, the name of the database module containing the CDS design-time entities, and the string `-hdi-container`, for example:

`<XS_UserName>-ctetig24[...]-<DB_Module>-hdi-container`

To bind to the HDI container, in the SAP HANA run-time [Catalog](#) tool, right-click [Catalog](#) in the catalog list, and in the [Search HDI Containers](#) dialog, locate the container to which you want to bind, and choose [Bind](#).

## Related Information

[Set up an HDI Container \[page 176\]](#)

[Create the Data Persistence Artifacts in XS Advanced \[page 200\]](#)

[Create a CDS Document \(XS Advanced\) \[page 208\]](#)

[CDS View Syntax Options \[page 298\]](#)

### 5.3.6.1 CDS Views

A view is an entity that is not persistent; it is defined as the projection of other entities. SAP HANA Extended Application Services (SAP HANA XS) enables you to create a CDS view as a design-time file in the repository.

SAP HANA Extended Application Services (SAP HANA XS) enables you to define a view in a CDS document, which you store as design-time file in the repository. Repository files can be read by applications that you

develop. In addition, all repository files including your view definition can be transported to other SAP HANA systems, for example, in a delivery unit.

If your application refers to the design-time version of a view from the repository rather than the runtime version in the catalog, for example, by using the explicit path to the repository file (with suffix), any changes to the repository version of the file are visible as soon as they are committed to the repository. There is no need to wait for the repository to activate a runtime version of the view.

To define a transportable view using the CDS-compliant view specifications, use something like the code illustrated in the following example:

```
context Views {
    VIEW AddressView AS SELECT FROM Address {
        id,
        street.name,
        street.number
    };
<...>
}
```

When a CDS document is activated, the activation process generates a corresponding catalog object for each of the artifacts defined in the document; the location in the catalog is determined by the type of object generated. For example, in SAP HANA XS classic the corresponding catalog object for a CDS view definition is generated in the following location:

► <SID> ► Catalog ► <MYSHEMA> ► Views ▾

Views defined in a CDS document can make use of the following SQL features:

- CDS Type definition
- Expressions and functions (for example, “`a + b as theSum`”)
- Aggregates, “`GROUP BY`”, and “`HAVING`” clauses
- Associations (including filters and prefixes)
- `ORDER BY`, `CASE`, `UNION`, `JOIN`, and `TOP`
- With Parameters
- Select Distinct
- Spatial Data (for example, “`ST_Distance`”)

➔ Tip

For more information about the syntax required when using these SQL features in a CDS view, see *CDS View Syntax Options* in *Related Information*.

## Type Definition

In a CDS view definition, you can explicitly specify the type of a select item, as illustrated in the following example:

Sample Code

```
type MyInteger : Integer;
entity E {
```

```
a : MyInteger;
b : MyInteger;
};
view V as select from E {
  a,
  a+b as s1,
  a+b as s2 : MyInteger
};
```

In the example of different type definitions, the following is true:

- a,  
Has type “MyInteger”
- a+b as s1,  
Has type “Integer” and any information about the user-defined type is lost
- a+b as s2 : MyInteger  
Has type “MyInteger”, which is explicitly specified

#### i Note

If necessary, a CAST function is added to the generated view in SAP HANA; this ensures that the select item's type in the SAP HANA view is the SAP HANA “type” corresponding to the explicitly specified CDS type.

## Related Information

[Create a CDS View in XS Advanced \[page 293\]](#)

[CDS View Syntax Options \[page 298\]](#)

[CDS Associations \[page 281\]](#)

### 5.3.6.2 CDS View Syntax Options

SAP HANA XS includes a dedicated, CDS-compliant syntax, which you must adhere to when using a CDS document to define a view as a design-time artifact.

#### Example

#### i Note

The following example is intended for illustration purposes only and might contain syntactical errors. For further details about the keywords illustrated, click the links provided.

```
context views {
  const x : Integer = 4;
  const y : Integer = 5;
  const z : Integer = 6;
  VIEW MyView1 AS SELECT FROM Employee
```

```

{
    a + b AS theSum
},
VIEW MyView2 AS SELECT FROM Employee
{ officeId.building,
  officeId.floor,
  officeId.roomNumber,
  office.capacity,
  count(id) AS seatsTaken,
  count(id)/office.capacity as occupancyRate
} WHERE officeId.building = 1
GROUP BY officeId.building,
         officeId.floor,
         officeId.roomNumber,
         office.capacity,
         office.type
HAVING office.type = 'office' AND count(id)/office.capacity < 0.5;
VIEW MyView3 AS SELECT FROM Employee
{ orgUnit,
  salary
} ORDER BY salary DESC;
VIEW MyView4 AS SELECT FROM Employee {
CASE
    WHEN a < 10 then 'small'
    WHEN 10 <= a AND a < 100 THEN 'medium'
    ELSE 'large'
END AS size
};
VIEW MyView5 AS
  SELECT FROM E1 { a, b, c}
  UNION
  SELECT FROM E2 { z, x, y};
VIEW MyView6 AS SELECT FROM Customer {
  name,
  orders[status='open'].{ id   as orderId,
                        date as orderDate,
                        items[price>200].{ descr,
                                             price }  }
};

VIEW MyView7 as
  select from E { a, b, c}
  order by a limit 10 offset 30;
VIEW V_join as select from E join (F as X full outer join G on X.id = G.id) on
E.id = c {
  a, b, c
};
VIEW V_top as select from E TOP 10 { a, b, c};
VIEW V_dist as select from E distinct { a };
VIEW V_param with parameters PAR1: Integer, PAR2: MyUserDefinedType, PAR3:
type of E_elt
  as select from MyEntity {
    id,
    elt };
VIEW V_type as select from E {
  a,
  a+b as s1,
  a+b as s2 : MyInteger
};
view VE as select from E mixin {
  f : Association[1] to VF on f.vy = $projection.vb;
} into {
  a as va,
  b as vb,
  f as vf
};
VIEW SpatialView1 as select from Person {
  name,
  homeAddress.street_name || ', ' || homeAddress.city as home,

```

```

    officeAddress.street_name || ', ' || officeAddress.city as office,
    round( homeAddress.loc.ST_Distance(officeAddress.loc, 'meter')/1000, 1) as
distanceHomeToWork,
    round( homeAddress.loc.ST_Distance(NEW ST_POINT(8.644072, 49.292910),
'meter')/1000, 1) as distFromSAP03
};
}

```

## Expressions and Functions

In a CDS view definition you can use any of the functions and expressions listed in the following example:

```

View MyView9 AS SELECT FROM SampleEntity
{
  a + b  AS theSum,
  a - b  AS theDifference,
  a * b  AS theProduct,
  a / b  AS theQuotient,
  -a      AS theUnaryMinus,
  c || d AS theConcatenation
};

```

### Note

When expressions are used in a view element, an alias must be specified, for example, AS theSum.

## Aggregates

In a CDS view definition, you can use the following aggregates:

- AVG
- COUNT
- MIN
- MAX
- SUM
- STDDEV
- VAR

The following example shows how to use aggregates and expressions to collect information about headcount and salary per organizational unit for all employees hired from 2011 to now.

```

VIEW MyView10 AS SELECT FROM Employee
{
  orgUnit,
  count(id)   AS headCount,
  sum(salary) AS totalSalary,
  max(salary) AS maxSalary
}
WHERE joinDate > date'2011-01-01'
GROUP BY orgUnit;

```

### Note

Expressions are not allowed in the GROUP BY clause.

## Constants in Views

With SPS 11, you can use constants in the views, as illustrated in “MyView” at the end of the following example:

### Sample Code

```
context MyContext {
    const MyIntConst      : Integer      = 7;
    const MyStringConst   : String(10)   = 'bright';
    const MyDecConst     : Decimal(4,2) = 3.14;
    const MyDateTimeConst : UTCDateTime = '2015-09-30 14:33';
    entity MyEntity {
        key id : Integer;
        a : Integer;
        b : String(100);
        c : Decimal(20,10);
        d : UTCDateTime;
        your : Association[1] to YourEntity on your.a - a < MyIntConst;
    };
    entity YourEntity {
        key id : Integer;
        a : Integer;
    };
    entity HerEntity {
        key id : Integer;
        t : String(20);
    };
    view MyView as select from MyEntity
        inner join HerEntity on locate (b, :MyStringConst) > 0
    {
        a + :MyIntConst as x,
        b || ' is ' || :MyStringConst as y,
        c * sin(:MyDecConst) as z
    } where d < :MyContext.MyDateTimeConst;
};
```

When constants are used in a view definition, their name must be prefixed with the scope operator “:”. Usually names that appear in a query are resolved as alias or element names. The scope operator instructs the compiler to resolve the name outside of the query.

### Sample Code

```
context NameResolution {
    const a : Integer = 4;
    const b : Integer = 5;
    const c : Integer = 6;
    entity E {
        key id : Integer;
        a : Integer;
        c : Integer;
    };
    view V as select from E {
        a as a1,
```

```

    b,
    :a as a2,
E.a as a3,
:E,
:E.a as a4,
:c
};

}

```

The following table explains how the constants used in view “V” are resolved.

Table 35: Constant Declaration and Result

Constant Expression	Result	Comments
a as a1,	Success	“a” is resolved in the space of alias and element names, for example, element “a” of entity “E”.
b,	Error	There is no alias and no element with name “b” in entity “E”
:a as a2,	Success	Scope operator “:” instructs the compiler to search for element “a” outside of the query (finds the constant “a”).
E.a as a3,	Success	“E” is resolved in the space of alias and element names, so this matches element “a” of entity “Entity” .
:E,	Success	Error: no access to “E” via “:”
:E.a as a4,	Error	Error: no access to “E” (or any of its elements) via “:”
:c	Error	Error: there is no alias for “c”.

## SELECT

In the following example of an association in a `SELECT` list, a view compiles a list of all employees; the list includes the employee's name, the capacity of the employee's office, and the color of the carpet in the office. The association follows the to-one association `office` from entity `Employee` to entity `Room` to collect the relevant information about the office.

```

VIEW MyView11 AS SELECT FROM Employee
{
  name.last,
  office.capacity,
  office.carpetColor
};

```

## Subqueries

You can define subqueries in a CDS view, as illustrated in the following example:

### Code Syntax

```
select from (select from F {a as x, b as y}) as Q {
```

```
x+y as xy,  
  (select from E {a} where b=Q.y) as a  
} where x < all (select from E{b})
```

## ⚠ Restriction

In a **correlated** subquery, elements of outer queries must always be addressed by means of a table alias.

## WHERE

The following example shows how the syntax required in the WHERE clause used in a CDS view definition. In this example, the WHERE clause is used in an association to restrict the result set according to information located in the association's target. Further filtering of the result set can be defined with the AND modifier.

```
VIEW EmployeesInRoom_ABC_3_4 AS SELECT FROM Employee  
{  
    name.last  
} WHERE officeId.building = 'ABC'  
      AND officeId.floor     = 3  
      AND officeId.number    = 4;
```

## FROM

The following example shows the syntax required when using the FROM clause in a CDS view definition. This example shows an association that lists the license plates of all company cars.

```
VIEW CompanyCarLicensePlates AS SELECT FROM Employee.companyCar  
{  
    licensePlate  
};
```

In the FROM clause, you can use the following elements:

- an entity or a view defined in the same CDS source file
- a native SAP HANA table or view that is available in the schema specified in the schema annotation (@Schema in the corresponding CDS document)

If a CDS view references a native SAP HANA table, the table and column names must be specified using their effective SAP HANA names.

```
create table foo (  
    bar    : Integer,  
    "gloo" : Integer  
)
```

This means that if a table (foo) or its columns (bar and "gloo" were created **without** using quotation marks (""), the corresponding uppercase names for the table or columns must be used in the CDS document, as illustrated in the following example.

```
VIEW MyViewOnNative as SELECT FROM FOO
```

```
{  
    BAR,  
    gloo  
};
```

## GROUP BY

The following example shows the syntax required when using the GROUP BY clause in a CDS view definition. This example shows an association in a view that compiles a list of all offices that are less than 50% occupied.

```
VIEW V11 AS SELECT FROM Employee  
{  
    officeId.building,  
    officeId.floor,  
    officeId.roomNumber,  
    office.capacity,  
    count(id) as seatsTaken,  
    count(id)/office.capacity as occupancyRate  
} GROUP BY officeId.building,  
        officeId.floor,  
        officeId.roomNumber,  
        office.capacity,  
        office.type  
HAVING office.type = 'office' AND count(id)/capacity < 0.5;
```

## HAVING

The following example shows the syntax required when using the HAVING clause in a CDS view definition. This example shows a view with an association that compiles a list of all offices that are less than 50% occupied.

```
VIEW V11 AS SELECT FROM Employee  
{  
    officeId.building,  
    officeId.floor,  
    officeId.roomNumber,  
    office.capacity,  
    count(id)      as seatsTaken,  
    count(id)/office.capacity as occupancyRate  
} GROUP BY officeId.building,  
        officeId.floor,  
        officeId.roomNumber,  
        office.capacity,  
        office.type  
HAVING office.type = 'office' AND count(id)/capacity < 0.5;
```

## ORDER BY

The ORDER BY operator enables you to list results according to an expression or position, for example salary.

```
VIEW MyView3 AS SELECT FROM Employee
```

```
{  
    orgUnit,  
    salary  
} ORDER BY salary DESC;
```

In the same way as with plain SQL, the `ASC` and `DESC` operators enable you to sort the list order as follows.

- `ASC`  
Display the result set in **ascending** order
- `DESC`  
Display the result set in **descending** order

## LIMIT/OFFSET

You can use the SQL clauses `LIMIT` and `OFFSET` in a CDS query. The `LIMIT <INTEGER> [OFFSET <INTEGER>]` operator enables you to restrict the number of output records to display to a specified “limit”; the `OFFSET <INTEGER>` specifies the number of records to skip before displaying the records according to the defined `LIMIT`.

```
VIEW MyViewV AS SELECT FROM E  
{ a, b, c}  
order by a limit 10 offset 30;
```

## CASE

In the same way as in plain SQL, you can use the `case` expression in a CDS view definition to introduce `IF-THEN-ELSE` conditions without the need to use procedures.

```
entity MyEntity12 {  
key id : Integer;  
    a : Integer;  
    color : String(1);  
};  
  
VIEW MyView12 AS SELECT FROM MyEntity12 {  
    id,  
    CASE color // defined in MyEntity12  
        WHEN 'R' THEN 'red'  
        WHEN 'G' THEN 'green'  
        WHEN 'B' THEN 'blue'  
        ELSE 'black'  
    END AS color,  
    CASE  
        WHEN a < 10 then 'small'  
        WHEN 10 <= a AND a < 100 THEN 'medium'  
        ELSE 'large'  
    END AS size  
};
```

In the first example of usage of the `CASE` operator, `CASE color` shows a “switched” `CASE` (one table column and multiple values). The second example of `CASE` usage shows a “conditional” `CASE` with multiple arbitrary conditions, possibly referring to different table columns.

## UNION

Enables multiple select statements to be combined but return only one result set. UNION works in the same way as the SAP HANA SQL command of the same name; it selects all unique records from all select statements by removing duplicates found from different select statements. The signature of the result view is equal to the signature of the first SELECT in the union.

### i Note

View MyView5 has elements a, b, and c.

```
entity E1 {  
    key a : Integer;  
    b : String(20);  
    c : LocalDate;  
};  
entity E2 {  
    key x : String(20);  
    y : LocalDate;  
    z : Integer;  
};  
VIEW MyView5 AS  
    SELECT FROM E1 { a, b, c}  
    UNION  
    SELECT FROM E2 { z, x, y};
```

## JOIN

You can include a JOIN clause in a CDS view definition; the following JOIN types are supported:

- [ INNER ] JOIN
- LEFT [ OUTER ] JOIN
- RIGHT [ OUTER ] JOIN
- FULL [ OUTER ] JOIN
- CROSS JOIN

The following example shows a simple join.

### Sample Code

```
entity E {  
    key id : Integer;  
    a : Integer;  
};  
entity F {  
    key id : Integer;  
    b : Integer;  
};  
entity G {  
    key id : Integer;  
    c : Integer;  
};  
view V_join as select from E join (F as X full outer join G on X.id = G.id) on  
E.id = c {  
    a, b, c
```

```
};
```

## TOP

You can use the SQL clause `TOP` in a CDS query, as illustrated in the following example:

### Sample Code

```
view V_top as select from E TOP 10 { a, b, c};
```

### ⚠ Restriction

It is not permitted to use `TOP` in combination with the `LIMIT` clause in a CDS query.

## SELECT DISTINCT

CDS now supports the `SELECT DISTINCT` semantic. The position of the `DISTINCT` keyword is important; it must appear directly in front of the curly brace, as illustrated in the following example:

### Sample Code

```
entity E {
    key id : Integer;
    a : Integer;
};
entity F {
    key id : Integer;
    b : Integer;
};
entity G {
    key id : Integer;
    c : Integer;
};
view V_dist as select from E distinct { a };
```

## With Parameters

You can define parameters for use in a CDS view; this allows you to pass additional values to modify the results of the query at runtime. Parameters must be defined in the view definition before the query block, as illustrated in the following example:

### Note

Keywords are case insensitive.

## Sample Code

### Parameters in a CDS View

```
context MyContext
{
    entity MyEntity1 {
        id: Integer;
        elt: String(100); };
    entity MyEntity2 {
        id: Integer;
        elt: String(100); };
    type MyUserDefinedType: type of E_elt;
    view MyParamView with parameters PAR1: Integer,
                           PAR2: MyUserDefinedType,
                           PAR3: type of E_elt
        as select from MyEntity {
            id,
            elt };
```

### Parameters in View Queries

Parameters can be used in a query at any position where an expression is allowed. A parameter is referred to inside a query by prefixing the parameter name either with the colon Scope operator ":" or the string "\$parameters".

#### Tip

If no matching parameter can be found, the scope operator "escapes" from the query and attempts to resolve the identifier outside the query.

## Sample Code

### Using Parameters in a View Query

```
view ExampleView with parameters PAR1: Integer,
                           PAR2: UserDefinedType,
                           PAR3: type of E_elt
        as select from SomeEntity
            left outer join SomeOtherEntity
                on SomeEntity.id < SomeOtherEntity.id + :PAR1
{
    id + :PAR1 as idWithOffset,
    elt,
    :PAR1,
    $parameters.PAR3
} where elt != $parameters.PAR2;
```

### Invoking a View with Parameters

Parameters are passed to views as a comma-separated list in parentheses. Optional filter expressions must then follow the parameter list.

#### Restriction

It is not allowed to use a query as value expression. Nor is it allowed to provide a parameter list in the ON condition of an association definition to a parameterized view. This is because the association definition

establishes the relationship between the two entities but makes no assumptions about the run-time conditions. For the same reason, it is not allowed to specify filter conditions in those `ON` conditions.

The following example shows two entities `SourceEntity` and `TargetEntity` and a parameterized view `TargetWindowView`, which selects from `TargetEntity`. An association is established between `SourceEntity` and `TargetEntity`.

### Sample Code

```
entity SourceEntity {
    id: Integer;
    someElementOfSourceEntity: String(100);
    toTargetViaParamView: association to TargetWindowView on
        toTargetViaParamView.targetId = id;
};

entity TargetEntity {
    targetId: Integer;
    someElementOfTargetEntity: String(100);
};

view TargetWindowView with parameters LOWER_LIMIT: Integer
as select from TargetEntity {
    targetId,
    someElementOfTargetEntity
} where targetId > :LOWER_LIMIT
    and targetId <= :LOWER_LIMIT + 10;
```

It is now possible to query `SourceEntity` in a view; it is also possible to follow the association to `TargetWindowView`, for example, by providing the required parameters, as illustrated in the following example:

### Sample Code

Query a Parameterized CDS View (with Association)

```
view SourceConsumption with parameters CUSTOMER_ID: Integer
as select from SourceEntity {
    someElementOfSourceEntity,
    toTargetViaParamView(LOWER_LIMIT:
$parameters.CUSTOMER_ID).someElementOfTargetEntity
};
```

It is also possible to follow the association in the `FROM` clause; this provides access only to the elements of the target artifact:

### Sample Code

Follow an Association in the FROM Clause

```
view ConsumptionView with parameters CUSTOMER_ID: Integer
as select from SourceEntity.toTargetViaParamView(LOWER_LIMIT: :CUSTOMER_ID)
{
    id,
    someElementOfTargetEntity
};
```

You can select directly from the view with parameters, adding a free JOIN expression, as illustrated in the following example:

### Sample Code

Select from a Parameterized View with JOIN Expression

```
view ConsumptionView with parameters CUSTOMER_ID: Integer
  as select from TargetWindowView(LOWER_LIMIT: :CUSTOMER_ID) as TWV_ALIAS
    RIGHT OUTER JOIN ... ON TWV_ALIAS.targetId ....
{
  ...
};
```

### Annotations in Parameter Definitions

Parameter definitions can be annotated in the same way as any other artifact in CDS; the annotations must be prepended to the parameter name. Multiple annotations are separated either by whitespace or new-line characters.

#### Tip

To improve readability and comprehension, it is recommended to include only one annotation assignment per line.

In the following example, the view TargetWindowView selects from the entity TargetEntity; the annotation @positiveValuesOnly is not checked; and the targetId is required for the ON condition in the entity SourceEntity.

### Sample Code

Annotation Assignments to Parameter Definitions in CDS Views

```
annotation remark: String(100);

view TargetWindowView with parameters
  @remark: 'This is an arbitrary annotation'
  @positiveValuesOnly: true
  LOWER_LIMIT: Integer
as select from TargetEntity
{
  targetId,
  ....
} where targetId > :LOWER_LIMIT and targetId <= :LOWER_LIMIT + 10;
```

## Associations, Filters, and Prefixes

You can define an association as a view element, for example, by defining an ad-hoc association in the `mixin` clause and then adding the association to the `SELECT` list, as illustrated in the following example:

### Sample Code

#### Associations as View Elements

```
entity E {
    a : Integer;
    b : Integer;
};

entity F {
    x : Integer;
    y : Integer;
};

view VE as select from E mixin {
    f : Association[1] to VF on f.vy = $projection.vb;
} into {
    a as va,
    b as vb,
    f as vf
};
view VF as select from F {
    x as vx,
    y as vy
};
```

In the `ON` condition of this type of association in a view, it is necessary to use the pseudo-identifier `$projection` to specify that the following element name must be resolved in the `select` list of the view ("VE") rather than in the entity ("E") in the `FROM` clause

### Filter Conditions

It is possible to apply a filter condition when resolving associations between entities; the filter is merged into the `ON`-condition of the resulting `JOIN`. The following example shows how to get a list of customers and then filter the list according to the sales orders that are currently "open" for each customer. In the example, the filter is inserted after the association `orders`; this ensures that the list displayed by the view only contains those orders that satisfy the condition `[status='open']`.

### Sample Code

```
view C1 as select from Customer {
    name,
    orders[status='open'].id as orderId
};
```

The following example shows how to use the prefix notation to ensure that the compiler understands that there is only one association (`orders`) to resolve but with multiple elements (`id` and `date`):

### Sample Code

```
view C1 as select from Customer {
    name,
    orders[status='open'].{ id   as orderId,
```

```
        date as orderDate
    }
};
```

## ➔ Tip

Filter conditions and prefixes can be nested.

The following example shows how to use the associations `orders` and `items` in a view that displays a list of customers with open sales orders for items with a price greater than 200.

### Sample Code

```
view C2 as select from Customer {
    name,
    orders[status='open'].{ id      as orderId,
                           date    as orderDate,
                           items[price>200].{ descr,
                                              price
                                         }
                           }
};
```

## Prefix Notation

The prefix notation can also be used without filters. The following example shows how to get a list of all customers with details of their sales orders. In this example, all uses of the association `orders` are combined so that there is only one `JOIN` to the table `SalesOrder`. Similarly, both uses of the association `items` are combined, and there is only one `JOIN` to the table `Item`.

### Sample Code

```
view C3 as select from Customer {
    name,
    orders.id          as orderId,
    orders.date        as orderDate,
    orders.items.descr as itemDescr,
    orders.items.price as itemPrice
};
```

The example above can be expressed more elegantly by combining the associations `orders` and `items` using the following prefix notation:

### Sample Code

```
view C1 as select from Customer {
    name,
    orders.{ id      as orderId,
              date    as orderDate,
              items. { descr as itemDescr,
                        price as itemPrice
                      }
              }
};
```

## Type Definition

In a CDS view definition, you can explicitly specify the type of a select item, as illustrated in the following example:

### Sample Code

```
type MyInteger : Integer;
entity E {
    a : MyInteger;
    b : MyInteger;
};
view V as select from E {
    a,
    a+b as s1,
    a+b as s2 : MyInteger
};
```

In the example of different type definitions, the following is true:

- a,  
Has type “MyInteger”
- a+b as s1,  
Has type “Integer” and any information about the user-defined type is lost
- a+b as s2 : MyInteger  
Has type “MyInteger”, which is explicitly specified

### Note

If necessary, a CAST function is added to the generated view in SAP HANA; this ensures that the select item's type in the SAP HANA view is the SAP HANA “type” corresponding to the explicitly specified CDS type.

## Spatial Functions

The following view (SpatialView1) displays a list of all persons selected from the entity Person and uses the spatial function ST\_Distance (\*) to include information such as the distance between each person's home and business address (distanceHomeToWork), and the distance between their home address and the building SAP03 (distFromSAP03). The value for both distances is measured in kilometers, which is rounded up and displayed to one decimal point.

### Sample Code

```
view SpatialView1 as select from Person {
    name,
    homeAddress.street_name || ', ' || homeAddress.city as home,
    officeAddress.street_name || ', ' || officeAddress.city as office,
    round( homeAddress.loc.ST_Distance(officeAddress.loc, 'meter')/1000, 1) as
    distanceHomeToWork,
    round( homeAddress.loc.ST_Distance(NEW ST_POINT(8.644072, 49.292910),
    'meter')/1000, 1) as distFromSAP03
```

```
};
```

### Caution

(\*) 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 options is available in SAP Help Portal. 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.

## Related Information

[Spatial Types and Functions \[page 314\]](#)

### 5.3.6.3 Spatial Types and Functions

CDS supports the use of Geographic Information Systems (GIS) functions and element types in CDS-compliant entities and views.

Spatial data is data that describes the position, shape, and orientation of objects in a defined space; the data is represented as two-dimensional geometries in the form of points, line strings, and polygons. The following examples shows how to use the spatial function `ST_Distance` in a CDS view. The underlying spatial data used in the view is defined in a CDS entity using the type `ST_POINT`.

The following example, the CDS entity `Address` is used to store geo-spatial coordinates in element `loc` of type `ST_POINT`:

#### Sample Code

```
namespace samples;
@Schema: 'MYSCHHEMA'
context Spatial {
    entity Person {
        key id : Integer;
        name : String(100);
        homeAddress : Association[1] to Address;
```

```

        officeAddress : Association[1] to Address;
    };
    entity Address {
        key id : Integer;
        street_number : Integer;
        street_name : String(100);
        zip : String(10);
        city : String(100);
        loc : hana.ST_POINT(4326);
    };
    view GeoView1 as select from Person {
        name,
        homeAddress.street_name || ', ' || homeAddress.city as home,
        officeAddress.street_name || ', ' || officeAddress.city as office,
        round( homeAddress.loc.ST_Distance(officeAddress.loc, 'meter')/1000,
1) as distanceHomeToWork,
        round( homeAddress.loc.ST_Distance(NEW ST_POINT(8.644072, 49.292910),
'meter')/1000, 1) as distFromSAP03
    };
}

```

The view `GeoView1` is used to display a list of all persons using the spatial function `ST_Distance` to include information such as the distance between each person's home and business address (`distanceHomeToWork`), and the distance between their home address and the building SAP03 (`distFromSAP03`). The value for both distances is measured in kilometers.

### Caution

(\*) 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 options is available in SAP Help Portal. 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.

## Related Information

[CDS View Syntax Options \[page 298\]](#)

[CDS Primitive Data Types \[page 274\]](#)

## 5.3.7 Create a CDS Extension

Define the artifacts required to extend an existing CDS model.

### Prerequisites

- You have access to the Web-based development tools included with SAP Web IDE for SAP HANA
- Each CDS extension package must have the following elements:
  - The package descriptor (`.package.hdbc`)  
The package descriptor for a CDS extension has no name, only the suffix. Its contents must conform to the required syntax.
  - The CDS extension descriptor (`myCDSExtension.hdbc`)  
The extension descriptor's contents must conform to the required syntax.

### Context

In this simple CRM scenario, the base application consists of a “type” named `Address` and an entity named `Customer`. In the first extension package, `banking`, we add a new “type” named `BankingAccount` and a new “element” named `account` to the entity `Customer`. In a further extension package named `onlineBanking` that depends on the package `banking` we add a new element to type `BankingAccount` and add a new element to type `Address`.

#### Sample Code

```
<myCDSBankingApp>
|- db/
|   |- package.json                                # Database deployment artifacts
|   \- src/
|       |- .hdiconfig                               # HDI build plug-in configuration
|       |- .hdinamespace                           # HDI run-time name-space config
|       |- Address.hdbc
|       |- CRM.hdbc
|       |- banking/
|           |- package.hdbc                         # CDS extension package
|               |- BankingExtension.hdbc            # CDS extension package descriptor
|           \- onlineBanking/                      # CDS extension package
|               |- package.hdbc                     # CDS extension package descriptor
|                   \- OnlineBankingExtension.hdbc # CDS extension description
|- web/
|- js/
|- xs-security.json
\- mtad.yaml
```

## Procedure

1. Start the SAP HANA Web IDE for SAP HANA.

The SAP Web IDE for SAP HANA is available at the following URL:

`https://<HANA_HOST>:53075/`

➔ **Tip**

To display the URL for the SAP Web IDE for SAP HANA, open a command shell, log on to the XS advanced run time, and run the following command:

```
xs app webide --urls
```

2. Display the application project to which you want to add a CDS document.

In XS advanced, SAP Web IDE for SAP HANA creates an application within a context of a project. If you do not already have a project, there are a number of ways to create one, for example: by importing it, cloning it, or creating a new one from scratch.

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

Currently, there is only one type of project template available, namely: *Multi-Target Application Project*. Select *Multi-Target Application Project* and choose *Next*.

- c. Type a name for the new MTA project (for example, `myCDSApp`) and choose *Next* to confirm.
- d. Specify details of the new MTA project and choose *Next* to confirm.
- e. Create the new MTA project; choose *Finish*.

3. Navigate to the database module of the application for which you want to create CDS extensions.

In SAP Web IDE for SAP HANA, database artifacts such as the ones defined in a CDS document belong in the MTA's database "module".

➔ **Tip**

If you do not already have a database module, right-click the root folder of your new MTA project and, in the context menu, choose **New** **HDB Module**. Name the new database model `db`.

4. Create the base database application using CDS.

The base CDS application in `myCDSBankingApp/src/` must contain the following artifacts:

**i Note**

For the purposes of this tutorial, we are using the base CDS documents `Address.hdbc` and `CRM.hdbc`; an existing application would have different CDS documents. However, the `.hdinamespace` and `.hdiconfig` files would be present.

- `.hdinamespace`  
The name-space definition to use when deploying the database application
- `.hdiconfig`  
The list of plug-ins to use to create catalog objects when deploying the database application

- Address.hdbcards  
A CDS document containing the definition of the CDS data “`type`” Address
  - CRM.hdbcards  
A CDS document named `CRM.hdbcards` which contains the definition of the CDS “`entity`” Customer
- a. Define the name space that applies to this CDS application.

The name space to use for the deployment of the CDS application is defined in the configuration file `.hdinamespace`; in this case, it should look like the following example:

### Note

The “`append`” value ensures that the name-space rule applies to **\*all\*** subfolders in the CDS application structure.

### Sample Code

```
.hdinamespace

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

- b. Create the CDS data type `Address.hdbcards`.

Navigate to the application's database module `db/`, right-click the folder `db/src/` and choose  **New**  in the context menu.

The CDS data type definition should look like the following example:

### Sample Code

```
type Address {
  zipCode : String(5);
  city : String(40);
  street : String(40);
  nr : String(10);
};
```

- c. Create the CDS document `CRM.hdbcards`.

Navigate to the `src/` folder in your application's database module `db/`, right-click the folder `db/src/` and choose  **New**  in the context menu.

The CDS definition for the `Customer` entity (table) should look like the following example:

### Sample Code

```
using Address;
context CRM {
  entity Customer {
    name : String(40);
    address : Address;
  };
};
```

5. Create a CDS extension called “banking”.

The CDS extension banking must contain the following artifacts:

- `.package.hdbc`  
A CDS document containing the definition of the CDS extension package `banking`/
  - `BankingExtension.hdbc`  
A CDS document containing the definition of the CDS extension `BankingExtension`
- a. Create a new folder for the CDS extension `banking/`.

Navigate to the `src/` folder in your application's database module `db/`, right-click the folder `db/src/` and choose `New > Folder` in the context menu. Name the new folder “`banking`”.

- b. Create the CDS extension package descriptor `.package.hdbc`.

**Note**

The leading dot (.) in the extension-package file name is mandatory.

Navigate to the `src/` folder in your application's database module `db/`, right-click the folder `db/src/banking` and choose `New > CDS Artifact` in the context menu. Name the new artifact `.package.hdbc`.

The CDS extension **package** definition should look like the following example:

**Sample Code**

```
package banking;
```

- c. Create the CDS extension descriptor `BankingExtension.hdbc`.

Navigate to the `src/` folder in your application's database module `db/`, right-click the folder `db/src/banking` and choose `New > CDS Artifact` in the context menu.

The CDS extension definition `BankingExtension.hdbc` should look like the following example:

**Sample Code**

```
namespace banking;
in package banking;
using CRM;
using CRM.Customer;
extend context CRM with {
    type BankingAccount {
        BIC : String(8);
        IBAN : String(120);
    };
};
extend entity Customer with {
    account: CRM.BankingAccount;
};
```

6. Create a CDS extension called “`onlineBanking`”.

The CDS extension `onlineBanking` must contain the following artifacts:

- `.package.hdbc`

- A CDS document containing the description of the CDS extension package `onlineBanking/`
- o `BankingExtension.hdbc`  
A CDS document containing the description of the CDS extension `OnlineBankingExtension`

a. Create a new folder for the CDS extension `onlineBanking/`.

    Navigate to the `src/` folder in your application's database module `db/`, right-click the folder `db/src/` and choose **New > Folder** in the context menu. Name the new folder “`onlineBanking`”.

b. Create the CDS extension package descriptor `.package.hdbc`.

#### Note

The leading dot (.) in the package file name is mandatory.

    Navigate to the `src/` folder in your application's database module `db/`, right-click the folder `db/src/` `onlineBanking` and choose **New > CDS Artifact** in the context menu. Name the new artifact `.package.hdbc`.

The CDS extension **package** descriptor should look like the following example:

#### Sample Code

```
package onlineBanking depends on banking;
```

c. Create the CDS extension descriptor `BankingExtension.hdbc`.

    Navigate to the `src/` folder in your application's database module `db/`, right-click the folder `db/src/` `onlineBanking` and choose **New > CDS Artifact** in the context menu. Name the new CDS artifact `BankingExtension.hdbc`

The CDS extension definition `BankingExtension.hdbc` should look like the following example:

#### Sample Code

```
namespace onlineBanking;
in package onlineBanking;
using Address;
using CRM.BankingAccount;
extend type Address with {
    email : String(60);
};
extend type BankingAccount with {
    PIN : String(5);
};
```

7. Build the CDS application's database module.

Building a database module activates the data model and creates corresponding object in the database catalog for each artifact defined in the CDS document. In this case, the build creates all the CDS artifacts for the base CDS application as well as the artifacts defined in the two extension packages.

In SAP Web IDE for SAP HANA, right-click the CDS application's database module (`<myCDSapp>/db`) and choose **Build > Build** in the context-sensitive menu.

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

## Related Information

- [The CDS Extension Descriptor \[page 321\]](#)
- [The CDS Extension Descriptor Syntax \[page 324\]](#)
- [The CDS Extension Package Descriptor \[page 329\]](#)
- [The CDS Extension Package Descriptor Syntax \[page 329\]](#)

### 5.3.7.1 The CDS Extension Descriptor

Defines in a separate file the properties required to modify an existing CDS artifact definition.

The CDS extension mechanism allows you to add properties to existing artifact definitions without modifying the original source files. In this way, you can split the definition of an artifact across multiple files each of which can have a different life cycle and code owner. For example, a customer can add a new element to an existing entity definition by the following statement:

#### Sample Code

CDS Artifact Extension Syntax

```
extend EntityE with {  
    newElement: Integer;  
}
```

In the example above, the code illustrated shows how to define a new **element** inside an existing entity (`EntityE`) artifact.

#### Note

The `extend` statement changes an existing artifact; it does not define any additional artifact.

It is essential to ensure that additional element definitions specified in custom extensions do not break the existing definitions of the base application. This is achieved by adapting the name-search rules and by additional checks for the `extend` statements. For the definition of these rules and checks, it is necessary to define the relationship between an `extend` statement and the artifact definitions, as well as the relationship between an `extend` statement and any additional `extend` statements.

## Organization of Extensions

When you extend an SAP application, you typically add new elements to entities or views; these additional elements usually work together and can, themselves, require additional artifacts, for example, “types” used as

element “types”. To facilitate the process, we define an extension package (or package for short), which is a set of extend statements, normal artifact definitions (for example, “types” which are used in an extend declaration), and extension relationships (also known as “dependencies”). Each CDS source file belongs to exactly one package; all the definitions in this file contribute to that one (single) package. However, a “package” typically contains contributions from multiple CDS source files.

### ➔ Tip

It is also possible to use a package to define a clear structure for an application, even if no extensions are involved.

## Package Hierarchies

The extension mechanism can be used by developers as well as SAP industry solutions, partners, and customers. A productive system is likely to have more than one package; some packages might be independent from each other; some packages might depend on other packages. With such a model, we get an acyclic directed graph, with the base application and the extension packages as nodes and the dependencies as edges. This induces a partial order on the packages with the base application as lowest package (for simplicity we also call the base application a package). There is not necessarily a single top package (here: the final customer extension).

It is essential to ensure that which package is semantically self-contained and self-explanatory; avoid defining “micro” packages which can be technically applied individually but have no independent business value.

### ⚠ Restriction

Cyclic dependencies between extension packages are not allowed.

## Package Definition

It is necessary to specify which `extend` statements and normal artifact definitions belong to which package and, in addition, on which other packages a package depends. A package is considered to be a normal CDS artifact; it has a name, and a corresponding definition, and its use can be found in the CDS Catalog. An extension package is defined by a special CDS source file with the file suffix `.package.hdbc`.

### i Note

The full stop (.) **before** the extension-package file name is mandatory.

The following simple code example illustrates the syntax required for defining a CDS extension package and its dependencies:

### Sample Code

#### CDS Extension Package Syntax

```
source = packageDefinition
```

```

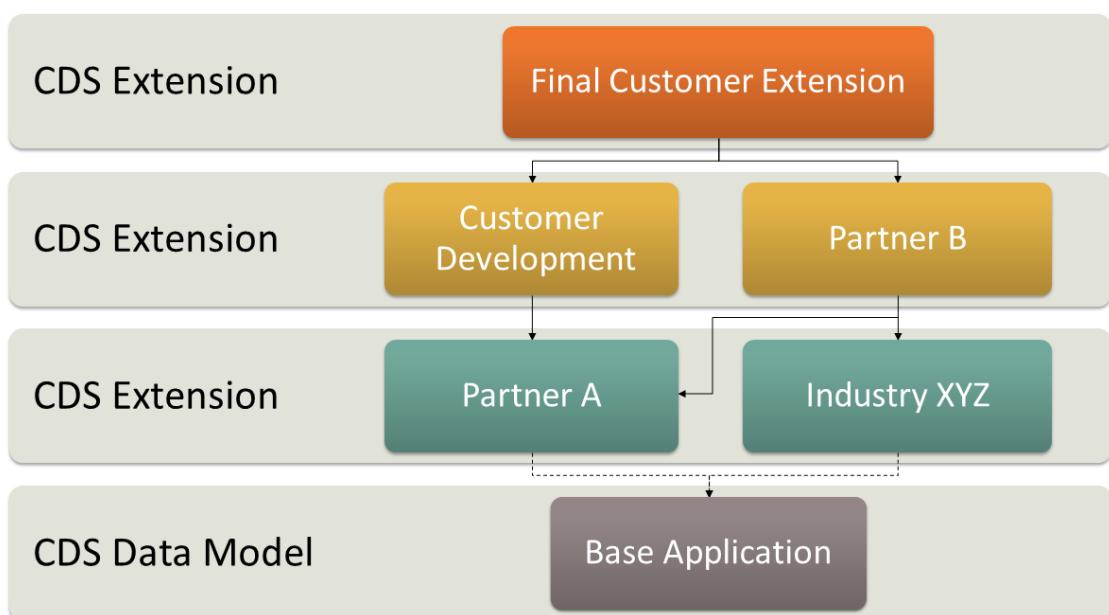
packageDefinition = "package" packageName (
    "depends" "on" packageName ( "," packageName )* )? ";" 
packageName = identifier ( "." identifier )*

```

The name of the package defined in the file must be identical to the name space that is applicable for the file (as specified in the relevant HDI container-configuration file (`.hdinamespace`)).

### ➔ Tip

The base package is not explicitly defined; it contains all CDS sources that are not explicitly assigned to a package.



To define a package hierarchy according to the diagram above, the following package definition files need to be provided (the names are just an example and do not confirm to any recommended naming convention):

Table 36: Package Hierarchy and Definition Files

Package Directory (Namespace)	Content of Package Definition File ( <code>.package.hdbc</code> )
<code>myapp.partners.A</code>	<code>package myapp.partners.A;</code>
<code>myapp.industries.XYZ</code>	<code>package myapp.industries.XYZ;</code>
<code>myapp.customDev</code>	<code>package myapp.customDev depends on myapp.partners.A;</code>
<code>myapp.partners.B</code>	<code>package myapp.partners.B depends on myapp.partners.A, myapp.industries.XYZ;</code>
<code>myapp.customer</code>	<code>package myapp.customer depends on myapp.customDev, myapp.partners.B;</code>

## Package and Source-File Assignment

To ensure that all the definitions in the CDS source, both normal artifact definitions and extend statements, belong to the respective package, you must assign a CDS source file to an extension package. To assign a CDS source to an extension package, add the statement `in package` to the beginning of the CDS source file, as illustrated in the following example:

### Sample Code

CDS Extension Package Assignment

```
in package <packageName>;
```

### Note

The `in package` statement must be placed at the beginning of the CDS document, for example, before (or after) the name-space declaration, if present, but in all cases **before** all other statements in the CDS document.

## Related Information

[Create a CDS Extension \[page 316\]](#)

[The CDS Extension Descriptor Syntax \[page 324\]](#)

[The CDS Extension Package Descriptor \[page 329\]](#)

## 5.3.7.2 The CDS Extension Descriptor Syntax

The syntax required to define a CDS extension artifact.

The CDS extension mechanism allows you to add properties to existing artifact definitions without modifying the original source files. The content extension of the CDS extension descriptor must conform to the following format:

### Note

The following example of a CDS document for XS advanced is incomplete; it is intended for illustration purposes only.

### Sample Code

CDS Artifact Extension Syntax

```
namespace banking;
in package banking;
using CRM;
```

```
using CRM.Customer;
extend context CRM with {
    type BankingAccount {
        BIC : String(8);
        IBAN : String(120);
    };
};
extend entity Customer with {
    account: CRM.BankingAccount;
};
extend type Address with {
    email : String(60);
};
extend view MyView with {
    a,
    b as newB,
    ass[y=2].x as elemViaAssoc,
    sum(t) as aNewAggregate
};
```

## ⚠ Restriction

If a CDS artifact is defined in a package, it cannot be extended in the same package. In addition, the same CDS artifact cannot be extended twice in the **same** package.

## in package

Use the keywords “`in package`” to assign a CDS source document to a CDS extension package. This ensures that all the definitions in the CDS source files, both normal artifact definitions and extend statements, belong to the named package.

### Sample Code

CDS Artifact Extension Syntax

```
namespace banking;
in package banking;
```

### ℹ Note

The `in package` keyword must be inserted at the beginning of the CDS document: before or after the `namespace` declaration, if present, but always before all other statements.

## using

All artifacts that are to be extended in a CDS source document must be made available with the `using` declaration.

### Sample Code

```
namespace banking;
in package banking;
using CRM;
using CRM.Customer;
```

## Extending Elements in CDS Entities or Structured Types

You can use a CDS extension to add new elements to an existing CDS entity, as illustrated in the following example:

### Sample Code

Extend a CDS Entity with New Elements

```
extend entity MyEntity with {
  name1 : type1;
  name2 : type2;
  ...
};
```

The SAP HANA table generated for the CDS entity contains all specified extension elements. New elements can also be added to an existing structured type. It does not matter whether the original type is defined by means of the keyword “type” or “table type”. The following example shows how to extend a table type:

### Sample Code

Extend a CDS Structured Type

```
extend type MyType with {
  name1 : type1;
  name2 : type2;
  ...
};
```

### Note

For a “table type”, the generated SAP HANA table type contains all extension elements. This kind of extension does not work for scalar types.

## Extending SELECT Items in a CDS View

You can use a CDS extension to add new `SELECT` items to an existing CDS view, as illustrated in the following example:

### Sample Code

#### Extend a CDS View

```
extend view MyView with {
  a,
  b as newB,
  ass[y=2].x as elemViaAssoc,
  sum(t) as aNewAggregate
};
```

The SAP HANA view generated for the CDS view contains all extension items added to the `SELECT` clause.

### ⚠ Restriction

It is not possible to extend any part of a view definition other than the `SELECT` clause.

## Adding Artifacts to a CDS Context

You can use a CDS extension to add new artifacts (for example, tables, types, or views) to an existing CDS context, as illustrated in the following example:

### Sample Code

#### Extend CDS Context with New Artifacts

```
extend context MyContext with {
  type T1 : Integer;
  type S1 {
    a : INteger;
    b : String;
  };
  entity E1 {
    elem1 : Integer;
    elem2 : S1;
  };
  view V1 as select from E1 {, elem1, elem2 };
};
```

## Extending CDS Annotations

You can use a CDS extension to add new `@annotations` to an existing CDS artifact or element; the syntax you use to add the annotations differs according to whether you are adding them to a CDS artifact or an element, as illustrated in the following examples:

### Sample Code

Extending CDS Artifacts with Annotations

```
@MyIntegerAnnotation : 44  
extend entity MyEntity;
```

### Sample Code

Extending CDS Elements with Annotations

```
extend entity MyEntity with {  
    @MyIntegerAnnotation : 45  
    extend baseElement;  
};
```

### Sample Code

Extending CDS Entities with Annotated Elements

```
extend entity MyEntity with {  
    @MyIntegerAnnotation : 45  
    newElement : String(88);  
};
```

### Tip

Adding associations to elements of structured types and to `SELECT` items in views works in the same way.

## Extending a CDS Entity's Technical Configuration

You can use a CDS extension to add new elements to the `technical configuration` section of an existing CDS entity, as illustrated in the following example:

### Sample Code

Extending the CDS Entity's Technical Configuration

```
extend entity MyEntity with technical configuration {  
    partition by hash (baseElement) partitions 2;  
};
```

## ➔ Tip

You can use the same method to extend a CDS entity with additional indexes.

## Related Information

[The CDS Extension Descriptor \[page 321\]](#)

[Create a CDS Extension \[page 316\]](#)

[The CDS Extension Package Descriptor \[page 329\]](#)

[The CDS Extension Package Descriptor Syntax \[page 329\]](#)

### 5.3.7.3 The CDS Extension Package Descriptor

In the context of a CDS extension scenario, it is necessary to specify which `extend` statements and normal CDS artifact definitions belong to which package and, in addition, on which other packages a CDS extension package depends.

A CDS extension package is a normal CDS artifact; it has a name, and both its definition and its use can be found in the CDS Catalog. A package is defined by a special CDS source file named `.package.hdbc`. The syntax for defining a package and its dependencies is illustrated in the following example:

#### Sample Code

```
package <onlineBanking> depends on <banking>;
```

## Related Information

[The CDS Extension Package Descriptor Syntax \[page 329\]](#)

[Create a CDS Extension \[page 316\]](#)

### 5.3.7.4 The CDS Extension Package Descriptor Syntax

Required syntax for the CDS extension descriptor.

A CDS extension package is defined in a CDS extension package descriptor, which is a special CDS source file named `.package.hdbc`, as illustrated in the following example:

### Note

The leading dot (.) in the file name for the CDS extension package descriptor is mandatory.

### Sample Code

Extension Package Descriptor (.package.hdbcdfs)

```
package <onlineBanking> depends on <banking>;
```

In this example, the package `<onlineBanking>` depends on another CDS extension package `<banking>`, which contains extensions for a CDS base application `<myCDSapp>`.

The syntax for defining a package and its dependencies is illustrated in the following example:

### Sample Code

CDS Extension **Package** Descriptor Syntax

```
source = packageDefinition
packageDefinition = "package" packageName (
    "depends" "on" packageName ( "," packageName )* )? ";"*
packageName = identifier ( "." identifier )*
```

The name of the package defined in the file must be identical to the name space that is applicable for the file (as specified in the application's corresponding `.hdinamespace` file).

### Note

It is not necessary to explicitly define the base package for the CDS application; it contains all those CDS sources that are not explicitly assigned to a package.

## packageDefinition

The syntax for defining a dependency between CDS extension packages is illustrated in the following example:

### Sample Code

One-to-One Package Definition (.package.hdbcdfs)

```
package <onlineBanking> depends on <banking>;
```

A package can depend on multiple packages, as illustrated in the following example:

### Sample Code

One-to-Many Package Definition (.package.hdbcdfs)

```
package <onlineBanking> depends on (<banking>, <banking1>, <[...]>);
```

The following example illustrates the syntax required in a package-dependency statement specified in the CDS extension package descriptor (.package.hdbcdfs)

### Sample Code

#### CDS Extension **Package** Descriptor Syntax

```
source = packageDefinition
packageDefinition = "package" packageName (
    "depends" "on" packageName ( "," packageName )* )? ";"

packageName = identifier ( "." identifier )*
```

## packageName

The syntax required when specifying the name of a CDS extension package in a package-dependency definition is illustrated in the following example:

### Sample Code

#### One-to-One Package Definition (.package.hdbcdfs)

```
package <onlineBanking> depends on <banking>;
```

A package name can include a full stop (.), for example, to express a full path in a name space, as illustrated in the following example:

### Sample Code

#### One-to-Many Package Definition (.package.hdbcdfs)

```
package <src.onlineBanking> depends on (<src.banking>);
```

The following example illustrates the syntax required when defining the name of a package in the CDS extension package descriptor (.package.hdbcdfs)

### Sample Code

#### CDS Extension **Package** Descriptor Syntax

```
source = packageDefinition
packageDefinition = "package" packageName (
    "depends" "on" packageName ( "," packageName )* )? ";"

packageName = identifier ( "." identifier )*
```

## Related Information

[Create a CDS Extension \[page 316\]](#)

[The CDS Extension Descriptor \[page 321\]](#)

[The CDS Extension Descriptor Syntax \[page 324\]](#)

[The CDS Extension Package Descriptor \[page 329\]](#)

### 5.3.8 Create a CDS Access-Policy Document (XS Advanced)

As an XS advanced developer, you want to set instance-based authorizations for accessing data in the SAP HANA database.

#### Context

CDS access-policy documents are coded in the Data Control Language (DCL). CDS access policies have the same file extension as CDS documents. In a CDS access policy document, you can create CDS roles and CDS aspects for instance-based authorizations. It is also possible to use grant statements on CDS views and CDS roles and include the statements in the definition of the CDS roles. The grant statements determine which data set a user is authorized to access.

#### Procedure

1. Start the SAP HANA Web IDE for SAP HANA.

The SAP Web IDE for SAP HANA is available at the following default URL:

`https://<HANA_HOST>:53075`

➔ Tip

If the SAP Web IDE for SAP HANA is running on a non-default port, open a command shell, log on to the XS advanced run time, and run the following command:

```
xs app webide --urls
```

2. Open the application project to which you want to add your CDS access policy.
3. Open your multi-target application project, or if required, create a new project.
4. Define a new CDS access-policy document.

A CDS access-policy document is a file with a CDS file extension (`.hdbcds`). Enter a file name, for example `MyAccessPolicy` and choose *Finish*.

Define the CDS role in your CDS access policy. Save the CDS access policy file to your local project workspace.

## Caution

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

## Sample Code

```
namespace com.sap.dcl.example;
using com.sap.dcl.example::<DDL_file_name> as ddl;

AccessPolicy MyAccessPolicy {

    role salesOrderCountryUsa {
        grant select on ddl.salesOrderView
            where customerCountry = 'USA';
    };
}
```

5. Check the syntax of CDS sources.

Check the syntax of CDS access policies in the same way as you would check the syntax of CDS DDL sources.

6. Activate your CDS access-policy document in the same way as you would activate DDL sources.

After the activation of the CDS DCL source files (CDS access-policy document and source file with DDL code), the DCL artifacts you defined previously (for example, roles) are added to the SAP HANA database catalog.

- You can find the generated database role in the *Security* section of your SAP HANA database. In this example, the naming convention of the roles is as follows:

<project\_path.project>::<DCL\_source\_file>.<role\_name>  
com.sap.dcl.example::dcl.salesOrderCountryUsa

## Results

The database roles are also available in the SAP HANA Web IDE for SAP HANA.

## Note

This kind of database roles need to be assigned to the technical database user of the XS advanced application. To assign database roles to users, the XS advanced developer needs the HDI container's `default_access_role`. For more information about the default access role, see *Related Links*.

## Related Information

[Create a CDS Document \(XS Advanced\) \[page 208\]](#)

[The Default Access Role for HDI Containers \[page 192\]](#)

[Configure a Database Connection \[page 621\]](#)

## 5.3.8.1 CDS Access Policies in XS Advanced

As a developer in XS advanced, you can set instance-based authorizations for accessing data in the SAP HANA database. You define access policies using the CDS data control language (DCL). Write CDS access policy documents using DCL code where you define CDS roles. Instance-based authorizations are based on CDS views.

### Prerequisites

An XS advanced developer must have the standard developer authorization profile to assign roles to SAP HANA users.

#### ⚠ Caution

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

### CDS Documents for Access Policies Defined with Data Control Language

If you want to create access policies using DCL, you must define the policies in CDS documents. CDS documents are CDS source files with the suffix .hdbc<sub>ds</sub> which are located in the db folder of the HDI container. You use DCL code to define access-control logic for Core Data Services (CDS) views from the SAP HANA database. Definitions in DCL code enable you to filter access to data in the database based on static values, aspects, and based on external attributes.

#### ⚠ Caution

If you do **not** create and deploy CDS access policies for the CDS view, any user who can access the CDS view has access to all data returned by the CDS view.

CDS documents with DCL code or DDL code only differ in their top level element:

- DCL documents have `AccessPolicy` as top level element (whereas CDS documents with DDL code usually have `Context` as top level element).
- You can define CDS aspects and CDS roles within an `AccessPolicy` keyword section, as illustrated in the examples below.

When XS advanced developers compile an access-policy definition using a CDS document with DCL code, they generate corresponding database catalog artifacts, for example, roles.

#### Sample Code

```
namespace com.sap.dcl.example;
using com.sap.dcl.example::<DDL_file_name> as ddl;

AccessPolicy <DCL_file_name> {
```

```

role salesOrderCountryUsa {
    grant select on ddl.salesOrderView
        where customerCountry = 'USA';
};

}

```

In the CDS source file, the XS advanced developer must explicitly specify the structured privilege check for a CDS view, as illustrated in the following example.

### **Restriction**

In XS advanced, the `@schema` annotation cannot be used in either CDS documents with DCL code or CDS documents with DDL code.

### **Source Code**

```

entity salesOrder {
    key id
        : Integer;
    toCustomer
        : Association[0..1] to businessPartner;
    currencyCode
        : String(5);
    grossAmount
        : Integer;
    document
        : LargeBinary;
};

view salesOrderView as select from salesOrder
{
    key id as id,
    toCustomer.toAddress.country as customerCountry,
    currencyCode,
    grossAmount,
    document
} with structured privilege check;

```

### **Tip**

In XS advanced, it is recommended to use `key` fields in CDS views when creating instance-based authorization checks.

The keys in CDS views should be defined in a similar way to the key definition for DDL entities, for the following reasons:

- Enhanced system performance
- The field types `LargeBinary` and `LargeString` must only be used in CDS views with key fields so that instance-based authorizations are possible.

### **Note**

Only CDS **views** can be protected by a `grant` statement; CDS entities are not supported. You can use DCL for calculation views, but it is mandatory to include the calculation view in the CDS access policy, for example, by including a `using` statement at the start of the document. Only calculation views generated outside of CDS can be protected by a `grant` statement. A definition of calculation views is not supported by DDL.

## Related Information

[Create a CDS Access-Policy Document \(XS Advanced\) \[page 332\]](#)

[Create a CDS Aspect \(XS Advanced\) \[page 336\]](#)

[Create a CDS Role \(XS Advanced\) \[page 337\]](#)

### 5.3.9 Create a CDS Aspect (XS Advanced)

Use CDS aspects to create a CDS role that references external dynamic criteria.

#### Context

CDS aspects associate an attribute with permitted values of a user. The values are taken, for example, from a CDS entity or a CDS view, which is defined in a CDS DDL document. A `grant` statement in a role can use one or more “aspects” in a `where` clause.

#### ⚠ Caution

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

#### Procedure

1. Start the SAP HANA Web IDE for SAP HANA.

The SAP Web IDE for SAP HANA is available at the following default URL:

`https://<HANA_HOST>:53075/`

#### ➔ Tip

If the SAP Web IDE for SAP HANA is running on a non-default port, open a command shell, log on to the XS advanced run time, and run the following command:

```
xs app webide --urls
```

2. Open the application project to which you want to add your CDS aspect.
3. Open your multi-target application project, or if required, create a new project.
4. Open the CDS access-policy document.

Define the CDS aspects in your CDS access-policy document. Save the DCL definition of your CDS aspects to your local project workspace.

In this example, you want to enable users to access only the sales orders of their own country. For this, you need the country information so that you can use it in the respective role. You define a CDS aspect

that associates `country` from the `address` CDS entity with the corresponding employee's login name. This CDS aspect is then used by the corresponding CDS role.

### Sample Code

```
namespace com.sap.dcl.example;
using com.sap.dcl.example::<DDL_file_name> as ddl;

AccessPolicy <DCL_file_name> {

    aspect aspCountry as
        select from ddl.address { country }
        where $user in toEmployee.loginName;
    role salesOrderCountryOwn {
        // grant based on an aspect
        grant select on ddl.salesOrderView
            where customerCountry = aspect :aspCountry;
    };
}
```

5. Check the syntax of CDS sources.

Check the syntax of CDS access policies in the same way as you would check the syntax of CDS DDL sources.

6. Activate your CDS access policy document.

## Related Information

[Create a CDS Document \(XS Advanced\) \[page 208\]](#)

[Create a CDS Role \(XS Advanced\) \[page 337\]](#)

## 5.3.10 Create a CDS Role (XS Advanced)

Set instance-based authorizations for accessing data in the SAP HANA database.

## Context

CDS access-policy documents contain a set of CDS role definitions coded in the Data Control Language (DCL). You can use CDS roles to create instance-based authorizations. A role can contain CDS “aspects”, and it is also possible to use `grant` statements on CDS views and CDS roles and include the statements in the definition of the CDS roles. The database roles generated from the CDS role definition determine which data sets a user is authorized to access.

### Caution

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



## Example

If you provide authorization to access sales orders, you can add conditions that filter the view users have on the sales orders, for example, as follows:

- View the sales orders of all countries.
- View only the sales orders of one specific country, for example: the USA. The CDS role in the CDS access policy document contains a fixed value for "USA".
- View only the sales orders of one specific country. The CDS role in the CDS access policy document contains a CDS aspect associated with the country value of the CDS entity called `address`.
- View only the sales order of his or her country. Here, external user information in XS advanced provides the country attribute that is used in the CDS DCL definition.

## Procedure

1. Start the SAP HANA Web IDE for SAP HANA.

The SAP Web IDE for SAP HANA is available by default at the following default URL:

`https://<HANA_HOST>:53075/`



If the SAP Web IDE for SAP HANA is running on a non-default port, open a command shell, log on to the XS advanced run time, and run the following command:

`xs app webide --urls`

2. Open the application project to which you want to add your CDS role.
3. Open your multi-target application project, or if required, create a new project.
4. Open the CDS access policy document.

Define the CDS roles in your CDS access policy. Save the DCL definition of your CDS roles to your local project workspace.



## Sample Code

```
namespace com.sap.dcl.example;
using com.sap.dcl.example::<DDL_file_name> as ddl;

AccessPolicy <DCL_file_name> {
    role salesOrderCountryAll {
        // grant all countries
        grant select on ddl.salesOrderView;
    };
    aspect aspCountry as
        select from ddl.address { country }
        where $user in toEmployee.loginName;

    role salesOrderCountryUsa {
        // grant of a static value
        grant select on ddl.salesOrderView
            where customerCountry = 'USA';
    };
}
```

```

role salesOrderCountryOwnAndUsa {
    // grant based on an aspect
    grant select on ddl.salesOrderView
        where customerCountry = aspect :aspCountry;
    // grant based on an external attribute
    grant select on ddl.salesOrderView
        where customerCountry in $env.user.country;

    // grant of another role
    grant salesOrderCountryUsa;
};

};

```

5. Check the syntax of CDS sources.

Check the syntax of the CDS roles in the same way as you would check the syntax of CDS DDL sources.

6. Activate your CDS access policy document in the same way as you would activate DDL sources.

After the activation of the CDS access policy document with the CDS roles, the DCL artifacts you defined are added to the SAP HANA database catalog.

- After activation, the generated database roles are located in the *Security* section of your SAP HANA database. In this example, the naming convention of the roles is as follows:

```

<project_path>.<project>::<DCL_source_file>.<role_name>
com.sap.dcl.example::dcl.salesOrderCountryAll
com.sap.dcl.example::dcl.salesOrderCountryUsa
com.sap.dcl.example::dcl.salesOrderCountryOwnAndUsa

```

## Results

The database roles are also available in the SAP HANA Web IDE for SAP HANA.

**i Note**

Roles are assigned to the technical database user of the XS advanced application. To assign roles to users, you need the `default_access_role` for the HDI container. For information, see *Related Links*).

## Related Information

[Create a CDS Document \(XS Advanced\) \[page 208\]](#)

[The Default Access Role for HDI Containers \[page 192\]](#)

## 5.3.10.1 CDS Role Syntax Options

The options available for modeling instance-based authorizations in CDS roles.

### Overview

It is possible to combine grant statements in a role definition or use grant statements of multiple types. The following list shows the different ways in which grant statements can be used to define a CDS role:

- Using a grant on a CDS view with static values
- Using a grant on CDS aspects that associate a dynamic element of a CDS view (in a CDS DDL document)
- Using a grant on external attributes
- Using a grant of another role

CDS aspects can also be included in CDS role definitions and can be referenced in grants by using the following syntax:

```
aspect :<role_name>.<aspect_name>
```

The database roles generated from the CDS role definition determine which data set a user is authorized to access. If multiple roles are assigned to a user, the specified user has the authorizations defined in all of the assigned roles.

#### Caution

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

### CDS Role Using Grant with Static Values

#### Sample Code

```
grant select on ddl.salesOrderView  
    where customerCountry = 'USA';
```

In this CDS role, you use a `where` clause to directly refer to the `customerCountry` element defined in the CDS view called `salesOrderView`. The grant only allows a view on sales orders with the `customerCountry` value USA.

## CDS Role Using Grant with CDS Aspects

### Sample Code

```
grant select on ddl.salesOrderView  
    where customerCountry = aspect :aspCountry;
```

In this CDS role, you use the previously defined CDS aspect `aspCountry`. It uses the country element of the address information that comes with the CDS entity called `address`. Since users have different addresses, this data is variable and dynamic. The `customerCountry` element defined in the CDS view called `salesOrderView` is associated with the CDS aspect `aspCountry`. This CDS role definition allows users to view the sales orders of the `customerCountry` value provided by their address, that is; the sales orders of their own country.

## CDS Role Using Grant with External Attributes

### Sample Code

```
grant select on ddl.salesOrderView  
    where customerCountry in $env.user.country;
```

In this CDS role, you directly use external attributes that come with the user data of the XS advanced token for the database. The `customerCountry` element defined in the CDS view called `salesOrderView` is associated with the country attribute of the user's login data. Access is enabled to the external attribute using `$env.user.<external_attribute>`. This grant allows users to view the sales orders of the `customerCountry` value provided by the XS advanced token.

### Note

It is also possible to use XS advanced attributes that have multiple values.

## CDS Role Using Grant of Another Role

### Sample Code

```
grant salesOrderCountryUsa;
```

This grant includes the already defined CDS role `salesOrderCountryUsa`. Here, the CDS role `salesOrderCountryUsa` is already defined, for example, in the same CDS access-policy document. It is also possible to define CDS roles in separate CDS DCL files. This grant with an included CDS role allows users to also view the sales orders of the USA.

### Note

You can also use a `grant` of native SAP HANA roles, for example, defined in design-time artifacts with the file extension `.hdbrole`. However, you must introduce the native SAP HANA role by including a `using` declaration first. Only then is it possible to integrate the native role using the `grant` statement.

## Related Information

[The Application Security Descriptor \[page 764\]](#)

## 5.4 Maintaining JSON Collections in the SAP HANA Document Store

The SAP HANA Document Store (DocStore) is used to store collections which contain one or more JSON artifacts (documents).

The SAP HANA DocStore is a place where you can collect JSON documents, that is; files with content that is formatted according to the rules defined in the JavaScript Object Notation. The DocStore allows native operations on JSON documents, for example: filtering and aggregation, as well as joins with SAP HANA column- or row-store tables.

You can use standard SQL commands in an SQL console to maintain the document collections in the SAP HANA DocStore and, in addition, the JSON documents that make up the collection, and any values defined in the individual JSON documents.

In this section you can find information about creating and maintaining JSON documents in collections stored in the SAP HANA DocStore, for example:

- Create a SAP HANA Documentation Store.
- Add collections to (and remove them from) the DocStore.
- Maintain JSON documents in DocStore collections.

## Related Information

[Create a JSON Document Store \[page 343\]](#)

[Add and Remove JSON Collections in the Document Store \[page 345\]](#)

## 5.4.1 Create a JSON Document Store

Set up a store in SAP HANA for your JSON documents.

### Context

The SAP HANA Document Store (DocStore) is a place where you can collect JSON documents, that is; text files formatted according to the rules defined in the JavaScript Object Notation. The SAP HANA DocStore allows native operations on JSON documents, for example: filtering and aggregation, as well as joins with SAP HANA column- or row-store tables.

### Procedure

1. Check that the DocStore is enabled.

For multi-database instances, there can be 0 (none) or 1 (one) DocStore per tenant database; you can enable the DocStore in a database tenant by running the following command as administrator in the SQL console:

```
ALTER DATABASE <database> ADD 'docstore'
```

2. Create a JSON collection in the DocStore using SQL.

Open an SQL console and enter the following SQL command:

```
create collection Customers;
```

3. Create a JSON collection using a design-time artifact.

You can define a JSON collection in a design-time artifact with the suffix “`hdbcollection`”, for example, `AUTHORS.hdbcollection`. The file format uses a DDL-style syntax which is equivalent to the corresponding `CREATE COLLECTION` SQL syntax, but without the leading `CREATE` command, as illustrated in the following example:

#### Code Syntax

```
/src/AUTHORS.hdbcollection  
COLLECTION "AUTHORS"
```

The `hdbcollection` artifact must be located in the `db/` module of your XS advanced application.

Assuming that the corresponding HDI plug-in is available, the collection defined in the design-time artifact is activated in the catalog at application-deployment time.

#### Note

If the DocStore is not enabled and available at deployment time, the deployment operation fails and no collection object is created in the database catalog.

## Related Information

[Maintaining JSON Collections in the SAP HANA Document Store \[page 342\]](#)

[Add and Remove JSON Collections in the Document Store \[page 345\]](#)

[Document Store Collections \(.hdbcollection\) \[page 793\]](#)

### 5.4.1.1 The SAP HANA JSON Document Store

The SAP HANA Document Store (DocStore) contains JSON artifacts grouped in collections.

The SAP HANA DocStore is a feature that you can use to store JSON artifacts. The DocStore allows you to perform all standard operations on the JSON documents, for example, filtering, aggregation, and joining the JSON documents with HANA column or row store tables.

The Document Store is an optional feature of the SAP HANA database which you have to create for each tenant database. Only a single DocStore process per tenant is possible. To create the DocStore you can use the `ALTER DATABASE` statement, as illustrated in the following example:

```
ALTER DATABASE <database name> ADD 'docstore'
```

For a single database container, it is necessary to enable the DocStore by setting the `docstore` configuration parameter, as illustrated in the following SQL example:

```
ALTER SYSTEM ALTER CONFIGURATION('daemon.ini', 'SYSTEM')
SET ('docstore', 'instances') = '1'
WITH RECONFIGURE;
```

#### **Restriction**

Only a single DocStore per instance is allowed.

The DocStore does not have a predetermined SQL port; all communication is routed through the standard index servers. For a multi-database installation, port numbers are assigned according to the rules of multi-database installations. For single database containers port 3<`instanceNr`>60 is used as default.

## Related Information

[Create a JSON Document Store \[page 343\]](#)

[Maintaining JSON Collections in the SAP HANA Document Store \[page 342\]](#)

[JSON Collections \[page 347\]](#)

## 5.4.2 Add and Remove JSON Collections in the Document Store

Maintain new and existing collections of JSON documents in the SAP HANA DocStore.

### Context

You can use standard SQL commands in an SQL console to maintain the document collections in the SAP HANA DocStore and, in addition, the JSON documents that make up the collection. For example, you can add, update, and remove JSON documents in a collection. You can also maintain the values defined in the individual JSON documents.

### Procedure

1. Check that the DocStore is enabled.

For multi-database instances, there can be 0 (none) or 1 (one) DocStore per tenant database; you can enable the DocStore in a database tenant by running the following command as administrator in the SQL console:

```
ALTER DATABASE <database> ADD 'docstore'
```

2. Create a JSON collection in the DocStore.

To create a collection of JSON documents called “CUSTOMERS” run the following command as administrator in an SQL console:

```
CREATE COLLECTION CUSTOMERS;
```

3. Insert a JSON document into the collection.

Insert JSON-formatted information into the collection, as illustrated in the following example:

#### Sample Code

```
insert into Customers values({  
    "firstName": 'Kofi',  
    "familyName": 'Tetteh',  
    "address": {  
        "street": 'Nkrumah Close',  
        "number": '10',  
        "city": 'Accra'  
    },  
    "details": {  
        "age": 43  
    }  
});
```

### Sample Code

```
insert into Customers values({  
    "firstName": 'Kwame',  
    "familyName": 'Asumang',  
    "address": {  
        "street": 'Independence Avenue',  
        "number": '147',  
        "city": 'Accra'  
    },  
    "details": {  
        "age": 27  
    }  
});
```

4. Run a query on the content of a JSON document in the DocStore.

Search the JSON collection for information about specific addresses, as illustrated in the following example:

### Sample Code

```
select * from Customers where "address"."city" = 'Accra';
```

### Sample Code

```
select * from Customers where "details"."age" = 40;
```

5. Update individual JSON documents in a collection.

To update specified values defined in the JSON documents stored in the collection “CUSTOMERS”, use the `UPDATE` command as follows, which specifies what values to change (`SET`) in the JSON documents and “where”:

### Sample Code

```
UPDATE Customers SET { "name": 'Paul', "age": 50 } where "age" > 30;
```

The result of this change request in the JSON collection is as follows:

### Sample Code

```
{ "name": "Paul", "age": 50 }  
{ "name": "Kwame", "age": 27 }
```

6. Remove JSON documents from a collection.

To remove JSON documents stored in the collection “CUSTOMERS”, use the `DELETE FROM` command as follows, where all documents that match the condition defined in the `WHERE` clause are deleted:

### Sample Code

```
DELETE FROM CUSTOMERS WHERE <condition>;
```

To remove all documents from a JSON collection, use the following command, in which WHERE true is implied:

#### Sample Code

```
DELETE FROM CUSTOMERS;
```

7. Rename a JSON document collection.

To rename an existing JSON collection, use the following command:

#### Sample Code

```
RENAME COLLECTION <collection_name> TO <new_collection_name>;
```

## Related Information

[Create a JSON Document Store \[page 343\]](#)

[Maintaining JSON Collections in the SAP HANA Document Store \[page 342\]](#)

### 5.4.2.1 JSON Collections

JSON documents are grouped together as “collections” that are stored in the SAP HANA Documentation Store (DocStore).

Data defined in documents formatted according to the JavaScript Object Notation (JSON) is stored in JSON artifacts that can be grouped into so-called “collections” and stored in the SAP HANA DocStore. The content of a JSON document may be deeply structured, however, unlike XML, a JSON document does not have a schema. This means that any valid JSON data may be inserted without first declaring its structure.

Document collections and the JSON content can be maintained with SQL in the same way as data stored in tables. For example, you can insert data with the `INSERT` statement and query the document data with the `SELECT` command. You can also use a single query statement to read data from tables and collections at the same time, and you can combine standard tables and JSON collections by joining them in the same way as the `JOIN` between any other column- or row-store tables.

Technically, collections are stored in a dedicated binary format; in SQL however, they are defined as tables with their own sub-type, which you can see by selecting collections on the basis of the table type '`COLLECTION`', as illustrated in the following example:

```
SELECT * FROM tables WHERE table_type = 'COLLECTION'
```

The following example is included here to illustrate how a collection of customer details can be created, updated and read. The `INSERT` statement illustrates the syntax and structure of nested JSON data and the

corresponding SELECT statements show the use of path expressions to navigate the content of the JSON document:

```
create collection Customers;

insert into Customers values({
    "name": 'Paul',
    "address": {
        "street": 'Main Street 10',
        "city": 'Heidelberg'
    }
});

select * from Customers where "address"."city" = 'Heidelberg';

select "address"."city", count(*) from Customers group by "address"."city";
```

## JSON Documents

The content of a JSON document may be deeply structured, however, unlike XML, a JSON document does not have a schema. This means that any valid JSON data may be inserted without first declaring its structure.

### Sample Code

#### JSON Nested Content

```
{
    "author": {
        "address": {
            "street": "Main Street"
            "number": 101
        },
        ...
    },
    ...
}
```

To access the information in the nested structure defined in the JSON document, you can use the “dot notation” to define the path to the required value, for example: "author"."address"."street"

In a JSON document, it is also possible to define elements in an array, such as the one illustrated in the following simple example:

### Sample Code

#### JSON Nested Content

```
{
    "author": {
        "addresses": [
            {
                "street": "First Street"
            },
            {
                "street": "Second Street"
            }
        ],
        ...
    }
},
```

```
    ...  
}
```

To enable access to elements defined in an array, a standard index operator [ # ] is required. The index positions start at "1", which means the first element has the position 1. So, to access the value defined in the **second** address in the example above ("Second Street"), you would use something like the following example: "author"."addresses"[2].**street**"

➔ **Tip**

This notation can be used anywhere where it is necessary to access a value, for example, "Sort", "Filter", "Projection". If no value exists in the document for a given identifier a NULL value is returned instead.

## Related Information

[Document Store Collections \(.hdbcollection\) \[page 793\]](#)

[The SAP HANA JSON Document Store \[page 344\]](#)

[Add and Remove JSON Collections in the Document Store \[page 345\]](#)

### 5.4.3 Import JSON Documents into a DocStore Collection

Import JSON data from a CSV file into a collection in the SAP HANA DocStore.

## Prerequisites

The success of the import operation depends on the following prerequisites:

- The SAP HANA DocStore is enabled
- The target document collection specified in the data-import definition is available in the DocStore, for example, CUSTOMERS.

➔ **Tip**

You can define the collection in the SQL console or in an HDI design-time artifact, for example, CUSTOMERS.hdbcollection.

## Context

You can use the HDI Table Data plug-in (`hdbtabledata`) to import one or more JSON documents stored in a CSV file; multiple JSON documents are separated by a line-break character, for example, \n.

## Caution

Although JSON documents do not have a schema, it is not recommended to use different data types for the same identifier in different JSON documents. This could cause inconsistent results returned from queries.

## Procedure

1. Create the source file containing the data to import into your collection in the SAP HANA DocStore.

The source data (JSON documents) must be stored in a comma-separated values (CSV) file, which must be placed in your XS advanced application's database module. Since the CSV file must only contain **one** column, no comma is required, as there is only one value. In the CSV file, each line represents a single JSON document, and the line-break character is used to define the end of one JSON document and the start of the next, as illustrated in the following example:

### Tip

The line-break is automatically added by your text editor; it is not necessary to add the line-break character to the CSV file manually.

### Sample Code

```
sap::myJSONData.csv

{"name": "Peter", "address": {"street": "Main Street 10", "city": "Heidelberg"} } \n
{"name": "Kwame", "address": {"street": "High Street 15", "city": "Accra"} } \n
{"name": "Nelson", "address": {"street": "Low Street 101", "city": "Johannesburg"} }
```

2. Create the data-import definition file.

In SAP HANA XS advanced, you can configure the import operation in a so-called Table Data definition - a design-time artifact with the hdbtabledata suffix, for example, `myJSONdata`. This file must also be located in the XS advanced application's database module.

- a. Specify the target column mapping for the data import.

Since the CSV file must only contain **one** column, the mapping should look as follows:

### Sample Code

```
"column_mappings": { "MY_DocStore_COLLECTION": 1 },
```

- b. Specify the type of table into which you want to import the JSON data.

It is only possible to import JSON data into a "table" of type "COLLECTION". For this reason, in the table-data definition, use the "`is_collection_table`" parameter (default is `false`) to define the target table "type" as illustrated in the following example.

### Sample Code

```
is_collection_table": true,
```

- c. Specify details of the source data to be imported.

You will need to specify the "data\_type": (CSV) of the source file itself; the format of the data in the CSV file ("dialect": "HANA\_JSON"); the name of the source CSV file ("file\_name": ), and indicate that the CSV file does not contain any header information ("has\_header": false).

- d. Lastly, it is mandatory to specify the name of the collection ("target\_table": ) into which the JSON documents will be inserted.

The completed definition file for your JSON data import operation should look like the following example:

### Restriction

It is not allowed to use multiple data sources to fill a collection table, for example, by using the `include_filter` parameter in the "import\_settings" section of the table-data definition file.

It is only possible to import data into one, single collection from one, single `hdbtabledata` file, and the import operation always imports **all** the content included in the corresponding CSV file.

### Sample Code

`hdbtabledata` File for JSON data import

```
{  
    "format_version": 1,  
    "imports": [  
        {  
            "column_mappings": {  
                "CUSTOMERS": 1  
            },  
            "is_collection_table": true,  
            "source_data": {  
                "data_type": "CSV",  
                "dialect": "HANA-JSON",  
                "file_name": "sap::myJSONcustomerData.csv",  
                "has_header": false  
            },  
            "target_table": "CUSTOMERS"  
        }  
    ]  
}
```

3. Start the import operation.

To start the import of the JSON documents into the specified collection in the SAP HANA DocStore, build the XS advanced application's database module which contains the CSV and table-data definition artifacts. For example, in the project navigation pane in SAP Web IDE for SAP HANA, right-click the `hdb` module and choose *Build* in the context menu.

### Note

Importing data into a collection table fails if the parameter "is\_collection\_table" : false is set (or not set at all).

## Related Information

[Document Store Collections \(.hdbcollection\) \[page 793\]](#)

[Table Data \(.hdbtabledata\) \[page 825\]](#)

[Maintaining JSON Collections in the SAP HANA Document Store \[page 342\]](#)

## 5.5 Creating Procedures and Functions in XS Advanced

Database procedures and functions can be used to help manage the underlying data model.

As part of the process of defining the database persistence model for your application, you create database design-time artifacts such as tables and views, for example using Core Data Services (CDS). However, you can also create procedures and table or scalar functions, for example, using SQLScript; procedures can be used to insert data into (and remove data from) tables or views.

In XS advanced, each database artifact has a distinct file extension, for example,

`myProcedure.hdbprocedure`, `mySynonym.hdbsynonym`, `myScalarFunction.hdbfunction`; the file extension determines which plug-in is used to generate the corresponding catalog objects.

### Tip

Scalar and table functions have the same file extension `.hdbfunction`. The HDI uses the same plugin to generate the catalog object.

For easier navigation and maintenance, it is recommended to place your procedures and functions in a separate folder in the application's database (`db/`) module, as illustrated in the following example:

### Sample Code

```
<MyAppName>
|- db/
|  |- package.json
|  |- src/
|  |  |- .hdiconfig
|  |  |- .hdinamespace
|  |  |- data/
|  |  |  |- myEntity.hdbcds
|  |  |  |- myDataType.hdbcds
|  |  |  |- myDoc.hdbcds
|  |  |- procedures
|  |  |  |- myProc.hdbprocedure
|  |  |  \- myFunc.hdbfunction
|- web/
|  |- xs-app.json
|  \- resources/
|- js/
|  |- start.js
|  |- package.json
|  \- src/
|- security/
|  \- xs-security.json
\- mtad.yaml
```

## Related Information

- [Create a Database Procedure in XS Advanced \[page 363\]](#)
- [Create a Database Function in XS advanced \[page 366\]](#)
- [HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)
- [Creating the Data Persistence Artifacts in XS Advanced \[page 199\]](#)

### 5.5.1 Create the Underlying Data Artifacts for a Procedures

Create the data objects that you plan to use and modify with your functions and procedures.

#### Prerequisites

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

#### ⚠ Caution

The code examples included in this document for XS advanced are sometimes either syntactically incomplete or include additional definitions that you might not always need; as a general rule, code examples are intended for illustration purposes only.

#### Context

Database procedures and functions are used to automate actions that need to be regularly performed on underlying data objects, such as tables or views. In this task, we show you how to create some tables, views, and data types that can then be used by the database procedures and functions you create in subsequent tasks. You also create a sequence to enable you to add items to a table in the correct order, and upload some initial data into the new tables you create.

#### Procedure

1. Start the SAP Web IDE for SAP HANA.
2. Display the application project to which you want to add a database procedure.

An application is created within a context of a project. If you do not already have a project, there are a number of ways to create one, for example: by importing it, cloning it, or creating a new one from scratch.

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

Currently, there is only one type of project template available, namely: *Multi-Target Application Project*. Select *Multi-Target Application Project* and choose *Next*.

- c. Type the name **myworld** for the new MTA project and choose *Next* to confirm.
- d. Specify details of the new MTA project and choose *Next* to confirm.
- e. Create the new MTA project; choose *Finish*.

### 3. Create a new application folder for your data artifacts.

In the `db/` module of your application project, create a folder named `data` in the `src/` folder, for example, `myworld/db/src/data`. You use this folder to store your tables, types, views, sequences, and so on.

### 4. Create a CDS document to define your data model

In this step we create a Core Data Services (CDS) document to define a couple of simple database tables: "header" and "item" for purchase orders. The procedure we define later is used to run queries on (and add content to) these tables.

#### Tip

If you do not already have a database module, right-click the root folder of your new MTA project and, in the context menu, choose **New** **HDB Module**. Name the new database model **db**.

- a. Right-click the folder `myworld/db/src/data` and choose **New** **CDS Artifact** in the context menu.
- b. Name the new CDS artifact **PurchaseOrder**.

This CDS document will be used to define multiple database objects, for example:

- o Two tables ("Header" and "Item")
- o Some data types ("BusinessKey", "SDate", "CurrencyT", "AmountT", "QuantityT", "UnitT", and "StatusT")
- o A reusable structure "HistoryT"
- o An SQL view "ItemView"

The setup Wizard adds the mandatory suffix for CDS artifacts (`.hdbcards`) to the new file name automatically.

### 5. Define the name space and context in which the entities "Header" and "Item" will be created.

As a first step, add the CDS code for the name space and context.

#### Caution

The top-level element of a CDS document (in this case the context "PurchaseOrder") **must** match the name of the design-time artifact, for example, `PurchaseOrder.hdbcards`.

### Sample Code

purchaseOrder..hdbcds

```
namespace myworld.db.data;
context PurchaseOrder {
};
```

6. Add some data types to the CDS document.

Enter the CDS code for a collection of data types (for example, `sdate`, `CurrencyT`, etc) to the CDS document; the types are referenced by the entities “Header” and “Item”, which you add in a subsequent step.

### Sample Code

```
namespace myworld.db.data;
context PurchaseOrder {
    type BusinessKey : String(10);
        type SDate : LocalDate;
        type CurrencyT : String(5);
        type AmountT : Decimal(15,2);
        type QuantityT : Decimal(13,3);
        type UnitT: String(3);
        type StatusT: String(1);
};
```

7. Add a reusable structure to the CDS document.

Enter the CDS code for a reusable structure “`HistoryT`” to the CDS document; the elements in the reusable structure are referenced by the entities “Header” and “Item”, which you add in a subsequent step.

### Sample Code

```
namespace myworld.db.data;
context PurchaseOrder {
    type BusinessKey : String(10);
        type SDate : LocalDate;
        type CurrencyT : String(5);
        type AmountT : Decimal(15,2);
        type QuantityT : Decimal(13,3);
        type UnitT: String(3);
        type StatusT: String(1);

    Type HistoryT {
        CREATEDBY : BusinessKey;
        CREATEDAT : SDate;
        CHANGEDBY : BusinessKey;
        CHANGEDAT : SDate;
    };
};
```

8. Add some simple database tables to the database module.

Enter the CDS code for the simple entities “Header” and “Item”, as illustrated in the following example.

## Sample Code

```
namespace myworld.db.data;
context PurchaseOrder {
    type BusinessKey : String(10);
    type SDate : LocalDate;
    type CurrencyT : String(5);
    type AmountT : Decimal(15,2);
    type QuantityT : Decimal(13,3);
    type UnitT: String(3);
    type StatusT: String(1);

    Type HistoryT {
        CREATEDBY : BusinessKey;
        CREATEDAT : SDate;
        CHANGEDBY : BusinessKey;
        CHANGEDAT : SDate;
    };
    Entity Header {
        key PURCHASEORDERID: BusinessKey;
        ITEMS: Association[*] to Item on ITEMS.PURCHASEORDERID =
PURCHASEORDERID;
        HISTORY: HistoryT;
        NOTEID: BusinessKey null;
        PARTNER: BusinessKey;
        CURRENCY: CurrencyT;
        GROSSAMOUNT: AmountT;
        NETAMOUNT: AmountT;
        TAXAMOUNT: AmountT;
        LIFECYCLESTATUS: StatusT;
        APPROVALSTATUS: StatusT;
        CONFIRMSTATUS: StatusT;
        ORDERINGSTATUS: StatusT;
        INVOICINGSTATUS: StatusT;
    } technical configuration {
        column store;
    };
    Entity Item {
        key PURCHASEORDERID: BusinessKey;
        key PURCHASEORDERITEM: BusinessKey;
        HEADER: Association[1] to Header on HEADER.PURCHASEORDERID =
PURCHASEORDERID;
        PRODUCT: BusinessKey;
        NOTEID: BusinessKey null;
        CURRENCY: CurrencyT;
        GROSSAMOUNT: AmountT;
        NETAMOUNT: AmountT;
        TAXAMOUNT: AmountT;
        QUANTITY: QuantityT;
        QUANTITYUNIT: UnitT;
        DELIVERYDATE: SDate;
    } technical configuration {
        column store;
    };
};
```

9. Add an SQL view to the database model.

Enter the CDS code for the simple view "ItemView", as illustrated in the following example.

## Sample Code

```
namespace myworld.db.data;
context PurchaseOrder {
```

```

type BusinessKey : String(10);
type SDate : LocalDate;
type CurrencyT : String(5);
type AmountT : Decimal(15,2);
type QuantityT : Decimal(13,3);
type UnitT: String(3);
type StatusT: String(1);

Type HistoryT {
    CREATEDBY : BusinessKey;
    CREATEDAT : SDate;
    CHANGEDBY : BusinessKey;
    CHANGEDAT : SDate;
};

Entity Header {
    key PURCHASEORDERID: BusinessKey;
    ITEMS: Association[*] to Item on ITEMS.PURCHASEORDERID =
PURCHASEORDERID;
    HISTORY: HistoryT;
    NOTEID: BusinessKey null;
    PARTNER: BusinessKey;
    CURRENCY: CurrencyT;
    GROSSAMOUNT: AmountT;
    NETAMOUNT: AmountT;
    TAXAMOUNT: AmountT;
    LIFECYCLESTATUS: StatusT;
    APPROVALSTATUS: StatusT;
    CONFIRMSTATUS: StatusT;
    ORDERINGSTATUS: StatusT;
    INVOICINGSTATUS: StatusT;
} technical configuration {
    column store;
};

Entity Item {
    key PURCHASEORDERID: BusinessKey;
    key PURCHASEORDERITEM: BusinessKey;
    HEADER: Association[1] to Header on HEADER.PURCHASEORDERID =
PURCHASEORDERID;
    PRODUCT: BusinessKey;
    NOTEID: BusinessKey null;
    CURRENCY: CurrencyT;
    GROSSAMOUNT: AmountT;
    NETAMOUNT: AmountT;
    TAXAMOUNT: AmountT;
    QUANTITY: QuantityT;
    QUANTITYUNIT: UnitT;
    DELIVERYDATE: SDate;
} technical configuration {
    column store;
};

define view ItemView as SELECT from Item {
    PURCHASEORDERID as "PurchaseOrderItemId",
    PURCHASEORDERITEM as "ItemPos",
    HEADER.PARTNER as "PartnerId",
    PRODUCT as "ProductID",
    CURRENCY as "CurrencyCode",
    GROSSAMOUNT as "Amount",
    NETAMOUNT as "NetAmount",
    TAXAMOUNT as "TaxAmount",
    QUANTITY as "Quantity",
    QUANTITYUNIT as "QuantityUnit",
    DELIVERYDATE as "DeliveryDate1"
} with structured privilege check;

};

```

10. Create a new file to define a database sequence.

The tables “Header” and “Item” use a unique order id number as the primary key. As a result, you need a sequence to have an auto incrementing unique id generated for each time new data is inserted into the tables.

- a. Right-click the folder `myworld/db/src/data` and choose in the context menu.
- b. Name the new file artifact `orderId.hdbsequence`.

Enter the following code to create a non-cycling sequence within your schema starting from 200000000 and ending with 299999999. Make the sequence dependent upon the header table with the full package id on the front of the header table name.

#### Sample Code

```
SEQUENCE "myworld.db.data::orderId"
INCREMENT BY 1
START WITH 200000000
MINVALUE 1
MAXVALUE 299999999
NO CYCLE
RESET BY SELECT IFNULL(MAX(PURCHASEORDERID), 0)+1
FROM "myworld.db.data::PurchaseOrder.Header"
```

- c. Save the sequence file.
11. Load some initial data into the new tables `Header` and `Item`.

In HDI, you can import data from a comma-separated-value file (CSV) into a database table; you need the following design-time artifacts:

- CSV file (`myData.csv`)  
Holds the data you want to load into the database table.
- Table definition file (`myTargetTable.hdbtabledata`)  
Specifies the target table for the data in the source `.csv` file to be uploaded.

- a. Create the `.hdbtabledata` file.

In the database module's data/ folder, create a file called `Purchase.hdbtabledata` and add the following code:

#### Sample Code

`Purchase.hdbtabledata`

```
{
"format_version": 1,
"imports": [
"target_table": "myworld.db.data::PurchaseOrder.Header",
"source_data": {
"data_type": "CSV",
"file_name": "myworld.db.data::header.csv",
"has_header": false,
"dialect": "HANA",
"type_config": {
"delimiter": ","
}
},
"import_settings": {
"import_columns": [
"PURCHASEORDERID",
"NOTEID",
"PARTNER",
]
```

```

        "CURRENCY",
        "GROSSAMOUNT",
        "NETAMOUNT",
        "TAXAMOUNT",
        "LIFECYCLESTATUS",
        "APPROVALSTATUS",
        "CONFIRMSTATUS",
        "ORDERINGSTATUS",
        "INVOICINGSTATUS"]
    },
    "column_mappings": {
        "PURCHASEORDERID": 1,
        "NOTEID": 6,
        "PARTNER": 7,
        "CURRENCY": 8,
        "GROSSAMOUNT": 9,
        "NETAMOUNT": 10,
        "TAXAMOUNT": 11,
        "LIFECYCLESTATUS": 12,
        "APPROVALSTATUS": 13,
        "CONFIRMSTATUS": 14,
        "ORDERINGSTATUS": 15,
        "INVOICINGSTATUS": 16
    }
},
{
    "target_table": "myworld.db.data::PurchaseOrder.Item",
    "source_data": {
        "data_type": "CSV",
        "file_name": "myworld.db.data::item.csv",
        "has_header": false,
        "dialect": "HANA",
        "type_config": {
            "delimiter": ","
        }
    },
    "import_settings": {
        "import_columns": [
            "PURCHASEORDERID",
                "PURCHASEORDERITEM",
                "PRODUCT",
                "NOTEID",
                "CURRENCY",
                "GROSSAMOUNT",
                "NETAMOUNT",
                "TAXAMOUNT",
                "QUANTITY",
                "QUANTITYUNIT"
            ],
            "column_mappings": {
                "PURCHASEORDERID": 1,
                "PURCHASEORDERITEM": 2,
                "PRODUCT": 3,
                "NOTEID": 4,
                "CURRENCY": 5,
                "GROSSAMOUNT": 6,
                "NETAMOUNT": 7,
                "TAXAMOUNT": 8,
                "QUANTITY": 9,
                "QUANTITYUNIT": 10
            }
        ]
    }
}

```

- b. Create the header.csv file.

In the database module's data/ folder, create a file called `header.csv` and add the following content and save the file:

#### Sample Code

`header.csv`

```
5000000000,0000000033,20120101,0000000033,20120101,9000000001,0100000000,  
EUR,13224.47,11113,2111.47,N,I,I,I,I  
0500000001,0000000033,20120102,0000000033,20120102,9000000001,0100000002,  
EUR,12493.73,10498.94,1994.79,N,I,I,I,I
```

- c. Create the `item.csv` file.

In the database module's data/ folder, create a file called `item.csv` and add the following content and save the file:

#### Sample Code

`item.csv`

```
5000000000,0000000010,HT-1000,,EUR,1137.64,956,181.64,1,EA,20121204  
5000000000,0000000020,HT-1091,,EUR,61.88,52,9.88,2,EA,20121204  
5000000000,0000000030,HT-6100,,EUR,1116.22,938,178.22,2,EA,20121204  
5000000000,0000000040,HT-1000,,EUR,2275.28,1912,363.28,2,EA,20121204  
5000000000,0000000050,HT-1091,,EUR,92.82,78,14.82,3,EA,20121204  
5000000000,0000000060,HT-6100,,EUR,1116.22,938,178.22,2,EA,20121204  
5000000000,0000000070,HT-1000,,EUR,2275.28,1912,363.28,2,EA,20121204  
5000000000,0000000080,HT-1091,,EUR,61.88,52,9.88,2,EA,20121204  
5000000000,0000000090,HT-6100,,EUR,1674.33,1407,267.33,3,EA,20121204  
5000000000,0000000100,HT-1000,,EUR,3412.92,2868,544.92,3,EA,20121204  
5000000001,0000000010,HT-1100,,USD,213.96,179.8,34.16,2,EA,20121204  
5000000001,0000000020,HT-2026,,USD,35.69,29.99,5.7,1,EA,20121204  
5000000001,0000000030,HT-1002,,USD,3736.6,3140,596.6,2,EA,20121204  
5000000001,0000000040,HT-1100,,USD,213.96,179.8,34.16,2,EA,20121204  
5000000001,0000000050,HT-2026,,USD,71.38,59.98,11.4,2,EA,20121204  
5000000001,0000000060,HT-1002,,USD,3736.6,3140,596.6,2,EA,20121204  
5000000001,0000000070,HT-1100,,USD,320.94,269.7,51.24,3,EA,20121204  
5000000001,0000000080,HT-2026,,USD,107.06,89.97,17.09,3,EA,20121204  
5000000001,0000000090,HT-1002,,USD,3736.6,3140,596.6,2,EA,20121204  
5000000001,0000000100,HT-1100,,USD,320.94,269.7,51.24,3,EA,20121204
```

12. Create a new database synonym.

A database synonym is required to enable access to any table or view outside of our application's container. In this step, to enable us to access the DUMMY view, you create **two** files: an synonym artifact (`.hdbsynonym`).

In the database module's data/ folder, create a file called `general.hdbsynonym`, add the following content, and save the file:

#### Sample Code

`general.hdbsynonym`

```
{  
    "myworld.db.data::DUMMY" : {  
        "target" : {  
            "schema" : "SYS",  
            "object" : "DUMMY"  
        }  
    }  
}
```

```
    }
}
```

### 13. Create a new structured privilege.

The structured privilege is the logical successor to the analytic privilege; it enables us to perform instance filtering for the CDS view `ItemView` that we created earlier.

- In the database module's `data` folder, create a new subfolder named `roles`, for example, `db/data/roles`.

In the database module's `data/roles` folder, create a file called

`PurchaseOrder.hdbstructuredprivilege`, add the following content, and save the file:

#### Sample Code

`PurchaseOrder.hdbstructuredprivilege`

```
STRUCTURED PRIVILEGE
  "myworld.db.roles::PO_VIEW_PRIVILEGE"
  FOR SELECT ON
    "myworld.db.data::PurchaseOrder.ItemView"
  WHERE "CurrencyCode" = 'EUR'
```

### 14. Create a new role for the structured privilege.

The technical user bound to the application's HDI container enables access to the HDI database objects from XS advanced. If you want to enable access by other database users (for example, external reporting tools) you must create a database role.

In the database module's `data/roles` folder, create a file called `dev007.hdbrole`, add the following content, and save the file:

#### Sample Code

`dev007.hdbrole`

```
{
  "role": {
    "name": "myworld.db.roles::dev007",
    "object_privileges": [
      {
        "name": "myworld.db.data::PurchaseOrder.Header",
        "type": "TABLE",
        "privileges": [ "SELECT" ]
      },
      {
        "name": "myworld.db.data::PurchaseOrder.Item",
        "type": "TABLE",
        "privileges": [ "SELECT" ]
      },
      {
        "name": "myworld.db.procedures::getPOItems",
        "type": "PROCEDURE",
        "privileges": [ "EXECUTE" ]
      },
      {
        "name": "myworld.db.data::PurchaseOrder.ItemView",
        "type": "VIEW",
        "privileges": [ "SELECT" ]
      }
    ]
  }
}
```

```

        ],
        "schema_analytic_privileges": [
            {
                "privileges": [ "myworld.db.roles::PO_VIEW_PRIVILEGE" ]
            }
        ]
    }
}

```

15. Check that the required plug-ins are available to build and deploy the database objects defined in the CDS document.

In SAP HANA XS advanced, a file suffix (for example, .hdbcds) is linked to a specific plug-in, which is used to generate the corresponding catalog object from a design-time artifact.

The container-configuration file db/src/.hdiconfig must contain (at least) the following entries:

### Note

The version number of the plug-in can differ according to the installed version of SAP HANA.

### Sample Code

db/src/.hdiconfig

```

"hdbcds": {
    "plugin_name": "com.sap.hana.di.cds",
    "plugin_version": "12.1.0"
},
"hdbssequence": {
    "plugin_name": "com.sap.hana.di.sequence",
    "plugin_version": "12.1.0"
},
"hdbtabledata" : {
    "plugin_name" : "com.sap.hana.di.tabledata",
    "plugin_version": "12.1.0"
},
"csv" : {
    "plugin_name" : "com.sap.hana.di.tabledata.source",
    "plugin_version": "12.1.0"
},
"hdbssynonym" : {
    "plugin_name" : "com.sap.hana.di.hdbsynonym",
    "plugin_version": "12.1.0"
},
"hdbsstructuredprivilege" : {
    "plugin_name" : "com.sap.hana.di.structuredprivilege",
    "plugin_version": "12.1.0"
},
"hdbrole" : {
    "plugin_name" : "com.sap.hana.di.role",
    "plugin_version": "12.1.0"
},
"hdbroleconfig" : {
    "plugin_name" : "com.sap.hana.di.role.config",
    "plugin_version": "12.1.0"
}

```

### ➔ Tip

By default, the `.hdiconfig` configuration file is hidden. To display it, choose **View > Show Hidden Files** in the menu bar.

#### 16. Build the HDB module `db`.

Building a database module activates the data model and creates the corresponding objects in the database catalog for each artifact defined in the CDS document. In this case, the build creates one database two tables named “Header” and “Item”, an SQL view named “ItemView”.

- a. In the SAP Web IDE for SAP HANA, choose **Build > Build**.

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

## Related Information

[Create a Database Procedure in XS Advanced \[page 363\]](#)

[Create a Database Function in XS advanced \[page 366\]](#)

## 5.5.2 Create a Database Procedure in XS Advanced

Create, edit, and deploy procedures.

## Prerequisites

- You have access to a running SAP HANA system where the XS advanced run time is installed and configured
- You have access to the developer tools included in SAP Web IDE for SAP HANA  
If SAP Web IDE for SAP HANA is not available at the default URL `https://host.acme.com:53075`, log in to the XS advanced run time and use the command `xs app webide` to display the URL required to access the tool.
- You have created the database tables `Header` and `Item` described in *Create the Underlying Data Artifacts for a Procedure*. For more information, see *Related Links* below.

### ⚠ Caution

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

## Context

In XS advanced a database procedure has the file extension .hdbprocedure, for example, `myProcedure.hdbprocedure`. To create and deploy a procedure in XS advanced using the SAP Web IDE for SAP HANA, perform the following steps:

## Procedure

1. Start the SAP Web IDE for SAP HANA.
2. Display the application project to which you want to add a database procedure.

An application is created within a context of a project. If you do not already have a project, there are a number of ways to create one, for example: by importing it, cloning it, or creating a new one from scratch.

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

Currently, there is only one type of project template available, namely: *Multi-Target Application Project*. Select *Multi-Target Application Project* and choose *Next*.

- c. Type the name `myworld` for the new MTA project and choose *Next* to confirm.
- d. Specify details of the new MTA project and choose *Next* to confirm.
- e. Create the new MTA project; choose *Finish*.

3. Create a stored procedure that enables you to add content to a database table .

The procedure you create in this step uses a `SELECT INNER JOIN` to retrieve data from two database tables, `Header` and `Item` (created in a previous task), filter the data by partner role, and join the results into the output parameter.

- a. Create a new folder for your stored procedures.

In the `db/` module create a folder named `procedures` in the `src/` folder, for example, `myworld/db/src/procedures`.

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

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

The new stored procedure opens in the SAP Web IDE for SAP HANA.

- c. Add the following SQLScript code to the procedure file `get_bp_addresses_by_role.hdbprocedure` and save the file.

### Sample Code

```
PROCEDURE "myworld.db.procedures::get_bp_addresses_by_role" (
    in im_partnerrole nvarchar(3),
    out ex_bp_addresses "myworld.db.data::Procedures.tt_bp_addresses"
)
LANGUAGE SQLSCRIPT
SQL SECURITY INVOKER
--DEFAULT SCHEMA <default_schema_name>
READS SQL DATA AS
```

```

BEGIN
/*
***** Write your procedure logic *****
*/
ex_bp_addresses =
    select a."PARTNERID", a."PARTNERROLE",
           a."EMAILADDRESS", a."COMPANYNAME",
           a."ADDRESSES.ADDRESSID" as "ADDRESSID",
           b."CITY", b."POSTALCODE", b."STREET"
    from "myworld.db.data::MD.BusinessPartner" as a
   inner join "myworld.db.data::MD.Addresses" as b
      on a."ADDRESSES.ADDRESSID" = b."ADDRESSID"
   where a."PARTNERROLE" = :im_partnerrole;
END

```

- Build the application database module db.

In the SAP Web IDE for SAP HANA, right-click the application module *myworld/db*, choose ► **Build** ► **Build** in the context menu, and follow the build progress in the output displayed in the console pane.

### Output Code

```
2:10:51 PM (Builder) Build of /myworld/db completed successfully.
```

- Check that the required plug-in is available to build and deploy the stored procedure.

In SAP HANA XS advanced, a file suffix (for example, .hdbprocedure) is linked to a specific plug-in, which is used to generate the corresponding catalog object from a design-time artifact.

The container-configuration file *.hdiconfig* must contain the following entry (version numbers can differ):

### Sample Code

```
"hdbprocedure": {
    "plugin_name": "com.sap.hana.di.procedure",
    "plugin_version": "11.1.0"
},
```

### Tip

By default, the *.hdiconfig* file is hidden. To display it, choose ► **View** ► **Show Hidden Files** in the menu bar.

- Test the stored procedure `createCountry.hdbprocedure`.

Use the database explorer to test the stored procedure and add some content to your database table `myworld.db::tinyf.country`.

- In the database browser, open the stored procedure `myworld.db::createCountry`.

The stored procedure is located in the application container `<UserName>-<ID>-myworld-hdi-container` in the folder *Procedures*. Right-click the procedure and choose ► **Invoke Procedure with UI** in the context menu.

- Add a country that does **not** already exist in the database, for example, “Ireland”.

In the *IM\_COUNTRY* box, enter **Ireland**; In the *IM\_CONTINENT* box, enter **Europe** and choose **Run** in the tool bar (or press **[F8]**).

- c. Check that the new country was added to the table *myworld.db::tinyf.country*.

The table is located in the application container *<UserName>-<ID>-myworld-hdi-container* in the folder *Tables*. Right-click the table *myworld.db::tinyf.country* and choose **▶ Open Content** in the context menu.

The country you added (for example, “Ireland”) should be displayed at the bottom of the table.

- d. Add a country that already exists in the database, for example, “Ireland”.

In the *IM\_COUNTRY* box, enter **Ireland**; in the *IM\_CONTINENT* box, enter **Europe** and choose **Run** in the tool bar (or press **[F8]**).

ERROR: Country Ireland already exists!

## Related Information

[Create a Database Function in XS advanced \[page 366\]](#)

[Creating the Data Persistence Artifacts in XS Advanced \[page 199\]](#)

### 5.5.3 Create a Database Function in XS advanced

Add a database function to your data model.

## Prerequisites

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

### ⚠ Caution

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

## Context

In SQL, a user-defined function (UDF) enables you to build complex logic into a single database object. A scalar UDF is a custom function that can be called in the SELECT and WHERE clauses of an SQL statement. In XS advanced a database function (scalar or table) has the file extension `.hdbfunction`, for example, `myScalarFunction.hdbfunction`. To create and deploy a procedure in XS advanced using the SAP Web IDE for SAP HANA, perform the following steps:

## Procedure

1. Start the SAP Web IDE for SAP HANA.
2. Display the application project to which you want to add a database function.

An application is created within a context of a project. If you do not already have a project, there are a number of ways to create one, for example: by importing it, cloning it, or creating a new one from scratch.

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

Currently, there is only one type of project template available, namely: *Multi-Target Application Project*. Select *Multi-Target Application Project* and choose *Next*.

- c. Type the name `myworld` for the new MTA project and choose *Next* to confirm.
- d. Specify details of the new MTA project and choose *Next* to confirm.
- e. Create the new MTA project; choose *Finish*.

3. Create a database function.

Database functions must be created in the application project's database module, for example, `db`. A scalar user-defined function (UDF) has a list of input parameters and returns the scalar values specified in the `RETURNS <return parameter list>` option defined in the SQL function, for example, `decimal(15, 2)`. The scalar UDF named “`apply_discount`” that you create in this tutorial performs the following actions:

- Applies a discount to the stored product price
- Calculates the sale price of a product including the suggested discount
- a. Create a new folder for your functions.

If it does not already exist, create a folder named `functions` in the `src/` folder of your application project's `db/` module, for example, `myworld/db/src/functions`.

- b. Create a new `.hdbfunction` file for your scalar user-defined function.

Right-click the application's database-module folder `db/src/procedures`, choose  *New > Function* from the context menu, and name the new stored scalar function `apply_discount`.

The new scalar function opens in the SAP Web IDE for SAP HANA.

- c. Add the following SQLScript code to the function-definition file `apply_discount.hdbfunction` and save the file.

## Sample Code

```
FUNCTION "myworld.db::apply_discount" (im_price decimal(15,2),
im_discount decimal(15,2))
    RETURNS result decimal(15,2)
    LANGUAGE SQLSCRIPT
    SQL SECURITY INVOKER AS
BEGIN
result := :im_price - ( :im_price * :im_discount );
END;
```

- d. Build the application database module db.

In the SAP Web IDE for SAP HANA, right-click the application module *myworld/db*, choose  **Build**  in the context menu, and follow the build progress in the output displayed in the console pane.

## Output Code

```
2:10:51 PM (Builder) Build of /myworld/db completed successfully.
```

4. Check that the required plug-in is available to build and deploy the function.

In SAP HANA XS advanced, a file suffix (for example, .hdbfunction) is linked to a specific plug-in, which is used to generate the corresponding catalog object from a design-time artifact.

The container-configuration file .hdiconfig must contain the following entry (version numbers can differ):

## Sample Code

```
"hdbfunction": {
    "plugin_name": "com.sap.hana.di.function",
    "plugin_version": "11.1.0"
},
```

## Tip

By default, the .hdiconfig configuration file is hidden. To display it, choose   **Show Hidden Files**  in the menu bar.

5. Test the new scalar function `apply_discount.hdbfunction`.

Use the SAP HANA database explorer included with the Web IDE to test the `apply_discount.hdbfunction` function.

- a. Open the database explorer from the Web IDE by choosing   **Database Explorer** .
- b. If your HDI container is not listed in the database browser, add it by choosing **Add Database to Database Explorer** from the database browser toolbar.

The HDI container is named `<UserName>-<ID>-myworld-hdi-container`

- c. In the database browser, open the function `apply_discount.hdbfunction`.

The function is located in the folder *Functions*.

- d. The function opens in a tab on the right.

## Related Information

[Create a Database Procedure in XS Advanced \[page 363\]](#)

[Creating Procedures and Functions in XS Advanced \[page 352\]](#)

## 5.6 Using Synonyms to Access External Schemas and Objects in XS Advanced

Deploy synonyms to enable cross-container access to external objects in XS advanced.

The information in this section covers the following basic application-development scenarios:

- Enable access to tables owned by a classical schema from an XS-advanced-based schema (HDI container).  
For example, enable access to an existing ERP schema from XS advanced.
- Enable access to tables owned by one XS-advanced-based schema from another XS-advanced-based schema.  
For example, enable one HDI container to access another HDI container, or enable an XS advanced Data Warehouse application to access a schema containing data in another XS advanced application's HDI container.

## Related Information

[Enable Access to Objects in a Remote Classic Schema \[page 370\]](#)

[Enable Access to Objects in Another HDI Container \[page 394\]](#)

## 5.6.1 Enable Access to Objects in a Remote Classic Schema

Use a synonym to enable access to objects in a remote schema not managed by XS advanced (for example, ERP).

### Prerequisites

To complete the steps described in this task, the following prerequisites apply:

- You have access to the XS advanced run-time environment
- You have access to SAP Web IDE for SAP HANA
- You have access to the XS command-line interface client
- You have access to a “classic” SAP database schema in a remote system (for example, SAP NetWeaver EPM)

### Context

You can enable an XS advanced application to access objects in a remote schema; that is, a classic schema in a remote database. The target objects for the synonyms can be tables, views, functions, and procedures, as well as database sequences.

#### i Note

For illustration purposes only, the target (base) objects used in this example are taken from SAP NetWeaver EPM content (Sales Order Model). However, you can substitute the target objects specified in the examples with your own target objects. Similarly, you can substitute the synonyms, roles, etc. specified in the examples with your own artifacts.

### Procedure

1. Create the underlying target objects in the external schema (for example, `EPM_DEV`) to which the XS advanced application's synonym points.

The target objects can be tables, views, functions, procedures, and so on. You might have a number of tables in a schema, not all of which should be exposed by a synonym.

#### ➔ Tip

If the target objects already exist, you can skip this step.

2. Create the roles to be granted for external access by means of the synonym.

The following roles are required here:

- `external_access_g`

Grant select privilege on schema `EPM_DEV` to "`EPM_XXX::external_access_g`" **with** grant option to allow access to the whole schema

```
create role "EPM_XXX::external_access_g";
grant select on SNWD_AD to "EPM_XXX::external_access_g" with grant option;
grant select on SNWD_BPA to "EPM_XXX::external_access_g" with grant option;
grant select on SNWD_PD to "EPM_XXX::external_access_g" with grant option;
grant select on SNWD_PD_CATGOS to "EPM_XXX::external_access_g" with grant option;
grant select on SNWD_SO to "EPM_XXX::external_access_g" with grant option;
grant select on SNWD_SO_I to "EPM_XXX::external_access_g" with grant option;
grant select on SNWD_SO_SL to "EPM_XXX::external_access_g" with grant option;
```

- `external_access`

Grant select privilege on schema `EPM_DEV` to "`EPM_XXX::external_access_appuser`" to enable access **without** grant option to the whole schema.

```
create role "EPM_XXX::external_access_appuser";
grant select on SNWD_AD to "EPM_XXX::external_access_appuser";
grant select on SNWD_BPA to "EPM_XXX::external_access_appuser";
grant select on SNWD_PD to "EPM_XXX::external_access_appuser";
grant select on SNWD_PD_CATGOS to "EPM_XXX::external_access_appuser";
grant select on SNWD_SO to "EPM_XXX::external_access_appuser";
grant select on SNWD_SO_I to "EPM_XXX::external_access_appuser";
grant select on SNWD_SO_SL to "EPM_XXX::external_access_appuser";
```

### 3. Create the synonym(s).

You can create the synonym design-time object in a subfolder of your XS advanced application's database module, for example in `/<MyApp>/db/src/synonyms/SNWD.hdbsynonym`. In this example, we name the synonym definition `SNWD.hdbsynonym`, which contains multiple synonyms referencing tables/views in the target schema "`EPM_DEV`".

#### Sample Code

XS Advanced Synonym Definition (`/MyApp/db/src/synonyms/SNWD.hdbsynonym`)

```
{
  "SNWD_AD": {
    "target": {
      "object": "SNWD_AD",
      "schema": "EPM_DEV"
    }
  },
  "SNWD_BPA": {
    "target": {
      "object": "SNWD_BPA",
      "schema": "EPM_DEV"
    }
  },
  [... synonyms {} ...]
  "SNWD_SO_SI": {
    "target": {
      "object": "SNWD_SO_SL",
      "schema": "EPM_DEV"
    }
  }
}
```

- Grant access to the synonym's target objects (for example, table); the access can also be used by any views that "consume" the target objects.

Access privileges are required on the target objects to which the synonym points. The type of access required (SELECT/EXECUTE) depends on the object type: SELECT for tables, views, and functions; EXECUTE for functions and procedures.

The access mode is only relevant for target objects that are procedures and views. The access mode specifies the user whose privileges will be checked upon execution of the procedure. For procedures, the developer must explicitly specify the desired access mode (for example, DEFINER/INVOKER) when writing the procedure. Views are always DEFINER mode.

Access can be enabled by referencing defined user roles with the "roles" property in an .hdbgrants file, as illustrated in the following example:

### Caution

To prevent unwanted schema access, for example, via "definer-mode", it is recommended to assign different roles to the object owner and the application user.

### Sample Code

myApp/db/cfg/emp\_dev.hdbgrants File

```
{  
    "EPM_XXX-table-grantor": {  
        "object_owner": {  
            "roles": [  
                "EPM_XXX::external_access_g"  
            ]  
        },  
        "application_user": {  
            "roles": [  
                "EPM_XXX::external_access_appuser"  
            ]  
        }  
    }  
}
```

"EPM\_XXX-table-grantor" is a symbolic name for a user-defined service that executes the GRANT statement. The user-defined service must be specified in the application project's development-descriptor (mta.yaml). "application\_user" refers to a technical access role; it is not a reference to a real user.

The role specified for assignment to "object\_owner" should contain the SELECT WITH GRANT OPTION privilege on the corresponding target table; the role specified for assignment to "application\_user" should specify the SELECT privilege (**without** GRANT OPTION) for the corresponding target tables, view, sequences, and EXECUTE WITH GRANT OPTION for the corresponding functions and procedures to enable access to the target objects via the synonyms created.

- Add references to "EPM\_XXX-table-grantor" in the development descriptor (mta.yaml) for the application that needs to use the synonym (to access the target objects).

### Sample Code

Application Development Descriptor (MyApp/mta.yaml)

```
_schema-version: '2.0'
```

```

ID: syn-hdi-classic-1
version: 0.0.1
modules:
  - name: db
    type: hdb
    path: db
    requires:
      - name: hdi-container
        properties:
          TARGET_CONTAINER: ~{hdi-container-service}
            # db module needs:
            # where synonyms are
created
  properties:
    - name: EPM_XXX-table-grantor
      properties:
        # for executing grant
statement

resources:
  - name: hdi-container
    type: com.sap.xs.hdi-container
    properties:
      hdi-container-service: ${service-name}
  - name: EPM_XXX-table-grantor
    type: org.cloudfoundry(existing-service)           # service created with
    xs cups

```

6. Create a user-defined service (for example, `EPM_XXX-table-grantor`) in the XS advanced application developer space.

The “user provided service” enables you to assign roles automatically to the generated users of the HDI container. The service uses the permissions assigned to a specified user to connect to a database and execute grant statements during application deployment. The grant statements are generated from content defined in an `.hdbgrants` file such as the one defined in a previous step.

### Note

XS advanced services have access to any other service running in the same organization and space. This means that any service running in the same organization and space as the user-provided service you create in this step can use the user-provided service to access external objects.

- a. Open a command shell and log on to the XS CLI with XS administrator privileges.

```
xs-admin-login
```

- b. Change to the target space where you want to create the user-defined service.

```
xs target -s <SPACE>
```

- c. Create the user-defined service `EPM_XXX-table-grantor`.

```
xs cups EPM_XXX-table-grantor -p "{\"host\":\"host.acme.com\", \"port\": \"30015\", \"certificate\": <myCertificate>, \"user\": \"EPM_DEV\", \"password\": \"Grant_123\", \"driver\": \"com.sap.db.jdbc.Driver\", \"tags\": [\"hana\"], \"schema\" : \"EPM_XXX\" }"
```

This command creates a user-defined service with the following configuration:

```
"EPM_XXX-table-grantor": [
  {
    "schema": "EPM_XXX",
    "password": "Grant_123",
    "driver": "com.sap.db.jdbc.Driver",
    "port": "30015",
    "host": "host.acme.com",
    "user": "EPM_DEV",
```

```
        "certificate": "<myCert>"  
        "tags": [ "hana" ]  
    }  
]
```

- "schema"  
The database schema that contains the objects to which access is to be granted.
- "user"/"password"  
Connection details for a database user that has grant permissions for the objects in the schema.
- "host"/"port"  
Required for the connection to the database: port is the SQL port of the index server.
- "driver"  
Required to set up the client connection to the database
- "certificate"  
If the database is configured to only accept **secure** JDBC connections, then the grantor service requires an SSL certificate that must be included in the user-provided service, for example, using the "certificate": "<myCertificate>" parameter.

#### ➔ Tip

Starting with version 3.0.0 of the HDI deployer service, the "host", "port", and "certificate" parameters are no longer required since they can be obtained from the target container binding. In this case, you must only specify the "user", "password", and "schema" when creating the new user-provided service.

- d. Check that the new user-defined service is running in the target space.

Use the command `xs services` to display a list of services available in the current space; the following service should be visible:

#### Output Code

```
xs services  
Getting services in org "myorg" / space "DEV" as XSA_ADMIN...  
Found services:  
name           service      plan   bound apps  
-----  
EPM_XXX-table-grantor    user-provided
```

7. Build the XS advanced application's DB module.

You can use the tools included in SAP Web IDE for SAP HANA.

8. Consume the synonym(s) created so far.

One obvious and simple way to consume synonyms is to use a calculation view (called `myView` in this example) on a table.

#### Sample Code

Calculation View /myApp/db/src/CUSTOMERS.hdbview

```
VIEW "CUSTOMERS" AS SELECT  
POSTAL_CODE, CITY, STREET, BUILDING  
FROM SNWD_AD
```

9. Check for the resulting objects in the database catalog.

You can use the *Database Explorer* tools included in SAP Web IDE for SAP HANA.

10. Create a role definition that enables access to the view.

► Tip

The view is not **required** for the creation of a synonym; it is used here to test “consumption” of the synonym.

Sample Code

Role Definition `external_v_access.hdbrole`

```
{  
  "role": {  
    "name": "external_v_access",  
    "object_privileges": [  
      {  
        "name": "CUSTOMERS",  
        "type": "VIEW",  
        "privileges": [ "SELECT" ]  
      }  
    ]  
  }  
}
```

11. Assign the role to consumers of the view.

The new role `external_v_access.hdbrole` must be assigned to the container-external user (for example, `SYN_CONS`) who needs access to the view (in this example, `myView`).

12. Test access to the view.

Both the XS advanced application user **and** the schema-external user should now have access to the view, which you can test by opening it in the *Database Explorer*.

- XS advanced application user  
With the container's access role, this user has access to all objects in the application container's runtime schema.
- Schema-external user  
Access to the view is granted by assignment of the role `external_v_access`.

## Related Information

[Database Synonyms in XS Advanced \[page 376\]](#)

[Syntax Options in the hdbgrants File \[page 387\]](#)

[Roles \(.hdbrole\) \[page 807\]](#)

[Enable Access to Objects in Another HDI Container \[page 394\]](#)

## 5.6.1.1 Database Synonyms in XS Advanced

You can use synonyms in XS advanced to enable access to objects that are not in the same schema or application container.

In general terms a synonym is an alias for a database object. Whenever you use a synonym, it is useful to try to mentally replace it by its base object: the object that is the target of the synonym. A synonym belongs to its own schema, which is typically (but not always) independent of the schema to which the target database object belongs. Synonyms can be created for tables, views, procedures, table functions, scalar functions and sequences. Synonyms are most commonly used to hide the real object names from consumers or to give a database object a more convenient name. In migration scenarios, it also sometimes necessary to be able to recreate the synonym after modifying it to point to another target object.

To create a synonym, a user needs the following privileges:

- **SELECT**  
For the tables, views, and sequences to which the synonym refers as base objects
- **EXECUTE**  
For procedures and functions to which the synonym refers as base object.

After creation, the synonym exists independently of the base target object. A synonym remains in existence even if its creator no longer has the privilege required to access the synonym's target database object or the base target object has been deleted.

### i Note

A synonym can be “private” (belonging to a specific schema), or “public” (residing in the special schema PUBLIC).

Although a public synonym can be used by any user without explicitly specifying the schema name, the appropriate privilege for accessing the base object of a public synonym is still required. If public and private objects or synonyms have identical names, the private synonym is used.

### ⚠ Restriction

It is not possible to use a synonym as the base (target) object for another synonym.

If you need to point a synonym to another synonym, you have to put a real database object in between the two synonyms. For example, you would need to create a view for a scenario where the second synonym's base object is a table or a table function. If the second synonym points to a procedure, you must insert an additional procedure between the synonyms. If the second synonym points to a scalar function, you must insert an additional scalar function. It is not possible to use a sequence as the base target object for another sequence with a synonym in between.

## Synonyms in XS Advanced

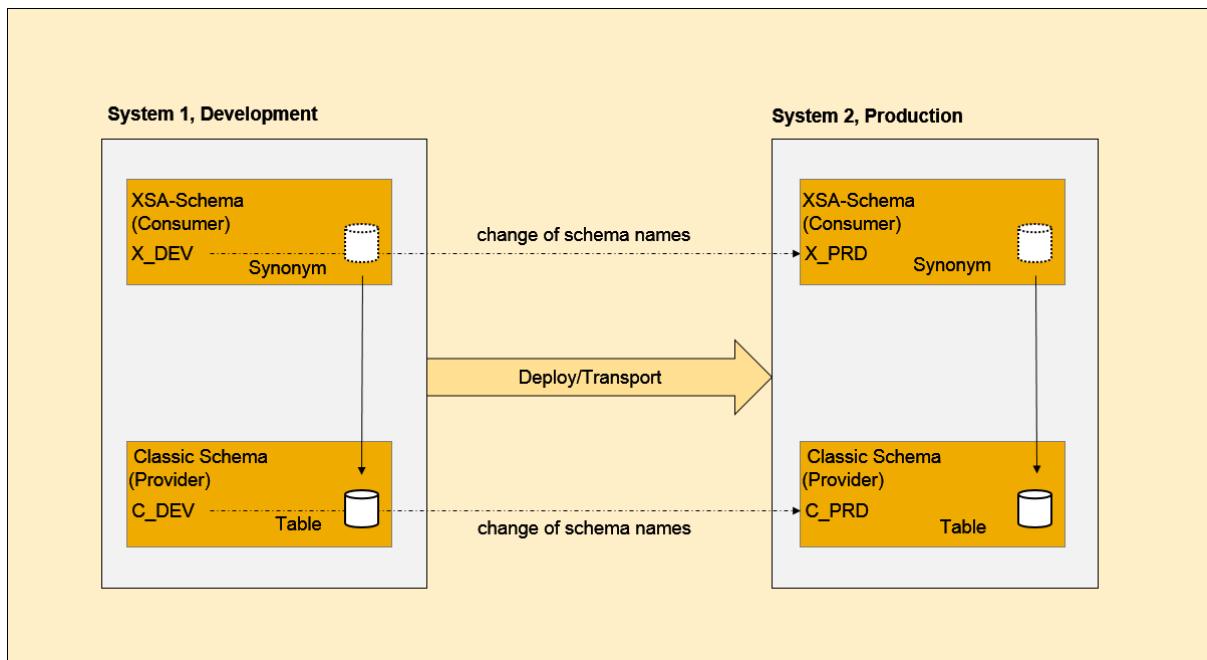
In XS advanced database objects such as tables and views reside in an application-specific container, which is a sort of generated schema. For this reason, development has to be done in a schema-less way. The concept of isolated HDI containers makes it possible to deploy multiple containers into the same system, have multiple

developers work on the same application independently from other developers, or even deploy applications and containers into a public Cloud environment, where binding between different containers (or services in general) should not be hard coded but done using a configuration mechanism during deployment.

Synonyms play a more important role in XS advanced and HDI than they do in the schemas used in XS classic. In XS advanced, using synonyms is the designated method to enable access to objects in other schemas, for example, a schema owned by another XS advanced application container.

The following diagram provides a visual overview of the most commonly used scenarios where synonyms in XS advanced can be effectively deployed to access objects in classic schemas outside of XS advanced, for example:

- Enabling access to tables provided by a classic (ERP) schema from an XS-advanced schema (an HDI container)
- Enabling access to tables provided by one XS-advanced schema from a different XS-advanced schema, for example, one schema containing data and different XS advanced Data Warehouse (DW) applications accessing the data
- Transporting schemas and the objects they contain to (and deploying them in) another environment



In the context of database synonyms, it is important to understand the role of the following users:

- Schema owner  
The user who owns the HDI container (or schema) where the synonym resides
- Object owner  
The user who creates the objects deployed within the container, for example, the synonym. The synonym's "target" objects belong to another user, for example, the user of the external schema.
- Application user  
The SAP HANA run-time user which runs the XS advanced applications in the HDI container

## Related Information

[Using Synonyms to Access External Schemas and Objects in XS Advanced \[page 369\]](#)  
[Syntax Options in the hdbgrants File \[page 387\]](#)

### 5.6.1.2 Users, Privileges, and Schemas

A short overview of which properties, users, and privileges are involved when using synonyms in XS advanced.

For security reasons, some database objects can be run in either `DEFINER` or `INVOKER` mode. The security mode is not defined in an access role; it is defined by the database object type, or in the case of a procedure, explicitly. The security mode is determined, in part, by the **type** of database object the application user calls, for example, a view, a synonym, or a procedure. More importantly for this XS advanced scenario, the security mode also determines which privileges need to be specified in the user role required to access the database object.

- [Definer-Mode Access \[page 379\]](#)
- [Invoker-Mode Access \[page 381\]](#)

#### ➔ Tip

In the default `DEFINER` security mode, a database object (for example, a definer-mode procedure or a view) is executed with the privileges assigned to the object's owner. In the `INVOKER` security mode, a database object (for example, an invoker-mode procedure or a user-defined function (UDF)) is executed with the privileges assigned to the user who calls it.

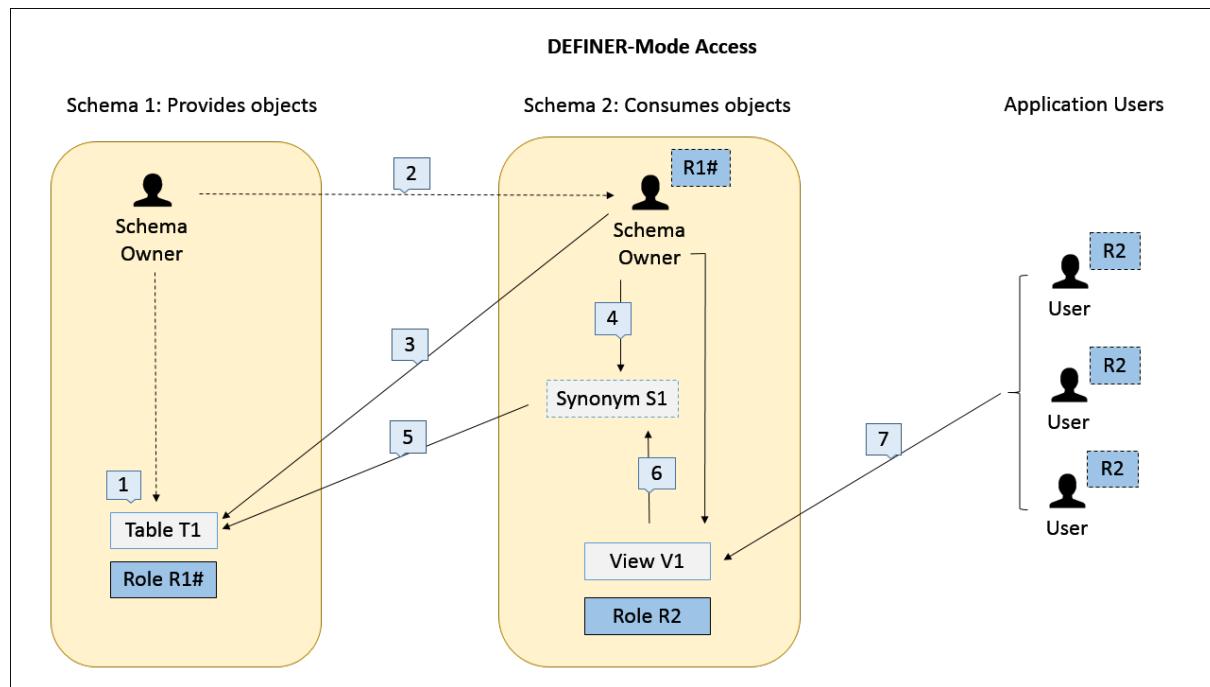
It is highly recommended to define object privileges in roles which can then be used to assign privileges for the target base object. This is particularly important for scenarios where the target base object is removed for some reason.

If a synonym's target base object (for example, a table or a view) is dropped and recreated, the corresponding object privilege has to be granted individually again to all referenced users. However, in most scenarios, the users consuming the target base object (indirectly via the synonym) are not known.

By using roles that include the object privilege, it is only necessary to reassign the object privilege to the respective role. All referencing users automatically receive the object privilege via their assigned roles.

## Definer-Mode Access

The following diagram illustrates the users and privileges that are involved when setting up and using synonyms to enable “DEFINER”-mode access to database objects in an external schema (or another container) in XS advanced:



The various roles illustrated in the diagram are described in the following list:

- Role R1#
  - SELECT privilege on Table T1 with GRANT OPTION  
Assigned to the owner of the schema that contains the consuming objects, for example, the synonym S1
- Role R2
  - SELECT privilege on View V1  
Assigned to the user of the application. This user is then able to access to the objects exposed by Role R2

### ➔ Tip

In the default “DEFINER” mode, a database object is executed with the privileges assigned to the object's owner; that is, the user who defines the object.

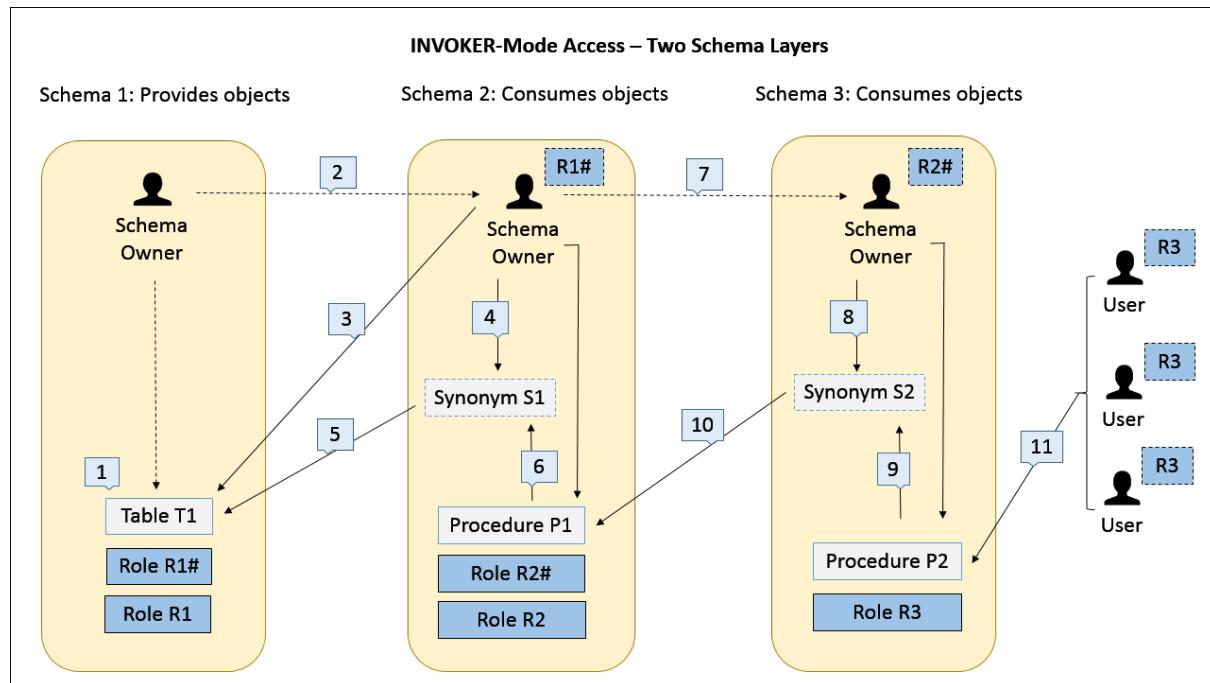
The following table provides more detailed information concerning the process of setting up and using synonyms in XS advanced:

Table 37: Synonym Users and Privileges for DEFINER-Mode Access

Step	Action	Comments
1	Create base target objects "Table T1" and role "Role R1#".	Synonyms can be created for other database objects, too, for example: tables, views, procedures, table functions, scalar functions, and sequences.  Role R1# assigns the SELECT privilege on Table T1 with GRANT OPTION
2	Grant privileges to "Schema2" owner.	Use RoleR1#.hdbrole to assign the required privileges.  <b>→ Tip</b> The "Schema Owner" in "Schema2" is the "object_owner" in the hdbgrants file.
3	Ensure the owner of "Schema2" has access to "Table T1"	The owner requires SELECT privilege on Table T1 with GRANT OPTION which is defined in Role R1#.
4	Create "Synonym S1" (and role "Role R2").	When creating a synonym, the SELECT privilege is required for the DEFINER-mode access to the table, view, or sequence referenced as a base object.  Role R2 defines the privileges required by an application user to access the View V1.
5	Use "Synonym S1" to access "Table T1".	To access objects within an HDI-container, only private (i.e. schema-local) synonyms can be used.
6	Create a view to access "Synonym S1".	"View V1" references "Synonym S1".
7	Application users can access "Table T1" in "Schema 1" via "View V1" in "Schema 2"	Requires the SELECT privilege on "View V1"; this is defined in Role R2.  SELECT privilege on Table T1 is defined in Role R1#)
		<b>→ Tip</b> User access to "View V1" is enabled using the technical user application_user (with the global access_role).

## Invoker-Mode Access

The following diagram illustrates the users and privileges that are involved when using synonyms to enable "INVOKER"-mode access to database objects in an external schema (or another container) in XS advanced:



The various roles illustrated in the diagram for "INVOKER"-mode access are described in the following list:

- Role R1
  - Assigned to the user of the application that consumes the underlying target objects
    - SELECT privilege on Table T1
- Role R1#
  - Assigned to the owner of schema 2 which contains the consuming objects procedure P1 and synonym S1
    - SELECT privilege with GRANT OPTION on Table T1
- Role R2
  - Assigned to the user of the application that consumes the underlying target objects
    - EXECUTE privilege on Procedure P1
    - The privileges defined in Role R1 (INVOKER mode only)
- Role R2#
  - Assigned to the owner of schema 3 which contains the consuming objects
    - EXECUTE privilege with GRANT OPTION on Procedure P1
    - The privileges defined in Role R1# (INVOKER mode only)
- Role R3
  - Assigned to the user of the application that consumes the underlying target objects
    - EXECUTE privilege on Procedure P2
    - The privileges defined in Role R2 (INVOKER mode only)
    - The privileges defined in Role R1 (INVOKER mode only); this is included in Role R2

➔ Tip

In "INVOKER" mode, a database object (for example, an invoker-mode procedure or a user-defined function (UDF)) is executed with the privileges assigned to the user who starts it.

The following table provides more detailed information concerning the process of setting up and using multiple synonyms to enable access to INVOKER-mode database objects in XS advanced:

Table 38: Synonym Users and Privileges for INVOKER-Mode Access

Step	Action	Comments
1	Create base target objects "Table T1" as well as "Role R1" and "Role R1#".	Synonyms can be created for other database objects, too, for example: tables, views, procedures, table functions, scalar functions, and sequences.  Role R1 assigns the SELECT privilege on Table T1  Role R1# assigns the SELECT privilege on Table T1 with GRANT OPTION
2	Assign access privileges to the owner of "Schema2".	Use RoleR1#.hdbrole to assign the required privileges.  ➔ Tip The "Schema Owner" in "Schema2" is the "object_owner" in the hdbgrants file.
3	Ensure the owner of "Schema2" has access to "Table T1"	Role R1# includes the SELECT privilege WITH GRANT OPTION on Table T1.
4	Create "Synonym S1", "Role R2", and "Role R2#".	"Role R2" defines the EXECUTE privilege on Procedure P1 and includes a reference to the privileges defined in Role R1 (INVOKER mode only)  "Role R2#" defines the EXECUTE privilege with GRANT OPTION on Procedure P1 and also includes a reference to the privileges defined in Role R1# (INVOKER mode only)
5	Use "Synonym S1" to access "Table T1".	When creating a synonym, the SELECT privilege is required for the table, view, or sequence referenced as a base object; for procedures and functions that are referenced as a base object, the EXECUTE privilege is required.  ⚠ Restriction To access objects within an HDI-container, only private (schema-local) synonyms can be used.

Step	Action	Comments
6	Create an INVOKER-mode procedure "Procedure P1" to access "Synonym S1".	"Procedure P1" references "Synonym S1"
7	Assign access privileges to the owner of "Schema 3".	<p>Use a database role object, for example, role RoleR2.hdbrole.</p> <p><b>► Tip</b> The "Schema Owner" in "Schema3" is the "object_owner" in the hdbgrants file; the user(s) in "Schema3" are the technical user "application_user".</p> <p><b>i Note</b> The role definition RoleR2.hdbrole must ensure that the owner of Schema 3 has the grant option and can assign privileges for access to "Procedure P1" in Schema 2.</p>
8	Create "Synonym S2" and "Role R3".	<p>Synonym S2 provides access to Procedure P1 which calls Synonym S1.</p> <p>Role R3 defines the privileges required for INVOKER mode access to Procedure P2. Role R3 must also include a reference to Role R2, which in turn includes Role R1.</p>
9	Create an INVOKER-mode procedure "Procedure P2" to access "Synonym S2".	As explained in a previous step, a synonym cannot be used as the base object for another synonym.
10	Enable access to base object Table T1 from Schema 3.	<p>Requires the following privileges:</p> <ul style="list-style-type: none"> <li>• Role R1: SELECT ON T1 Role R1#: SELECT ON T1 WITH GRANT OPTION</li> <li>• Role R2: EXECUTE ON P1 + Role R1 Role R2#: EXECUTE ON P1 WITH GRANT OPTION + Role R1#</li> <li>• Role R3: EXECUTE ON P2 + Role R2 + Role R1</li> </ul>
11	Assign access privileges to the application users.	<p>Access privileges are defined in Role R3.</p> <p><b>► Tip</b> In the hdbgrants file, the user(s) in "Schema3" are the technical user "application_user".</p>

## Related Information

[Enable Access to Objects in a Remote Classic Schema \[page 370\]](#)

[Database Synonyms in XS Advanced \[page 376\]](#)

[Syntax Options in the hdbgrants File \[page 387\]](#)

### 5.6.1.3 Granting Roles and Privileges for Use with Synonyms

Assigning roles and privileges to object owners for applications that use synonyms to access external objects.

You can use the `.hdbgrants` configuration file to assign privileges to the owner of the synonym object and the application users (consumers) of the synonym's target objects in the same way as you would with the SQL grant statement.

#### Sample Code

`SNWD-table.hdbgrants`

```
{  
  "EPM_XXX-table-grantor": {  
    "object_owner": {  
      "schema_roles": [  
        "roles" : "EPM_XXX::RoleR1"  
      ]  
    },  
    "application_user": {  
      "schema_roles": [  
        "roles" : "EPM_XXX::RoleR2"  
      ]  
    }  
  }  
}
```

“`EPM_XXX-table-grantor`” is a symbolic name for a user-defined service that executes the GRANT statement. The user-defined service must be specified in the resources section of the application project’s development-descriptor (`mta.yaml`).

#### object\_owner

The “`object_owner`” property defines the roles to be assigned to the container’s **object** owner. The name of the role differs according to whether the target object is in an XS advanced schema (application container) or a classic SAP schema (for example, ERP).

#### Sample Code

`.hdbgrants` File for Role Assignment in a Classic Schema

```
"object_owner": {  
  "schema_roles": [  
    "roles" : "EPM_XXX::RoleR1"  
  ]  
},
```

If the synonym’s target database object is in an XS advanced application’s HDI container, the owner of the object is the schema owner. HDI container schema owners have a special name that ends with the string “#oo”

(for example, ContainerOwner#00), the corresponding role must also have the suffix #, for example, RoleR1#, as illustrated in the following example:

#### Sample Code

.hdbgrants File for Role Assignment in an HDI Container Schema

```
"object_owner": {  
    "schema_roles": [  
        "roles" : "RoleR1#"  
    ]  
},
```

If a role name ends with "#", for example, MyRole#, the role definition (specified in an hdbrole artifact) is allowed to include "WITH GRANT OPTION" privileges as well as references to other roles whose names end with "#". The HDI container object owner needs most privileges "WITH GRANT OPTION".

## application\_user

The "application\_user" property defines the roles to be assigned to the user of the XS advanced application that consumes the defined synonym and requires access to the corresponding schemas containing the target objects referenced by the synonym. The following example shows how to assign application\_user to the privileges defined in the externally created role "EPM\_XXX::RoleR1".

#### Sample Code

SNWD-table.hdbgrants

```
"application_user": {  
    "schema_roles": [  
        "roles" : "EPM_XXX::RoleR1"  
    ]  
}
```

#### ➔ Tip

"application\_user"" does not refer to a real user; it refers to the technical user associated with the global "access\_role" required for access to the corresponding run-time container. The "access\_role" is assigned a set of default permissions for the run-time schema, for example: SELECT, INSERT, UPDATE, DELETE, EXECUTE, CREATE TEMPORARY TABLE, and SELECT CDS METADATA.

## schema\_roles

The schema\_roles property is used to specify one or more schema-local "roles" to users by means of a grantor service. In XS advanced, access to the synonym's target object is enabled by assigning a local HDI container role that grants the container's object owner access to the target object in the foreign schema. This role must also have the privileges\_with\_grant\_option, and for this reason the role name must end with

the hash (#) character, for example, RoleR1#. Assigning XS advanced roles to an HDI schema owner requires some changes to the .hdbgrants file, as illustrated in the following example:

### Sample Code

SNWD-table.hdbgrants for HDI Schema (Container) Roles

```
{  
    "EPM_XXX-table-grantor": {  
        "object_owner": {  
            "schema_roles": [  
                "roles" : "EPM_XXX::RoleR1#"  
            ]  
        },  
        "application_user": {  
            "schema_roles": [  
                "roles" : "EPM_XXX::RoleR1"  
            ]  
        }  
    }  
}
```

Due to naming conventions in HDI, and the fact that the container role must provide the GRANT privileges, the name of the role-definition assigned to the HDI container's object owner must have the suffix "#", for example, "MyRole#".

If a role name ends with the hash character (#), for example, MyRole#, the role definition can include "with-grant-option" privileges as well as references to other roles whose names also end with #. A role whose name ends with # can only be granted by means of the HDI APIs to a container's object owner, for example, <container>#OO.

## roles

Use the "roles" property to specify the role (or roles) to be assigned to the object owner (one per HDI container) and application user by the grantor service, as illustrated in the following example:

### Sample Code

SNWD-table.hdbgrants

```
{  
    "EPM_XXX-table-grantor": {  
        "object_owner": {  
            "schema_roles": [  
                "roles" : ["EPM_XXX::RoleR1#"]  
            ]  
        },  
        "application_user": {  
            "schema_roles": [  
                "roles" : ["EPM_XXX::RoleR1"]  
            ]  
        }  
    }  
}
```

The attribute “roles” is used to specify the name of the role that defines the privileges to assign to users. The role name must be a non-empty string that contains only permitted characters.

In the `.hdbgrants` file, the following types of role names are possible:

- `<RoleName>`

If a role name does **not** end with the hash character (#), for example, `MyRole`, then only privileges without grant options can be included in the role as well as references to any other roles whose names also do not end with “#”.

- `<RoleName>#`

If a role name ends with the hash character (#), for example, `MyRole#`, its corresponding role definition can include “with-grant-option” privileges as well as references to other roles whose names also end with #. A role whose name ends with # can only be granted by means of the HDI APIs to a container’s object owner, for example, `<container>#OO`.

The roles specified in the `hdbgrants` file must already exist as objects in the database catalog. The corresponding role-definition must define the various privileges required for access to database objects, for example: system, schema, object, and analytic privileges. In XS advanced, you can define a user role with an `.hbrole` artifact; in XS classic, you can use alternative methods to define access privileges to database objects, for example, the SQL command `create role "EPM_XXX::external access`. In addition, you will also have to grant the `SELECT` privilege manually to each base objects referenced by the synonym.

#### Caution

To prevent unnecessary access to the schema via “definer-mode” objects (procedures, views), it is recommended to assign different roles to the object owner and the application user. The role for the object owner should provide `GRANT` options; the role for the application user should provide privileges for “invoker-mode” objects (for example, invoker-mode procedures).

## Related Information

[Enable Access to Objects in a Remote Classic Schema \[page 370\]](#)

[Syntax Options in the `hdbgrants` File \[page 387\]](#)

### 5.6.1.4 Syntax Options in the `hdbgrants` File

Assign the privileges required by users to access objects in the target schema.

The `.hdbgrants` configuration file enables you to assign privileges to the owner of the synonym object and the application users (consumers) of the synonym’s target objects in the same way as you would with the SQL `grant` statement.

#### Note

The `.hdbgrants` file is not an HDI artifact, so no HDI plug-in configuration is required.

## Sample Code

```
external-access.hdbgrants

{
    "external-access": {
        "object_owner": {
            "system_privileges" : [ "SYSTEM PRIVILEGE 1", "SYSTEM PRIVILEGE 2" ]
            "global_roles" : [
                {
                    "roles" : [ "GLOBAL_ROLE_1", "GLOBAL_ROLE_2" ]
                    "roles_with_admin_option" : [ "GLOBAL_ROLE_3", "GLOBAL_ROLE_4" ]
                }
            ],
            "schema_privileges" : [
                {
                    "privileges" : [ "INSERT", "UPDATE" ],
                    "privileges_with_grant_option" : [ "SELECT" ]
                }
            ],
            "schema_roles" : [
                {
                    "roles" : [ "SCHEMA_ROLE_1", "SCHEMA_ROLE_2" ]
                    "roles_with_admin_option" : [ "SCHEMA_ROLE_3", "SCHEMA_ROLE_4" ]
                }
            ],
            "object_privileges" : [
                {
                    "name": "AN_OBJECT",
                    "privileges": [ "INSERT", "UPDATE" ],
                    "privileges_with_grant_option" : [ "SELECT" ]
                }
            ],
            "global_object_privileges" : [
                {
                    "name" : "A_REMOTE_SOURCE",
                    "type" : "REMOTE SOURCE",
                    "privileges" : [ "CREATE VIRTUAL TABLE" ],
                    "privileges_with_grant_option" : [ "CREATE VIRTUAL PROCEDURE" ]
                }
            ],
            "application_user": {
                "system_privileges": [...],
                "roles": [...],
                "schema_privileges": [...],
                "schema_roles": [...],
                "object_privileges": [...],
                "global_object_privileges": [...]
            }
        }
    }
}
```

## object\_owner

Defines the roles and privileges to be assigned to the owner of the specified container's **object**. In the "object\_owner" section, you can also set the following parameters:

- system\_privileges

- global\_roles
- schema\_privileges
- schema\_roles
- object\_privileges
- global\_object\_privileges

The name of the role differs according to whether the target object is in an XS advanced schema (application container) or a classic SAP schema (for example, ERP). The names of the privileges can include system, schema, and object (and global\_object) privileges, as illustrated in the following example:

### Sample Code

.hdbgrants File for Role Assignment in a Classic Schema

```
{
  "external_access": {
    "object_owner": {
      "system_privileges": [...],
      "global_roles": [...],
      "schema_privileges": [...],
      "schema_roles": [...],
      "object_privileges": [...],
      "global_object_privileges": [...]
    }
  }
}
```

## system\_privileges

You can use the “system\_privileges” parameter to assign one or more system permissions to the object owner as illustrated in the following example:

### Sample Code

System Privileges for the Object Owner in the hdbgrants File

```
{
  "external_access": {
    "object_owner": {
      "system_privileges" : ["System Privilege 1, System Privilege 2"],
      ...
    }
  }
}
```

## global\_roles

Use the “global\_roles” property to specify the role (or roles) to be assigned to the object owner (one per HDI container) and application user by the grantor service, as illustrated in the following example:

## Sample Code

SNWD-table.hdbgrants for Global Roles

```
{  
  "EPM_XXX-table-grantor": {  
    "object_owner": {  
      "global_roles": [  
        {  
          "roles" : ["Global_Role1", "Global_Role2"],  
          "roles_with_admin_option" : ["Global_Role3", "Global_Role4"]  
        }  
      ]  
    },  
    "application_user": {  
      "global_roles": [  
        {  
          "roles" : ["Global_Role1", "Global_Role2"],  
          "roles_with_admin_option" : ["Global_Role3", "Global_Role4"]  
        }  
      ]  
    },  
  }  
}
```

The property “roles” is used to specify the name of the role that defines the privileges to assign to users. The role name must be a non-empty string that contains only permitted characters.

In the .hdbgrants file, the following types of role names are possible:

- `<RoleName>`

If a role name does **not** end with the hash character (#), for example, `MyRole`, then only privileges without grant options can be included in the role as well as references to any other roles whose names also do not end with “#”.

- `<RoleName>#`

If a role name ends with the hash character (#), for example, `MyRole#`, its corresponding role definition can include “with-grant-option” privileges as well as references to other roles whose names also end with #. A role whose name ends with # can only be granted by means of the HDI APIs to a container’s object owner, for example, `<container>#OO`.

The roles specified in the hdbgrants file must already exist as objects in the database catalog. The corresponding role-definition must define the various privileges required for access to database objects, for example: system, schema, object, and analytic privileges. In XS advanced, you can define a user role with an .hdbrole artifact; in XS classic, you can use alternative methods to define access privileges to database objects, for example, the SQL command `create role "EPM_XXX::external access`. In addition, you will also have to grant the SELECT privilege manually to each base objects referenced by the synonym.

### Caution

To prevent unnecessary access to the schema via “definer-mode” objects (procedures, views), it is recommended to assign different roles to the object owner and the application user. The role for the object owner should provide GRANT options; the role for the application user should provide privileges for “invoker-mode” objects (for example, invoker-mode procedures). The roles themselves are defined in .hdbrole artifacts.

## schema\_privileges

You can use the “`schema_privileges`” parameter to assign permissions for the object owner in the target schema, for example, `INSERT` or `UPDATE` for “`privileges`”. You can also add the grant option to the system privilege, for example, `privileges_with_grant_option` on `SELECT`.

### Sample Code

Schema Privileges for the Object Owner in the `hdbgrants` File

```
{  
  "external_access": {  
    "object_owner" : {  
      [  
        ...  
      ],  
      "schema_privileges" : [  
        {  
          "privileges" : [ "INSERT", "UPDATE" ],  
          "privileges_with_grant_option" : [ "SELECT" ]  
        }  
      ],  
      ...  
    }  
  }  
}
```

## schema\_roles

Use “`schema_roles`” to grant schema-local roles to users by means of a grantor service. Multiple roles can be specified in an array, for example, “`roles`” : `["Schema_Role_1", "Schema_Role_1"]`. You can also specify additional roles that include administrator options, for example, “`roles_with_admin_option`” : `["Schema_Role_3", "Schema_Role_4"]`.

If the grantor service is bound to a normal database user, you can assign roles with or without administrator options, as illustrated in the following example:

### Sample Code

`SNWD-table.hdbgrants` for HDI Schema (Container) Roles

```
{  
  "EPM_XXX-table-grantor": {  
    "object_owner": {  
      "schema_roles": [  
        {  
          "roles" : ["Schema_RoleR1", "Schema_RoleR2"],  
          "roles_with_admin_option" : ["Schema_RoleR3", "Schema_RoleR4"]  
        }  
      ]  
    }  
  }  
}
```

## Restriction

For container roles, the `roles-with-admin-option` is not supported.

If the grantor service is bound to an HDI container, then it is not possible to use the `"roles_with_admin_option"`. In the following example, the role assignment works for both normal database-user bindings and bindings to an HDI container, too:

### Sample Code

`SNWD-table.hdbgrants` for HDI Schema (Container) Roles

```
{  
  "EPM_XXX-table-grantor": {  
    "object_owner": {  
      "schema_roles": [  
        {  
          "roles" : ["Schema_RoleR1", "Schema_RoleR2"]  
        }  
      ]  
    }  
  }  
}
```

### Note

Although the parameter `schema_roles` replaces `container_roles`, `container_roles` will continue to be supported for backward compatibility.

## object\_privileges

For each individual object, you can assign privileges to the object owner: either with the `grant` option (for example, `SELECT`) or `without` (for example, `INSERT`, `UPDATE`, as illustrated in the following example:

### Sample Code

Object Privileges in the `hdbgrants` File

```
{  
  "external_access": {  
    "object_owner" : {  
      [  
        ...  
      ],  
      "object_privileges" : [  
        {  
          "name": "AN_OBJECT",  
          "privileges": [ "INSERT", "UPDATE" ],  
          "privileges_with_grant_option" : [ "SELECT" ]  
        }  
      ]  
    }  
  }  
}
```

## global\_object\_privileges

For each individual object, you can assign global privileges to the object owner: either with the grant option (for example, SELECT) or **without** (for example, INSERT, UPDATE, as illustrated in the following example):

### Sample Code

Global Object Privileges in the hdbgrants File

```
{  
  "external_access": {  
    "object_owner" : {  
      [  
        ...  
      ],  
      "global_object_privileges" : [  
        {  
          "name" : "A_REMOTE_SOURCE",  
          "type" : "REMOTE SOURCE",  
          "privileges" : [ "CREATE VIRTUAL TABLE" ],  
          "privileges_with_grant_option" : [ "CREATE VIRTUAL PROCEDURE" ]  
        }  
      ]  
    }  
  }  
}
```

## application\_user

Defines the roles and permissions to be assigned to the user of the XS advanced application that consumes the defined synonym and requires access to the corresponding schemas containing the target objects referenced by the synonyms. The following example shows how to assign `application_user` to the privileges defined in the externally created role "`EPM_XXX::RoleR1`".

### Tip

"`application_user`" does not refer to a real user; it refers to the technical user associated with the global "`access_role`" required for access to the corresponding run-time container. The "`access_role`" is assigned a set of default permissions for the run-time schema, for example: SELECT, INSERT, UPDATE, DELETE, EXECUTE, CREATE TEMPORARY TABLE, and SELECT CDS METADATA.

- `system_privileges`
- `global_roles`
- `schema_privileges`
- `schema_roles`
- `object_privileges`
- `global_object_privileges`

## Sample Code

.hdbgrants File for Role Assignment in a Classic Schema

```
"application_user": {  
    "system_privileges": [...],  
    "global_roles": [...],  
    "schema_privileges": [...],  
    "schema_roles": [...],  
    "object_privileges": [...],  
    "global_object_privileges": [...]  
},
```

## Related Information

[Using Synonyms to Access External Schemas and Objects in XS Advanced \[page 369\]](#)

[Database Synonyms in XS Advanced \[page 376\]](#)

[Roles \(.hbrole\) \[page 807\]](#)

## 5.6.2 Enable Access to Objects in Another HDI Container

Use a synonym to enable access to another HDI container.

## Prerequisites

To complete the steps described in this task, the following prerequisites apply:

- You have access to the XS advanced run-time environment
- You have access to SAP Web IDE for SAP HANA
- You have access to the XS command-line interface client

## Context

You can use a synonym in one XS advanced application to enable access to database objects in a different XS advanced application; that is, objects in an HDI container that belongs to another XS advanced application. You need to ensure access to the external HDI container and specify the type of privileges (SELECT, EXECUTE) required on the target object (table, view, etc.).

## Procedure

1. Locate the target object in the external schema to which the XS advanced synonym will point.

The target objects can be tables, views, functions, procedures, and so on; the target schema is another HDI container.

2. Create the XS advanced role definitions that enable access to the target database object.

In this scenario, you create two roles:

### Note

Both these roles must be created in the HDI schema that contains the target database object, for example, `Table_T1`.

- a. Create a role for the schema owner of the XS advanced application that contains the synonym.

The role `Role_R1# (Role_R1G.hdbrole)` defines the privileges required for external access to a specific target object (with grant option); the role must be assigned to the user who needs access to the schema where the target object `Table_T1` is located. In this case, the access role is assigned to the owner of the schema containing the synonym that points to the target table.

### Sample Code

Role Definition `Role_R1G.hdbrole` for HDI Schema Owner

```
{  
  "role": {  
    "name": "Role_R1#",  
    "object_privileges": [  
      {  
        "name": "<Table_T1>",  
        "type": "TABLE",  
        "privileges_with_grant_option": [ "SELECT" ]  
      }  
    ]  
  }  
}
```

### Tip

The hash character (#) at the end of the role name signifies that the role is intended for assignment to a schema owner (for example, `<Container>#OO`) and that the role must include the “with grant option” privilege.

- b. Create a role for the user of the XS advanced application which contains the synonym.

The role `Role_R1.hdbrole` defines the privileges required by the application user who needs access to the target object `Table_T1` by means of a synonym; the “SELECT” privilege is required for the target object (**without** grant option). The role must be assigned to the user of the XS advanced application which contains the synonym that needs access to the target object `Table_T1`.

### Note

Role\_R1 is assigned to the XS advanced application user by including a reference to it in RoleR2, which you define in a later step. Role\_R2 is located in the same HDI schema as the synonym.

### Sample Code

Role Definition Role\_R1.hdbrole for XS Advanced Application User

```
{  
  "role": {  
    "name": "Role_R1",  
    "object_privileges": [  
      {  
        "name": "<Table_T1>",  
        "type": "TABLE",  
        "privileges": [ "SELECT" ]  
      }  
    ]  
  }  
}
```

3. Grant access to the schema containing the synonym's target object (for example, Table\_T1).

The owner of the schema containing the synonym requires SELECT privileges on the target object to which the synonym points. Access for the schema owner can be enabled in user roles which are then referenced with the “container\_roles” property in an .hdbgrants file, as illustrated in the following example:

### Sample Code

Access-Granting Configuration File (myApp/db/synonyms/Synonym\_S1-table.hdbgrants)

```
{  
  "EPM_XXX-table-grantor": {  
    "object_owner": {  
      "container_roles": [  
        "Role_R1#"  
      ]  
    },  
    "application_user": {  
      "container_roles": [  
        "Role_R1"  
      ]  
    }  
  }  
}
```

### Caution

To prevent unwanted schema access, for example, via views, it is recommended to assign different roles to the object owner and the application user.

4. Create the synonym(s).

You can create the synonym design-time object in a subfolder of your XS advanced application's database module, for example in /<MyApp>/db/src/synonyms/. In this example, we name the synonym definition

`Synonym_S1.hdbsynonym`, which contains multiple synonyms referencing tables (`Table_T1`, `Table_T2`, and `Table_T3`), as illustrated in the following example:

### Sample Code

XS Advanced Synonym-Definition File (`/MyApp/db/src/synonyms/Synonym_S1.hdbsynonym`)

```
{  
    "Table_T1": {}  
}
```

### Note

In this example, the synonym configuration is not included in the synonym file; it is moved to the synonym's corresponding configuration file `SynonymS1.hdbsynonymconfig`, as described in the following step.

5. Create a synonym configuration file.

In XS advanced, the synonym configuration can also be defined in a `.hdbsynonymconfig` file, for example, `myApp/db/cfg/Synonym_S1.hdbsynonymconfig`.

### Note

The XS classic `.hdbsynonymtemplate` files can also be used; they are converted to the `.hdbsynonymconfig` file format required by XS advanced at deployment time.

6. Define details of the synonym configuration.

Rather than defining a schema and a concrete schema name, you can use “`schema.configure`” to specify a path to a service name. The path expression is replaced at deployment time with the name of the schema of the referenced service. For example, in the following code example, “`schema.configure`” is replaced with schema used by the grantor service.

### Sample Code

Synonym Configuration File (`myApp/db/cfg/Synonym_S1.hdbsynonymconfig`)

```
{  
    "Table_T1": {  
        "target": {  
            "object": "Table_T1",  
            "schema.configure" : "<target-hdi-container>/schema"  
        }  
    },  
    [...]  
}
```

7. Define the required HDI plug-in configuration for the synonym.

Add the XS advanced application project's central HDI plugin-configuration file (`.hdbconfig`) to the database module's “`cfg/`” folder. You can copy the HDI plugin-configuration file from `myapp/db/src/.hdbconfig` to `myapp/db/cfg/.hdbconfig`.

The `.hdiconfig` configuration file should include the following entries:

### Sample Code

#### HDI Plug-in Configuration File (`.hdiconfig`)

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

### Tip

The `"plugin_version": "2.0.0.0"` property is optional; from SAP HANA 2.0, the version of all plugins shipped with SAP HANA is the same as (and equal to) the SAP HANA version.

8. Update the XS advanced application's development descriptor (`mta.yaml`).

Add references to “`<target-hdi-container>`” definition in the development descriptor for the application that needs to use the synonym to access the target objects, as illustrated in the example below.

### Sample Code

#### Application Development Descriptor (`MyApp/mta.yaml`)

```
modules:
- name: db
  type: hdb
  path: db
  requires:
    - name: hdi-container
      properties:
        TARGET_CONTAINER: ~{hdi-container-service}
    - name: EPM_XXX-table-grantor
      group: SERVICE_REPLACEMENTS
      properties:
        key: EPM_log-table-grantor
        service: ~{EPM_Synonym_S1-table-grantor-service}
resources:
- name: hdi-container
  type: com.sap.xs.hdi-container
  properties:
    hdi-container-service: ${service-name}
- name: EPM_XXX-table-grantor
  type: org.cloudfoundry(existing-service
  properties:
    EPM_Synonym_S1-table-grantor-service: ${service-name}
  parameters:
    service-name: <HDI Container Name>-hdi-container
```

9. Consume the synonym you created.

One way to consume synonyms from inside the XS advanced application's schema is to use a view (for example, `View_V1`) on the target table.

### Sample Code

SQL View /myApp/db/src/View\_V1.hdbview

```
VIEW "View_V1" AS SELECT * FROM Table_T1
```

10. Create a role definition that enables access to the view consuming `Synonym_S1`.

### Tip

The view is not **required** for the creation of a synonym; it is used here to test “consumption” of the synonym `Synonym_S1`.

The role definition (for example, `Role_R2.hdbrole`) must be assigned to the application users who need access to the view (in this example, `View_V1`). The XS advanced application user already has access to the view through the application container's `access_role`; the role `Role_R2.hdbrole` is required by users outside of the application.

### Sample Code

Role Definition `myApp/db/cfg/Role_R2.hdbrole`

```
{
  "role": {
    "name": "Role_R2",
    "object_privileges": [
      {
        "name": "View_V1",
        "type": "VIEW",
        "privileges": [ "SELECT" ]
      }
    ]
  }
}
```

11. Build both XS advanced application projects: the application that contains the target table (`Table_T1`) and the application that contains `Synonym_S1` and `View_V1`.
12. Check the database catalogs to ensure the appropriate objects exist and are accessible with the synonyms you created.

## Related Information

[Using Synonyms to Access External Schemas and Objects in XS Advanced \[page 369\]](#)

[Database Synonyms in XS Advanced \[page 376\]](#)

[Syntax Options in the hdbgrants File \[page 387\]](#)

[Users, Privileges, and Schemas \[page 378\]](#)

## 5.7 Setting Up the Analytic Model

Modeling refers to an activity of refining or slicing data in database tables by creating analytic models or views to depict a business scenario.

The modeling process involves the simulation of entities, such as CUSTOMER, PRODUCT, and SALES, and relationships between them. You can use these related entities (analytic models) in analytics applications such as SAP BusinessObjects Explorer and Microsoft Office for reporting use cases.

Analytic models or views use various combinations of content data (that is, non-metadata) to model a business use case. Content data can be classified as follows:

- Attribute: Descriptive data, such as customer ID, city, and country.
- Measure: Quantifiable data, such as revenue, quantity sold and counters.

SAP Web IDE for SAP HANA offers data modelers and other user personas the modeling tools to work with SAP HANA content repository and SAP HANA database catalog objects.

### Related Information

[Create Graphical Calculation Views \[page 400\]](#)

[Preview Calculation View Output \[page 404\]](#)

[Defining Data Access Privileges \[page 406\]](#)

### 5.7.1 Create Graphical Calculation Views

Create graphical calculation views using a graphical editor to depict a complex business scenario. You can also create graphical calculation views to include layers of calculation logic.

### Context

Graphical calculation views can bring together normalized data that are dispersed. You can combine multiple transaction tables while creating a graphical calculation view.

### Procedure

1. Start the SAP Web IDE for SAP HANA tool in a Web browser.
2. If you want to create a new project for the calculation view, do the following:
  - a. In the SAP Web IDE for SAP HANA, choose  *File*  *New*  *Project from Template*.

- b. Choose the project template type.
  - Currently, there is only one type of project template available, namely: *Multi-Target Application Project*. Select *Multi-Target Application Project* and choose *Next*.
  - c. Type a name for the new MTA project (for example, `myApp` and choose *Next* to confirm).
  - d. Specify details of the new MTA project and choose *Next* to confirm.
  - e. Create the new MTA project; choose *Finish*.
3. Select the *SAP HANA Database Module* in which you want to create the calculation view.

➔ **Tip**

If you do not already have a database module, right-click the root folder of your new MTA project and, in the context menu, choose  *New*  *SAP HANA Database Module*.

4. Browse to the `src` folder, right-click it and choose  *New*  *Calculation View*.
5. Enter the details for the new calculation view.
  - a. In the *Name* field, enter the name of the calculation view.
  - b. In the *Data Category* dropdown list, select a value.
6. Choose *Create*.  
The tool launches a new graphical calculation view editor with a semantics node and default aggregation, or projection view node depending on the data category that you select for the calculation view.
7. Continue modeling the graphical calculation view by dragging and dropping the necessary view nodes from the tool palette.
8. Add data sources.

If you want to add data sources to the view node,

- a. Select a view node.



- b. Choose  *(Add Data Source)*.
- c. In the *Find Data Sources* dialog box, select the type of the data source.
- d. Enter the name of the data source and select it from the list.

You can add one or more data sources depending on the selected view node.

- e. Choose *Finish*.

**i Note**

Supported data sources in view nodes in the current version.

The *Find Data Sources* dialog box displays multiple object types in the search result. But, depending on the selected view node, you can only add activated catalog tables, calculation views, CDS entities, CDS views, virtual tables, and table functions as data sources in the view nodes.

You can also model calculation views using data sources from any tenant database available in the same SAP HANA instance.

9. (Optional) Add data sources from external services (non HDI).

You can use synonyms to access objects from user-defined schemas (non HDI). The tool automatically creates the `.hdbsynonym` and `.hdbgrants` files necessary for consuming the synonym.

## Note

You can access data sources from external services only if you have created the necessary user-provided service, and configured the user-provided service in the `mta.yaml` file.

- a. Select a view node.



- b. Choose  (Add Data Source).

The tool opens the *Find Data Sources* dialog box.

- c. In the *External Services* dropdown list, select the required external service.
- d. Enter the name of the data source in the external service and select it from the search list.

You can add one or more data sources depending on the selected view node.

- e. Choose *Next*.
- f. (Optional) Modify the synonym name or the object name.
- g. Choose *Finish*.

The tool creates the synonym along with the `.hbdsynonym` and `.hdbgrants` files in the same SAP HANA Database Module.

10. Define output columns.

- a. Select a view node.
- b. On the *Mapping* tab, select the column you want to add to the output.
- c. In the context menu, choose *Add To Output*.
- d. If you want to add all columns in a data source to the output, then from the context menu of the data source, select *Add To Output*.

## Note

Using keep flag property. The *Keep Flag* property on attribute columns influence the result set. Use *Keep Flag* property, for example, if you want to aggregate the measures by performing an SQL `GROUP BY` operation on them, even when they are not included in the query.

1. Select the view node.
2. On the *Mapping* tab, select an output column.
3. In the *Properties* section, set the value of *Keep Flag* property to *True*.

11. Define attributes and measures.

If you are creating a calculation view with data category as cube, to successfully activate the calculation view, you must specify at least one column as a measure.

- a. Select the *Semantics* node.
- b. On the *Columns* tab, select a column value.
- c. In the *Type* dropdown list, select *Measure* or *Attribute*.

If the data category is set to Cube, an additional aggregation column is available to specify the aggregation type for measures.

12. (Optional) View modeler objects in the outline pane.

Use the outline pane in SAP Web IDE for SAP HANA to obtain a quick overview of the modeler objects (view nodes and columns) in the calculation view.

- a. In the menu bar, choose  *View* > *Outline* .

The tool displays all modeler objects in the calculation view. You can also select an object in the outline pane, and navigate to the editor to identify where the object is used in the calculation view.

- b. In the right menu bar, choose  (Outline) to show or hide the outline pane.
13. Choose [Save](#) on the menu bar to save your calculation view.
14. Build an SAP HANA Database Module.
- The build process uses the design-time database artifacts to generate the corresponding runtime objects in the database catalog.
- a. From the module context menu, choose [Build](#).

## Next Steps

After creating a graphical calculation view, you can modify the output to your needs. The following table shows how you can modify the calculation view.

Table 39: Working With View Nodes

Requirement	Task to Perform
Query data from two data sources and combine records from both the data sources, based on a join condition, or to obtain language-specific data.	Create Joins
Combine the results of two or more data sources.	Create Unions
Partition the data for a set of partition columns, and perform an order by SQL operation on the partitioned data.	Create Rank Nodes
Execute any of the available graph operations on the graph workspace.	Create Graph Nodes
Use the intersect view node in calculation views to perform intersect set operations on two data sources.	Use Intersect Set Operation
Use the minus view node in calculation views to perform minus set operations on two data sources.	Use Minus Set Operation
Use table function view nodes to model table functions with both tabular input parameters and scalar input parameters.	Model Table Functions as View Nodes
Filter the output of view nodes.	Filter Output of View Nodes.
Use a form-based editor to create virtual tables.	Create Virtual Tables

Table 40: Working With Columns

Requirement	Task to perform
Count the number of distinct values for a set of attribute columns.	Create Counters
Create new output columns and calculate their values at runtime using an expression.	Create Calculated Columns
Assign semantic types to provide more meaning, and to attach information about the type of attributes and measures in calculation views.	Assign Semantics
Parameterize calculation views and execute them based on the values users provide at query runtime.	Create Input Parameters

Requirement	Task to perform
Filter the results based on the values that users provide to attributes at runtime.	Assign Variables
Create level hierarchies to organize data in reporting tools.	Create Level Hierarchies
Create parent-child hierarchies to organize data in reporting tools.	Create Parent-Child Hierarchies
Associate measures with currency codes and perform currency conversions.	Associate Measures with Currency
Associate measures with unit of measures and perform unit conversions.	Associate Measures with Unit of Measure
Define data masking for column values when modeling a calculation view.	Mask Column Values in Client Tools
Define default values for columns (both attributes and measures) when no value is provided during a SQL <code>INSERT</code> operation.	Handle Null Values in Columns
Group related measures together in a folder.	Group Related Measures

Table 41: Working With Calculation View Properties

Requirement	Task to perform
Filter the view data either using a fixed client value or using a session client set for the user.	Filter Data for Specific Clients
Invalidate or remove data from the cache after specific time intervals.	Invalidate Cached Content
Prevent use of a calculation view.	Deprecate Calculation Views
Execute time travel queries on calculation views	Enable Calculation Views for Time Travel Queries

## Related Information

[Preview Calculation View Output \[page 404\]](#)

[Defining Data Access Privileges \[page 406\]](#)

## 5.7.2 Preview Calculation View Output

After modeling calculation views based on your requirements, deploy them and preview its output within the same tool.

## Context

The tool provides multiple preview options. You can preview output data of calculation views in simple in tabular format, or you can preview output data in graphical representations, such as bar graphs, area graphs,

and pie charts. If your view has hierachal data structures, then the tool allows you to preview output data in hierarchical representations.

You can also export and download the output data to .csv files. The tool also allows you to preview the SQL query that the tool generates for an activated calculation view.

## Procedure

1. Start the SAP Web IDE for SAP HANA tool in a Web browser.
2. In the *Workspace* view, select the required calculation view to preview its output.
3. In the context menu, choose *Data Preview*.
4. Provide input parameter or variable value.

If you have defined any input parameters in the calculation view, provide the required input parameter values.

- a. Select the required operator.
- b. Provide values for the *From* and *To* fields based on the selected operator.
- c. Choose  (Raw Data).
- d. Select *Open Content*.

In the *Raw Data* tab, the tool displays the output data in tabular format.

5. Apply filters.
  - a. If you want to apply filters on columns and view the filtered output data, choose  (Add Filter).
  - b. Choose *Add Filters*.
  - c. Choose a column and define filter conditions.
6. Export output data, if required.

If you want to export the raw data output to a .csv file,

- a. In the toolbar, choose  (Download).
- b. In the *Delimiter* dropdown list, select the required delimiter that the tool must use to sperate the values in the .csv file.
- c. Choose *Download*.

7. View SQL query for the calculation view, if required.
  - a. If you want to view the SQL query that the tool generates for the deployed calculation view, in the toolbar, choose *SQL*.
  - b. If you want to view and execute the SQL query in SQL editor, choose  *SQL* (Edit SQL Statement in SQL Console).

8. Preview output of calculation views in graphical representations.

The tool supports bar graph, area graph, pie chart and table charts, and other graphical representations to preview the output of a calculation view.

- a. In the menu bar, choose *Analysis*.
- b. Configure the axis values by dragging and dropping the required attributes and measures to the *Label Axis* and the *Value Axis*.

The tool displays the output data in graphical representation. Select the required chart icons in the menu to view the output in different graphical representation.

- c. In the menu bar, choose  (Display Settings) to toggle legends and values.
9. Preview output of intermediate nodes.  
If you have activated the calculation view, you can also preview the output of any of its intermediate view nodes. This helps you to know the output data, which is passed to the higher view node levels.
  - a. Open the calculation view in the view editor.
  - b. Select the required intermediate view node.
  - c. In the context menu, choose *Data Preview*.

## Related Information

[Create Graphical Calculation Views \[page 400\]](#)

### 5.7.3 Defining Data Access Privileges

Use the analytic privilege editor in the SAP Web IDE for SAP HANA tool to create analytic privileges.

You create analytic privileges to grant different users access to different portions of data in the same view based on their business role. Within the definition of an analytic privilege, the conditions that control which data users see is defined using SQL.

Standard object privileges (SELECT, ALTER, DROP, and more) implement coarse-grained authorization at object level only. Users either have access to an object, such as a table, view or procedure, or they don't. While this is sufficient, there are cases when access to data in an object depends on certain values or combinations of values. Analytic privileges are used in the SAP HANA database to provide such fine-grained control at row level of which data individual users can see within the same view.

#### Example

Sales data for all regions is contained within one calculation view. However, regional sales managers should only see the data for their region. In this case, an analytic privilege could be modeled so that they can all query the view, but only the data that each user is authorized to see is returned.

# 6 Defining Web-based Data Access

SAP HANA extended application services (SAP HANA XS) provide applications and application developers with access to the SAP HANA database using a consumption model that is exposed via HTTP.

In addition to providing application-specific consumption models, SAP HANA XS also host system services that are part of the SAP HANA database, for example: search services and a built-in Web server that provides access to static content stored in the SAP HANA repository.

The consumption model provided by SAP HANA XS focuses on server-side applications written in JavaScript and making use of a powerful set of specially developed API functions. However, you can use other methods to provide access to the data you want to expose in SAP HANA. For example, you can set up an ODATA service.

## Related Information

[Defining OData v2 Services for XS Advanced JavaScript Applications \[page 408\]](#)

## 6.1 Maintaining OData Services in XS Advanced

Define OData services for your Java and JavaScript applications in XS advanced.

In XS advanced, you can create OData services that use either OData version 2.0 or OData version 4.0. OData version 2.0 is the basis for the XSODATA service infrastructure. Among other things, OData version 4.0 extends the query filters available, for example, `$select` and `$expand`.

**i** Note

The XSODATA infrastructure for OData services based on OData version 2 will no longer be extended or improved. For new OData services, it is recommended to move to OData version 4 in combination with CDS.

The OData version your XS advanced applications can consume is determined by the language you use to develop the XS advanced application, for example, Java or JavaScript.

- XS advanced Java applications

**⚠** Restriction

OData version 4.0 is currently supported in XS advanced only on the Java stack. It is not possible to consume OData v2 services from Java application in XS advanced.

- XS advanced JavaScript (and Node.js) applications

For JavaScript (and Node.js) applications, it is recommended to define an XS OData service (`myOdataV2Service.xsodata`) using OData version 2.0.

### **Restriction**

Although it is possible to consume an OData v4 service, it is not recommended; it would require adding a Java module to your XS advanced Multi-Target Application (MTA) and routing the OData requests to the Java component.

Table 42: OData Versions and XS Advanced Application Stacks

Application Language	OData v 2.0	OData v 4.0
Java	-	✓
JavaScript/Node.js	✓	-

## Related Information

[Defining OData v2 Services for XS Advanced JavaScript Applications \[page 408\]](#)

[Defining OData v4 Services for XS Advanced Java Applications \[page 463\]](#)

### 6.1.1 Defining OData v2 Services for XS Advanced JavaScript Applications

Create the OData service definitions consumed by your SAP HANA XS advanced JavaScript application.

An OData v2 service definition for your XS advanced JavaScript application is described in an “xsodata” artifact, for example, `myODataService.xsodata`; the definition for your OData version 2 service should be placed in the `js/src/odata/services/` folder of your JavaScript project module. The following example shows two OData services (`srv1.xsodata` and `srvN.xsodata`) consumed by a JavaScript application.

### **Caution**

The XSODATA infrastructure for OData services based on OData version 2 will no longer be extended or improved. If you are developing a new OData service, it is recommended to move to OData version 4 in combination with CDS.

### **Example**

#### **Sample Code**

```
JavaScript-AppName
|- db/
|   |- package.json
|   |- src/
|       |- .hdiconfig
|           \- mytable.hdbdd
|
|- web/
```

```

|   |- xs-app.json
|   |- package.json
|   \- resources/
|- js/
|   |- start.js
|   |- package.json
|   \- src/
|       \- odata/
|           |- resources/
|           \- services/
|               |- srv1.xsodata
|               \- srvN.xsodata      # XSOData v2 service resources
|               # XSOData v2 service resources
|               # XSOData v2 services
|               # XSOData v2 service definition
|               # XSOData v2 service definition
|- security/
\- xs-security.json
\- mta.yaml

```

## Exposing a Database Table

You can expose the database table db/src/mytable.hdcds as 'My Entity Set' as follows:

1. Create the project structure (as shown in the example above) in your local system.
2. Create the OData service definition file myservice.xsodata.
3. Expose the table as EntitySet in the xsodata file:

### Sample Code

```

service {
  "mytable" as "MyEntitySet";
}

```

4. Deploy your application with the client deployment tool.
5. Call your service from the client UI, for example, a Web browser:

You can call your OData service with: the following URI

**http://<myServer>:<port>/odata/services/<myService>.xsodata**

You can requests the metadata of the OData service using the following URI

**http://<myServer>:<port>/odata/services/<myService>.xsodata/\$metadata**

### Restriction

OData supports both the ATOM and JSON formats in request and response payloads. However, the Node.js module only support JSON. To specify JSON as the payload format, you can either add the following system-query option to the request "\$format=json" or add the following request header "Accept:json".

## Related Information

[Define the Data an OData Service Exposes \(XS Advanced\) \[page 410\]](#)

[Create an OData Service Definition \(XS Advanced\) \[page 410\]](#)

## 6.1.1.1 Define the Data an OData Service Exposes (XS Advanced)

An OData service exposes data stored in database tables or views as OData collections for analysis and display by client applications. However, first of all, you need to ensure that the tables and views to expose as an OData collection actually exist.

### Context

To define the data to expose using an OData service, you must perform the following two steps:

### Procedure

1. Create a simple database table to expose with an OData service.
2. Create a simple database view to expose with an OData service. This step is optional; you can expose tables directly. In addition, you can create a calculation view as a modeling view.

## 6.1.1.2 Create an OData Service Definition (XS Advanced)

The OData service definition is a configuration file you use to specify which data (for example, views or tables) is exposed as an OData collection for analysis and display by client applications.

### Context

An OData service for SAP HANA XS Advanced is defined in a text file with the file extension `.xsodata`, for example, `OdataSrvDef.xsodata`. It must contain at least the entry service `{}`, which would generate an operational OData service with an empty service catalog and an empty metadata file.

### Procedure

1. Create the file that will contain your OData service definition.

An OData service definition is described in an `xsodata` artifact, for example, `myODataService.xsodata`; the service definition should be placed in the `services/` folder of your OData project structure.

2. Define the OData service.

The OData service definition uses the OData Service Definition Language (OSDL), which includes a list of keywords that you specify in the OData service-definition file to enable important features.

The following example shows a simple OData service definition exposing a simple table:

```
service {  
    "sample.odata::table" as "MyTable";  
}
```

This service definition exposes a table defined in the file `sample.odata::table.hdbtable` and creates an EntitySet for this entity named `MyTable`. The specification of an alias is optional. If omitted, the default name of the EntitySet is the name of the repository object file, in this example, `table`.

3. Save the OData service definition.

### 6.1.1.2.1 OData Service Definitions (XS Advanced)

The OData service definition is the mechanism you use to define what data to expose with OData, how, and to whom. Data exposed as an OData collection is available for analysis and display by client applications, for example, a browser that uses functions provided by an OData client library running on the client system.

To expose information by means of OData to applications using SAP HANA XS, you must define database views that provide the data with the required granularity. Then you create an OData service definition, which is a file you use to specify which database views or tables are exposed as OData collections.

**i Note**

SAP HANA XS supports OData version 2.0, which you can use to send OData queries (for example, using the HTTP GET method). Language encoding is restricted to UTF-8.

An OData service for SAP HANA XS is defined in a text file with the file suffix `.xsodata`, for example, `OdataSrvDef.xsodata`. The file must contain at least the entry `service {}`, which would generate a completely operational OData service with an empty service catalog and an empty metadata file. However, usually you use the service definition to expose objects in the database catalog, for example: tables, SQL views, or calculation rules.

In the OData service-definition file, you can use the following ways to name the SAP HANA objects you want to expose by OData:

**i Note**

The syntax to use in the OData service-definition file to reference objects is dictated by the object type, for example, database catalog (runtime).

- **Database objects**

Expose an object using the object's database catalog (runtime) name. The support for database objects is mainly intended for existing or replicated objects that do not have a repository design-time representation. The following example shows how to include a reference to a table in an OData service definition using the table's `runtime` name.

```
service {  
    "myTable" as "MyTable";  
}
```

By default, all entity sets and associations in an OData service are writeable, that is they can be modified with a CREATE, UPDATE, or DELETE requests. However, you can prevent the execution of a modification request by

setting the appropriate keyword (*create*, *update*, or *delete*) with the `forbidden` option in the OData service definition. The following example of an OData service definition for SAP HANA XS shows how to prevent any modification to the table `myTable` that is exposed by the OData service. Any attempt to make a modification to the indicated table using a CREATE, UPDATE, or DELETE request results in the HTTP response status 403 FORBIDDEN.

```
service {
    "sap.test::myTable"
        create forbidden
        update forbidden
        delete forbidden;
}
```

For CREATE requests, for example, to add a new entry to a table exposed by an OData service, you must specify an explicit key (not a generated key); the key must be included in the payload of the CREATE request. For UPDATE and DELETE requests, you do not need to specify the key explicitly (and if you do, it will be ignored); the key is already known, since it is essential to specify which entry in the table must be modified with the UPDATE or DELETE request.

**i Note**

Without any support for IN/OUT table parameters in SQLScript, it is not possible to use a sequence to create an entry in a table exposed by an OData service. However, you can use XS JavaScript exits to update a table with a generated value.

### 6.1.1.2.1.1 OData Service-Definition Type Mapping

During the activation of the OData service definition, SQL types defined in the service definition are mapped to EDM types according to a mapping table.

For example, the SQL type "Time" is mapped to the EDM type "EDM.Time"; the SQL type "Decimal" is mapped to the EDM type "EDM.Decimal"; the SQL types "Real" and "Float" are mapped to the EDM type "EDM.Single".

**i Note**

The OData implementation in SAP HANA Extended Application Services (SAP HANA XS) does not support all SQL types.

In the following example, the SQL types of columns in a table are mapped to the EDM types in the properties of an entity type.

```
{name = "ID"; sqlType = INTEGER; nullable = false;}, {name = "RefereeID";
```

```
sqlType = VARCHAR; nullable = true;}
```

```
<Property Name="ID" Type="Edm.Int32" Nullable="false"/> <Property
Name="RefereeID" Type="Edm.String" Nullable="true"/>
```

## Related Information

[OData Service Definition: SQL-EDM Type Mapping \(XS Advanced\) \[page 458\]](#)

### 6.1.1.2.1.2 OData Service-Definition Features

The OData service definition provides a list of keywords that you use in the OData service-definition file to enable important features. For example, the following list illustrates the most-commonly used features used in an OData service-definition and, where appropriate, indicates the keyword to use to enable the feature:

- **Aggregation**  
The results of aggregations on columns change dynamically, depending on the grouping conditions. As a result, aggregation cannot be done in SQL views; it needs to be specified in the OData service definition itself. Depending on the type of object you want to expose with OData, the columns to aggregate and the function used must be specified explicitly (**explicit aggregation**) or derived from metadata in the database (**derived aggregation**). Note that aggregated columns cannot be used in combination with the `$filter` query parameter, and aggregation is only possible with generated keys.
- **Association**  
Define associations between entities to express relationships between entities. With associations it is possible to reflect foreign key constraints on database tables, hierarchies and other relations between database objects.
- **Key Specification**  
The OData specification requires an `EntityType` to denote a set of properties forming a unique key. In SAP HANA, only tables can have a unique key, the primary key. All other (mostly view) objects require you to specify a key for the entity. The OData service definition language (OSDL) enables you to do this by denoting a set of existing columns or by generating a **local key**. Bear in mind that local keys are transient; they exist only for the duration of the current session and cannot be dereferenced.

#### i Note

OSDL is the language used to define a service definition; the language includes a list of keywords that you use in the OData service-definition file to enable the required features.

- **Parameter Entity Sets**  
You can use a special parameter entity set to enter input parameters for SAP HANA calculation views and analytic views. During activation of the entity set, the specified parameters are retrieved from the metadata of the calculation (or analytical) view and exposed as a new `EntitySet` with the name suffix "Parameters", for example "CalcViewParameters".
- **Projection**  
If the object you want to expose with an OData service has more columns than you actually want to expose, you can use SQL views to restrict the number of selected columns in the `SELECT`. However, for those cases where SQL views are not appropriate, you can use the **with** or **without** keywords in the OData service definition to **include** or **exclude** a list of columns.

### 6.1.1.3 Enable SAP OData Annotations

Add annotations to the OData v2\$metadata document.

#### Context

Version two of the OData protocol allows you to add annotations to the \$metadata document. The purpose of the annotations is to add information and hints to the EDM elements; the additional information can be used by the service clients (for example, an SAPUI5 application) to more clearly represent the service entities.

##### i Note

SAP defines its own specific set of OData annotations. However, not all of the SAP OData annotations are supported in XS OData for XS advanced model. For more information, see *SAP OData Annotations* in *Related Links* below.

#### Procedure

1. Open the OData service description, for example, myODataService.xsodata.
2. Enable OData annotations.

Use the `annotations { }` keyword to enable OData annotations, as illustrated in the following example:

##### Sample Code

Enabling OData Annotations

```
service ... {
    ...
}
annotations {
    enable OData4SAP;
}
```

##### ⚠ Restriction

The annotation configuration must be specified immediately after the “`service`” element.

If annotations are enabled for an XS OData service in XS advanced, the following XML name space is added for the `Edmx` element of the \$metadata document:

##### Sample Code

```
xmlns:sap="http://www.sap.com/Protocols/SAPData"
```

## Related Information

[SAP OData Annotations \[page 415\]](#)

[SAP Annotations for OData Version 2.0](#)

### 6.1.1.3.1 SAP OData Annotations

The OData v2 protocol allows the use of annotations in the metadata document.

The second version of the OData protocol allows you to add annotations to the metadata document; the purpose of the annotations is to add information and hints to the EDM elements. This structural metadata makes it easy to understand a service, and human-readable documentation can be directly embedded into the metadata document, helping developers consume an OData service. The added information can be used by the service clients (for example, and SAPUI5 application) to better represent the service entities. The annotations are divided into annotation elements and annotation attributes, represented as XML elements and XML attributes respectively.

Located at the OData service root, the service document lists the top-level resources that are available. The metadata document, on the other hand, describes the structure of all resources in the OData service; the metadata document is located at the address `$metadata` relative to the OData service root.

The OData protocol defines the concept of the annotations without adding any details concerning the particular names and values, which can or should be used for them. In addition, SAP defines its own specific set of annotations.

#### i Note

Not all of the SAP OData annotations are supported in XS OData for XS advanced model. For more information, see *SAP Annotations for OData Version 2.0* in *Related Links* below.

By default, the annotations are **not** serialized in the `$metadata` document of an XS OData service. For each of the annotations a default value is defined according to the information provided in *SAP Annotations for OData Version 2.0*. It is important to remember that an annotation is added to the metadata document only if the annotation's value differs from the default value.

To enable the annotations for a particular XS OData service, the following configuration has to be specified in the service document (the respective `.xsodata` service-definition file), as illustrated in the following example:

#### Sample Code

##### Enabling OData Annotations

```
service ... {
    ...
}
annotations {
    enable OData4SAP;
}
```

## Restriction

The annotation configuration must be specified immediately after the “`service`” element.

If annotations are enabled for an XS OData service in XS advanced, the following XML name space is added for the `Edmx` element of the `metadata` document:

## Sample Code

```
xmlns:sap="http://www.sap.com/Protocols/SAPData"
```

## Related Information

[SAP Annotations for OData Version 2.0](#) 

[Enable SAP OData Annotations \[page 414\]](#)

### 6.1.1.3.1.1 OData Entity Set Annotations

A list of the attributes that can be used as annotations in the entity `set` element (`edm:EntitySet`).

For XS OData services used in XS advanced, entity sets can be annotated with the following attributes:

- [sap:addressable \[page 416\]](#)  
Direct access to the specified entity set is permitted
- [sap:createable \[page 417\]](#)  
New entities can be created in the specified entity set
- [sap:updatable \[page 417\]](#)  
Existing entities in the specified entity set can be updated
- [sap:deletable \[page 418\]](#)  
Existing entities can be removed from the specified entity set

#### sap:addressable

Permit direct access to the specified entity set.

#### Default Value

“true”

#### Additional Supported Values

“false” if the entity set represents either a calculation view or input parameters for a calculation view.

## **sap:creatable**

Allow the creation of new entities in the specified entity set.

### **Default Value**

“true”

### **Additional Supported Values**

The following table shows the additional values that are supported for the attribute `sap:creatable` in the `edm:EntitySet` element and the corresponding requirements.

Table 43: Additional Supported Values

Parameter Value	Requirement
“false”	The “create forbidden” setting is defined for the entity set in the OData service definition ( <code>.xsodata</code> ) file
	The entity set represents a database view, for example, a table or a calculation view.
	The entity set represents input parameters for a calculation view.
	“aggregation” is defined for the entity set, for example, using the “aggregates always” expression in the OData service definition ( <code>.xsodata</code> ) file.

## **sap:updatable**

Allow changes and updates to existing entities in the specified entity set.

### **Default Value**

“true”

### **Additional Supported Values**

The following table shows the additional values that are supported for the attribute `sap:updatable` in the `edm:EntitySet` element and the corresponding requirements.

Table 44: Additional Supported Values

Parameter Value	Requirement
“false”	The “update forbidden” setting is defined for the entity set in the OData service definition ( <code>.xsodata</code> ) file
	The entity set represents a database view, for example, a table or a calculation view.
	The entity set represents input parameters for a calculation view.

Parameter Value	Requirement
	"aggregation" is defined for the entity set, for example, using the "aggregates always" expression in the OData service definition (.xsodata) file.
	A generated key is defined for the entity set.

## sap:deletable

Allow the deletion of entities from the specified entity set.

### Default Value

"true"

### Additional Supported Values

The following table shows the additional values that are supported for the attribute `sap:deletable` in the `edm:EntitySet` element and the corresponding requirements.

Table 45: Additional Supported Values

Parameter Value	Requirement
"false"	The "delete forbidden" setting is defined for the entity set in the OData service definition (.xsodata) file
	The entity set represents a database view, for example, a table or a calculation view.
	The entity set represents input parameters for a calculation view.
	"aggregation" is defined for the entity set, for example, using the "aggregates always" expression in the OData service definition (.xsodata) file.
	A generated key is defined for the entity set.

## Related Information

[SAP OData Annotations \[page 415\]](#)

[Enable SAP OData Annotations \[page 414\]](#)

[SAP Annotations for OData Version 2.0](#)

## 6.1.1.3.1.2 OData Entity-Type Annotations

A list of the attributes that can be used as annotations in the entity **type** element (`edm:EntityType`).

### sap:semantics

Describes the semantic of the entity type.

#### Default Value

Undefined.

#### Additional Supported Values

The following table shows the additional values that are supported for the attribute `sap:semantics` and the corresponding requirements.

Table 46: Additional Supported Values

Parameter Value	Requirement
"false"	The "delete forbidden" setting is defined for the entity set in the OData service definition (.xsodata) file
	The entity set represents a database view, for example, a table or a calculation view.
	The entity set represents input parameters for a calculation view.
	"aggregation" is defined for the entity set, for example, using the "aggregates always" expression in the OData service definition (.xsodata) file.
	A generated key is defined for the entity set.

### Related Information

[OData Entity Set Annotations \[page 416\]](#)

[Enable SAP OData Annotations \[page 414\]](#)

[SAP Annotations for OData Version 2.0](#) 

## 6.1.1.3.1.3 OData Annotations for Entity-Type Properties

A list of the supported annotations for properties of entity types.

- [sap:semantics \[page 420\]](#)

- [sap:parameter \[page 421\]](#)
- [sap:label \[page 421\]](#)
- [sap:filterable \[page 422\]](#)
- [sap:display-format \[page 422\]](#)
- [sap:aggregation-role \[page 423\]](#)
- [sap:unit \[page 423\]](#)
- [sap:filter-restriction \[page 423\]](#)

## sap:semantics

The semantic of the entity-type property.

### Default Value

Undefined; there is no defined default value.

### Additional Supported Values

The annotation is added only for the properties representing calculated attributes or input parameters in a calculation view. The annotation value corresponds to the semantic type of a calculated attribute or an input parameter. The semantic type of an **input parameter** is specified as a value of “type” attribute of “valueDomain” element, defined in a calculation view definition file (for example, myCalcView.hdbcalculationview), as illustrated in the following example:

#### Sample Code

```
<variable id="parameterID" parameter="true">
  <variableProperties datatype="VARCHAR" mandatory="true">
    <valueDomain type="Currency"/>
    ...
  </variableProperties>
</variable>
```

Semantic type of a **calculated attribute** is specified as a value of a “semanticType” attribute of “calculatedAttribute” element, defined in a calculation view definition file (for example, myCalcView.hdbcalculationview), as illustrated in the following example:

#### Sample Code

```
<calculatedAttribute id="attributeID" semanticType="currencyCode">
  ...
</calculatedAttribute>
```

The following table shows how the semantic type values of calculated attributes and input parameters are mapped to the values of the “sap:semantics” annotation:

#### ⚠ Restriction

Only those values listed in the following table are supported.

Table 47: Mapping Between Attribute Values and Annotation

Calculated Attribute Semantic Type	Input Parameter Semantic Type	sap:semantics value
currencyCode	Currency	currency-code
unitOfMeasure	UnitOfMeasure	unit-of-measure
date.businessDateFrom	-	dtstart
date.businessDateTo	-	dtend

## sap:parameter

A property is annotated with this annotation, if it represents a parameter.

### Default Value

Undefined; there is no defined default value.

### Additional Supported Values

The annotation is added only to the properties representing input parameters of a calculation view. The annotation value corresponds to the value of the “mandatory” attribute in the input parameter definition, defined in a calculation view definition file (`myCalcView.hdbcalculationview`).

#### Sample Code

```
<variable id="inputParameterId" parameter="true">
  <variableProperties datatype="INTEGER" mandatory="true">
  </variableProperties>
</variable>
```

The following table shows how the value of the “mandatory” attribute value is mapped to the value of the “sap:parameter” annotation:

Table 48: Mapping Between Attribute Values and Annotation

“mandatory” Attribute Value	sap:parameter Value
true	mandatory
false	optional

## sap:label

A short, human-readable text that is suitable for use in labels and captions in a user interface (UI).

### Default Value

Undefined; there is no defined default value.

## Additional Supported Values

The annotation is added only for the properties representing calculated attributes or input parameters of a calculation view. The annotation value corresponds to the “`defaultDescription`” attribute in the definition of an input parameter or a calculated attribute in the calculation view definition (`myCalcView.hdbcalculationview`) file.

### Sample Code

```
<variable id="inputParameterId" parameter="true">
    <descriptions defaultDescription="Input parameter label"/>
</variable>
<calculatedAttribute id="calcAttributeId">
    <descriptions defaultDescription="Calculated attribute label"/>
</calculatedAttribute>
```

## sap:filterable

Indicates if the property can be used with the “`$filter`” system query option.

### Default Value

“`true`”

## Additional Supported Values

The annotation is added with the “`false`” value if the property satisfies one of the following conditions:

- The property represents a generated key
- The property represents a measure attribute of a calculation view
- The property is used in the aggregation, defined as the “`aggregates always`” expression in the XS OData service-definition (`.xsodata`) file.

## sap:display-format

The format used to display the property.

### Default Value

Undefined; there is no defined default value.

## Additional Supported Values

The “`Date`” value can be used if the SQL `DATE` type is used for the property on the database side.

## sap:aggregation-role

The aggregation role of the property.

### Default Value

Undefined; there is no defined default value.

### Additional Supported Values

The “measure” value can be used if the property represents a measure attribute of a calculation view or is used in the aggregation that is defined in the “*aggregates always*” expression in the XS advanced OData service-definition (.xsodata) file.

## sap:unit

The name of a property in the context of the entity type containing the currency code or unit of measure for a numeric value of the current property.

### Default Value

Undefined; there is no defined default value.

### Additional Supported Values

The annotation is added only for the properties representing calculated attributes of a calculation view. The annotation value corresponds to the value of “attributeName” attribute of “unitCurrencyAttribute” element of the calculated attribute definition in myCalcView.hdbculationview file.

#### i Note

In the following example, the annotation value will be “currencyCodeAttribute”.

#### Sample Code

```
<calculatedAttribute id="attributeId">
    <unitCurrencyAttribute attributeName="currencyCodeAttribute"/>
</calculatedAttribute>
```

## sap:filter-restriction

Describes filter restrictions for the property, if any exist.

### Default Value

Undefined; there is no defined default value.

## Additional Supported Values

The annotation is added only for the properties representing input parameters of a calculation view. The annotation value corresponds to the values of “multiLine” and “type” attributes of the “selection” element of the input parameter definition in the calculation view definition file (for example, myCalcView.hdbcCalculationview).

### Sample Code

```
<variable id="parameterId" parameter="true">
  <variableProperties datatype="INTEGER">
    <selection multiLine="false" type="SingleValue"/>
  </variableProperties>
</variable>
```

The following table below shows the mapping between the “multiLine” and “type” attribute values and the value of the “sap:filter-restriction” annotation.

### ⚠ Restriction

Only those values specified in the following table are supported.

Table 49: Mapping Between Attribute Values and Annotation

multiLine	type	sap:filter-restriction
false	SingleValue	single-value
true	SingleValue	multi-value
false	Interval	interval

## Related Information

[Enable SAP OData Annotations \[page 414\]](#)

[OData Entity Set Annotations \[page 416\]](#)

[OData Entity-Type Annotations \[page 419\]](#)

[SAP Annotations for OData Version 2.0](#)

## 6.1.1.3.1.4 OData Navigation Property Annotations

A list of the annotations supported for use with relations between entities.

### Overview

- [sap:creatable \[page 425\]](#)
- [sap:filterable \[page 425\]](#)

#### sap:creatable

Indicates if new related entities can be created.

##### Default Value

“true”

##### Additional Supported Values

The annotation value is always “false” because neither “deep insert” (POST request payload containing data for both parent and related entity) nor POST request for the `.../EntitySet(key)/navPropertyName` URL is supported in XS OData for XS advanced.

#### sap:filterable

Indicates if the property can be used with the “\$filter” system query option.

##### Default Value

“true”

##### Additional Supported Values

The annotation value is always “false” because navigation properties cannot be used in a \$filter system query option in XS OData for XS advanced.

### Related Information

[OData Entity Set Annotations \[page 416\]](#)

[OData Entity-Type Annotations \[page 419\]](#)

[OData Annotations for Entity-Type Properties \[page 419\]](#)

## 6.1.1.3.1.5 OData Association Set Annotations

A list of the annotations supported for use with relations between association sets.

### Overview

- [sap:creatable \[page 426\]](#)
- [sap:updatable \[page 426\]](#)
- [sap:deletable \[page 427\]](#)

### sap:creatable

Allow the creation of relations represented by the association set.

#### Default Value

“true”

#### Additional Supported Values

“false” if the association set connects entity sets for calculation view results and calculation view input parameters.

### sap:updatable

Allow changes and updates to relations represented by the association set.

#### Default Value

“true”

#### Additional Supported Values

“false” if the association set connects entity sets for calculation view results and calculation view input parameters.

## sap:deletable

Allow the deletion of relations represented by the association set.

### Default Value

“true”

### Additional Supported Values

“false” if the association set connects entity sets for calculation view results and calculation view input parameters.

## Related Information

[OData Entity Set Annotations \[page 416\]](#)

[OData Entity-Type Annotations \[page 419\]](#)

[OData Annotations for Entity-Type Properties \[page 419\]](#)

[OData Navigation Property Annotations \[page 425\]](#)

[SAP Annotations for OData Version 2.0](#) 

## 6.1.1.4 OData Service-Definition Examples (XS Advanced)

The OData service definition describes how data exposed in an end point can be accessed by clients using the OData protocol.

Each of the examples listed below is explained in a separate section. The examples show how to use the OData Service Definition Language (OSDL) in the OData service-definition file to generate an operational OData service that enables clients to use SAP HANA XS to access the OData end point you set up.

- Empty Service
- Namespace Definition
- Object Exposure
- Property Projection
- Key Specification
- Associations
- Aggregation
- Parameter Entity Sets
- Nullable Properties

### Restriction

OData supports both the ATOM and JSON formats in request and response payloads. However, the Node.js module only support JSON. To specify JSON as the payload format, you can either add the following system-query option to the request “\$format=json” or add the following request header “Accept:json”.

## Related Information

[OData Empty Service \[page 428\]](#)  
[OData Namespace Definition \[page 429\]](#)  
[OData Object Exposure \[page 430\]](#)  
[OData Property Projection \[page 430\]](#)  
[OData Key Specification \[page 431\]](#)  
[OData Associations \[page 434\]](#)  
[OData Aggregation \[page 438\]](#)  
[OData Parameter Entity Sets \[page 439\]](#)  
[OData ETag Support \[page 441\]](#)  
[OData Nullable Properties \[page 442\]](#)  
[OData Configurable Cache Settings \[page 442\]](#)

### 6.1.1.4.1 OData Empty Service

An OData service for SAP HANA XS is defined by a text file containing at least the following line:

Service definition sample.odata:empty.xsodata

```
service {}
```

A service file with the minimal content generates an empty, completely operational OData service with an empty service catalog and an empty metadata file:

**i Note**

- Examples and graphics are provided for illustration purposes only; some URLs may differ from the ones shown.

`http://<myHANAServer>:<port>/odata/services/<myService>.xsodata`

```
{
  "d" : {
    "EntitySets" : []
  }
}
```

`http://<myHANAServer>:<port>/odata/services/<myService>.xsodata/$metadata`

```

<edmx:Edmx Version="1.0"
xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx">
  <edmx:DataServices m:DataServiceVersion="2.0"
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">
    <Schema Namespace="sample.odata.empty"
    xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
    xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
    xmlns="http://schemas.microsoft.com/ado/2007/05/edm">
      <EntityContainer Name="empty" m:IsDefaultEntityContainer="true"/>
    </Schema>
  </edmx:DataServices>
</edmx:Edmx>

```

An empty service metadata document consists of one Schema containing an empty EntityContainer. The name of the EntityContainer is the name of the .xsodata file, in this example "empty".

### 6.1.1.4.2 OData Namespace Definition

Every .xsodata file must define its own namespace by using the namespace keyword:

Service definition sample.odata:namespace.xsodata

```
service namespace "my.namespace" {}
```

The resulting service metadata document has the specified schema namespace:

**i** Note

Examples and graphics are provided for illustration purposes only; some URLs may differ from the ones shown.

```

http://<myHANAServer>:<port>/odata/services/<myService>.xsodata/$metadata

<edmx:Edmx Version="1.0"
xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx">
  <edmx:DataServices m:DataServiceVersion="2.0"
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">
    <Schema Namespace="my.namespace"
    xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
    xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
    xmlns="http://schemas.microsoft.com/ado/2007/05/edm">
      <EntityContainer Name="namespace" m:IsDefaultEntityContainer="true"/>
    </Schema>
  </edmx:DataServices>
</edmx:Edmx>

```

### 6.1.1.4.3 OData Object Exposure

In the examples provided to illustrate object exposure, the following definition of a table applies:

Table definition sample.odata:table.hdbtable

```
COLUMN TABLE "sample.odata::table" (
    "ID" INTEGER,
    "Text" NVARCHAR(1000),
    "Time" TIMESTAMP,
    PRIMARY KEY ("ID")
);
```

### Database Objects

Similar to the exposure of an object by using the repository design-time name is the exposure by the database name:

Service definition sample.odata:db.xsodata

```
service {
    "sample.odata::table" as "MyTable";
}
```

The service exposes the same table by using the database catalog name of the object and the name of the schema where the table is created in.

### 6.1.1.4.4 OData Property Projection

If the object you want to expose with an OData service has more columns than you actually want to expose, you can use SQL views to restrict the number of selected columns in the SELECT.

Nevertheless, SQL views are sometimes not appropriate, for example with calculation views, and for these cases we provide the possibility to restrict the properties in the OData service definition in two ways. By providing an including or an excluding list of columns.

#### Including Properties

You can specify the columns of an object that have to be exposed in the OData service by using the with keyword. Key fields of tables must not be omitted.

Service definition sample.odata:with.xsodata

```
service {
    "sample.odata::table" as "MyTable" with ("ID","Text");
}
```

The resulting `EntityType` then contains only the properties derived from the specified columns:

**i Note**

Examples and graphics are provided for illustration purposes only; some URLs may differ from the ones shown.

```
http://<myHANAServer>:<port>/odata/services/<myService>.xsodata/$metadata

<EntityType Name="MyTableType">
  <Key>
    <PropertyRef Name="ID"/>
  </Key>
  <Property Name="ID" Type="Edm.Int32" Nullable="false"/>
  <Property Name="Text" Type="Edm.String" Nullable="true" MaxLength="1000"/>
</EntityType>
```

## Excluding Properties

The opposite of the `with` keyword is the `without` keyword, which enables you to specify which columns you do NOT want to expose in the OData service:

Service definition `sample.odata:without.xsodata`

```
service {
  "sample.odata::table" as "MyTable" without ("Text", "Time");
}
```

The generated `EntityType` then does NOT contain the properties derived from the specified columns:

```
http://<myHANAServer>:<port>/odata/services/<myService>.xsodata/$metadata

<EntityType Name="MyTableType">
  <Key>
    <PropertyRef Name="ID"/>
  </Key>
  <Property Name="ID" Type="Edm.Int32" Nullable="false"/>
</EntityType>
```

## 6.1.1.4.5 OData Key Specification

The OData specification requires an `EntityType` to denote a set properties forming a unique key. In HANA only tables may have a unique key, the primary key. For all other (mostly view) objects you need to specify a key for the entity.

In OSDL, you can specify a key for an entity/object by denoting a set of existing columns or by generating a key.

### **i** Note

Key attributes are not evaluated.

For the examples illustrating key specification, we use the following SQL view, which selects all data from the specified table.

View definition `sample.odata:view.hdbview`

```
{  
    VIEW "sample.odata::view" as select * from "sample.odata::table"  
}
```

## Existing Key Properties

If the object has set of columns that may form a unique key, you can specify them as key for the entity. These key properties are always selected from the database, no matter if they are omitted in the `$select` query option. Therefore explicit keys are not suitable for calculation views and analytic views as the selection has an impact on the result.

Service definition `sample.odata:explicitkeys.xsodata/$metadata`

```
service {  
    "sample.odata::view" as "MyView" key ("ID", "Text");  
}
```

The metadata document for the exposure of the view above is almost equal to the metadata document for repository objects. Only the key is different and consists now of two columns:

### **i** Note

Examples and graphics are provided for illustration purposes only; some URLs may differ from the ones shown.

`http://<myHANAServer>:<port>/odata/services/<myService>.xsodata/$metadata`

```

<edmx:Edmx Version="1.0" xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx">
  <edmx:DataServices m:DataServiceVersion="2.0"
    xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">
    <Schema Namespace="sample.odata.explicitkeys"
      xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
      xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
      xmlns="http://schemas.microsoft.com/ado/2007/05/edm">
      <EntityType Name="MyViewType">
        <Key>
          <PropertyRef Name="ID"/>
          <PropertyRef Name="Text"/>
        </Key>
        <Property Name="ID" Type="Edm.Int32" Nullable="false"/>
        <Property Name="Text" Type="Edm.String" Nullable="true" MaxLength="1000"/>
        <Property Name="Time" Type="Edm.DateTime" Nullable="true"/>
      </EntityType>
      <EntityContainer Name="explicitkeys" m:IsDefaultEntityContainer="true">
        <EntitySet Name="MyView"
          EntityType="sample.odata.explicitkeys.MyViewType"/>
      </EntityContainer>
    </Schema>
  </edmx:DataServices>
</edmx:Edmx>

```

### Caution

The OData infrastructure cannot check whether your specified keys are unique, so be careful when choosing keys.

## Generated Local Key

For objects that do not have a unique key in their results, for example, calculation views or aggregated tables, you can generate a locally valid key. This key value numbers the results starting with 1 and is not meant for dereferencing the entity; you cannot use this key to retrieve the entity. The key is valid only for the duration of the current session and is used only to satisfy OData's need for a unique ID in the results. The property type of a generated local key is Edm.String and cannot be changed.

Service definition sample.odata:generatedkeys.xsodata

```

service {
  "sample.odata::view" as "MyView" key generate local "GenID";
}

```

[http://<myHANAServer>:<port>/odata/services/<myService>.xsodata/\\$metadata](http://<myHANAServer>:<port>/odata/services/<myService>.xsodata/$metadata)

```

<EntityType Name="MyViewType">
  <Key>
    <PropertyRef Name="GenID"/>
  </Key>
  <Property Name="GenID" Type="Edm.String" Nullable="false" MaxLength="2147483647"/>
  <Property Name="ID" Type="Edm.Int32" Nullable="false"/>
  <Property Name="Text" Type="Edm.String" Nullable="true" MaxLength="1000"/>
  <Property Name="Time" Type="Edm.DateTime" Nullable="true"/>
</EntityType>
<EntityContainer Name="generatedkeys" m:IsDefaultEntityContainer="true">
  <EntitySet Name="MyView" EntityType="sample.odata.generatedkeys.MyViewType"/>
</EntityContainer>

```

As a consequence of the transient nature of generated local keys, it is not possible to define navigation properties on these entities or use them in filter or order by conditions.

## 6.1.1.4.6 OData Associations

You can define associations between entities to express relationships between entities. With associations it is possible to reflect foreign key constraints on database tables, hierarchies and other relations between database objects. OSDL supports simple associations, where the information about the relationship is stored in one of the participating entities, and complex associations, where the relationship information is stored in a separate association table.

Associations themselves are freestanding. On top of them you can specify which of the entities participating in the relationship can navigate over the association to the other entity by creating NavigationProperties.

For the examples used to illustrate OData associations, we use the tables `customer` and `order`:

Table definition: `sample.odata:customer.hdbtable`

```

COLUMN TABLE "sample.odata::customer" (
  "ID" INTEGER NOT NULL,
  "OrderID" INTEGER,
  PRIMARY KEY ("ID")
);

```

Table definition: `sample.odata:order.hdbtable`

```

COLUMN TABLE "sample.odata::order" (
  "ID" INTEGER NOT NULL,
  "CustomerID" INTEGER,
  PRIMARY KEY ("ID")
);

```

There is one relationship `order.CustomerID` to `customer.ID`.

## Simple Associations

The definition of an association requires you to specify a name, which references two exposed entities and whose columns keep the relationship information. To distinguish the ends of the association, you must use the keywords `principal` and `dependent`. In addition, it is necessary to denote the multiplicity for each end of the association.

Service definition: `sample.odata:assocsimple.xsodata`

```
service {
    "sample.odata::customer" as "Customers";
    "sample.odata::order" as "Orders";
    association "Customer_Orders" with referential constraint principal
    "Customers"("ID") multiplicity "1" dependent "Orders"("CustomerID") multiplicity
    "*";
}
```

The association in the example above with the name `Customer_Orders` defines a relationship between the table `customer`, identified by its EntitySet name `Customers`, on the `principal` end, and the table `order`, identified by its entity set name `Orders`, on the `dependent` end. Involved columns of both tables are denoted in braces (`{}`) after the name of the corresponding entity set. The `multiplicity` keyword on each end of the association specifies their cardinality - in this example, one-to-many.

The `with referential constraint` syntax ensures that the referential constraint check is enforced at design time, for example, when you activate the service definition in the SAP HANA repository. The referential constraint information appears in the metadata document.

### i Note

SAP strongly recommends that you use the `with referential constraint` syntax.

The number of columns involved in the relationship must be equal for both ends of the association, and their order in the list is important. The order specifies which column in one table is compared to which column in the other table. In this simple example, the column `customer.ID` is compared to `order.CustomerID` in the generated table join.

As a result of the generation of the service definition above, an `AssociationSet` named `Customer_Orders` and an `Association` with name `Customer_OrdersType` are generated:

`http://<myHANAServer>:<port>/odata/services/<myService>.xsodata/$metadata`

```

<EntityType Name="CustomersType">
  <Key>
    <PropertyRef Name="ID"/>
  </Key>
  <Property Name="ID" Type="Edm.Int32" Nullable="false"/>
  <Property Name="RecruitID" Type="Edm.String" Nullable="true" MaxLength="1"/>
</EntityType>
<EntityType Name="OrdersType">
  <Key>
    <PropertyRef Name="ID"/>
  </Key>
  <Property Name="ID" Type="Edm.Int32" Nullable="false"/>
  <Property Name="CustomerID" Type="Edm.Int32" Nullable="false"/>
</EntityType>
<Association Name="Customer_OrdersType">
  <End Type="sample.odata.assocsimple.CustomersType" Role="CustomersPrincipal" Multiplicity="1"/>
  <End Type="sample.odata.assocsimple.OrdersType" Role="OrdersDependent" Multiplicity="*"/>
  <ReferentialConstraint>
    <Principal Role="CustomersPrincipal">
      <PropertyRef Name="ID"/>
    </Principal>
    <Dependent Role="OrdersDependent">
      <PropertyRef Name="CustomerID"/>
    </Dependent>
  </ReferentialConstraint>
</Association>
<EntityContainer Name="assocsimple" m:IsDefaultEntityContainer="true">
  <EntitySet Name="Customers" EntityType="sample.odata.assocsimple.CustomersType"/>
  <EntitySet Name="Orders" EntityType="sample.odata.assocsimple.OrdersType"/>
  <AssociationSet Name="Customer_Orders"
    Association="sample.odata.assocsimple.Customer_OrdersType">
    <End Role="CustomersPrincipal" EntitySet="Customers"/>
    <End Role="OrdersDependent" EntitySet="Orders"/>
  </AssociationSet>
</EntityContainer>

```

The second association is similar to the first one and is shown in the following listing:

```

association "Customer Recruit" with referential constraint principal
"Customers"("ID") multiplicity "1" dependent "Customers"("RecruitID")
multiplicity "*";

```

## Complex Associations

For the following example of a complex association, an additional table named `knows` is introduced that contains a relationship between customers.

Table definition: `sample.odata:knows.hdbtable`

```

COLUMN TABLE "sample.odata::knows" (
  "KnowingCustomerID" INTEGER NOT NULL,
  "KnowCustomerID" INTEGER NOT NULL,
  PRIMARY KEY ("KnowingCustomerID", "KnowCustomerID")
);

```

Relationships that are stored in association tables such as `knows` can be similarly defined as simple associations. Use the keyword `over` to specify the additional table and any required columns.

Service definition: `sample.odata:assoccomplex.xsodata`

```
service {
    "sample.odata::customer" as "Customers";
    "sample.odata::order" as "Orders";
    association "Customer_Orders"
        principal "Customers"("ID") multiplicity "*"
        dependent "Customers"("ID") multiplicity "*"
        over "sample.odata::knows" principal ("KnowingCustomerID") dependent
        ("KnownCustomerID");
}
```

With the keywords `principal` and `dependent` after `over` you can specify which columns from the association table are joined with the `principal` respectively `dependent` columns of the related entities. The number of columns must be equal in pairs, and their order in the list is important.

The generated `Association` in the metadata document is similar to the one created for a simple association except that the `ReferentialConstraint` is missing:

`http://<myHANAServer>:<port>/odata/services/<myService>.xsodata/$metadata`

```
<Association Name="Customer_OrdersType">
    <End Type="sample.odata.assoccomplex.CustomerType" Role="CustomersPrincipal" Multiplicity="*"/>
    <End Type="sample.odata.assoccomplex.CustomerType" Role="CustomersDependent" Multiplicity="*"/>
</Association>
<EntityContainer Name="assoccomplex" m:IsDefaultEntityContainer="true">
    <EntityType Name="Customers" EntityType="sample.odata.assoccomplex.CustomerType"/>
    <EntityType Name="Orders" EntityType="sample.odata.assoccomplex.OrderType"/>
    <AssociationSet Name="Customer_Orders"
        Association="sample.odata.assoccomplex.Customer_OrdersType">
        <End Role="CustomersPrincipal" EntitySet="Customers"/>
        <End Role="CustomersDependent" EntitySet="Customers"/>
    </AssociationSet>
</EntityContainer>
```

## Navigation Properties

By only defining an association, it is not possible to navigate from one entity to another. Associations need to be bound to entities by a `NavigationProperty`. You can create them by using the keyword `navigates`:

Service definition: `sample.odata:assocnav.xsodata`

```
service {
    "sample.odata::customer" as "Customers" navigates ("Customer_Orders" as
    "HisOrders");
    "sample.odata::order" as "Orders";
    association "Customer_Orders" principal "Customers"("ID") multiplicity "1"
    dependent "Orders"("CustomerID") multiplicity "*";
}
```

The example above says that it is possible to navigate from `Customers` over the association `Customer_Order` via the `NavigationProperty` named `"HisOrders"`.

The right association end is determined automatically by the entity set name. But if both ends are bound to the same entity, it is necessary to specify the starting end for the navigation. This is done by specifying either `from principal` or `from dependent` which refer to the `principal` and `dependent` ends in the association.

Service definition: `sample.odata:assocnavself.xsodata`

```
service {
    "sample.odata::customer" as "Customers"
        navigates ("Customer_Orders" as "HisOrders", "Customer_Recruit" as
    "Recruit" from principal);
    "sample.odata::order" as "Orders";
        association "Customer_Orders" principal "Customers"("ID") multiplicity "1"
dependent "Orders"("CustomerID") multiplicity "*";
        association "Customer_Recruit" principal "Customers"("ID") multiplicity
"1" dependent "Customers"("RecruitID") multiplicity "*";
}
```

In both cases a `NavigationProperty` is added to the `EntityType`.

`http://<myHANAServer>:<port>/odata/services/<myService>.xsodata/$metadata`

```
<EntityType Name="CustomersType">
<Key>
    <PropertyRef Name="ID"/>
</Key>
<Property Name="ID" Type="Edm.Int32" Nullable="false"/>
<Property Name="RecruitID" Type="Edm.String" Nullable="true" MaxLength="1"/>
<NavigationProperty Name="HisOrders"
Relationship="sample.odata.assocnavself.Customer_OrdersType" FromRole="CustomersPrincipal"
ToRole="OrdersDependent"/>
<NavigationProperty Name="Recruit"
Relationship="sample.odata.assocnavself.Customer_RecruitType" FromRole="CustomersPrincipal"
ToRole="CustomersDependent"/>
</EntityType>
```

### 6.1.1.4.7 OData Aggregation

The results of aggregations on columns change dynamically depending on the grouping conditions. This means that aggregation cannot be performed in SQL views; it needs to be specified in the OData service definition itself. Depending on the type of object to expose, you need to explicitly specify the columns to aggregate and the function to use or derived them from metadata in the database.

In general, aggregations do not have consequences for the metadata document. It just effects the semantics of the concerning properties during runtime. The grouping condition for the aggregation contain all selected non-aggregated properties. Furthermore, aggregated columns cannot be used in `$filter`, and aggregation is only possible with generated keys.

#### Derived Aggregation

The simplest way to define aggregations on columns in an object is to derive this information from metadata in the database. The only objects with this information are calculation views and analytic views. For all other

object types, for example, tables and SQL views, the activation will not work. To cause the service to use derived information, you must specify the keywords *aggregates always*, as illustrated in the following example:

```
service {
    "sample.odata::calc" as "CalcView"
        keys generate local "ID"
        aggregates always;
}
```

## Explicit Aggregation

The example for the explicit aggregation is based on the following table definition:  
sample.odata:revenues.hdbtable

```
COLUMN TABLE "sample.odata::revenues" (
    "Month" INTEGER NOT NULL,
    "Year" INTEGER NOT NULL,
    "Amount" INTEGER,
    PRIMARY KEY ("Month", "Year")
);
```

You can aggregate the columns of objects (without metadata) that are necessary for the derivation of aggregation by explicitly denoting the column names and the functions to use, as illustrated in the following example of a service definition: sample.odata:aggrexpl\_xsodata

```
service {
    "sample.odata::revenues" as "Revenues"
        keys generate local "ID"
        aggregates always (SUM of "Amount");
}
```

The results of the entity set `Revenues` always contain the aggregated value of the column `Amount`. To extract the aggregated revenue amount per year, add `$select=Year,Amount` to your requested URI.

### 6.1.1.4.8 OData Parameter Entity Sets

SAP HANA calculation views can interpret input parameters. For OData, these parameters can be entered by using a special parameter *entity set*.

Parameter entity sets can be generated for calculation views by adding *parameters via entity* to the entity, as illustrated in the following service-definition example:

```
service {
    "sample.odata::calc" as "CalcView"
        keys generate local "ID"
        parameters via entity;
}
```

During loading of the service, parameters specified in `sample.odata/calc.calculationview` are retrieved from the metadata of the calculation view and exposed as a new EntitySet named after the entity set name and the suffix `Parameters`, for example, `CalcViewParameters`. A NavigationProperty named `Results` is generated to retrieve the results from the parameterized call.

The name of the generated parameter entity set and the navigation property can be customized, as illustrated in the following service-definition example:

```
service {
    "sample.odata::calc" as "CalcView"
        keys generate local "ID"
        parameters via entity "CVParams" results property "Execute";
}
```

With the definition above, the name of the parameter entity set is `CVParams`, and the name of the `NavigationProperty` for the results is `Execute`.

## Navigating to Entities via Parameters

In an OData service definition, you can enable navigation between an entity and a parameterized entity. This feature is particularly useful if you need to have access to individual entries in a parameterized entity set, for example, a calculation view with parameters. If you need to access individual entries in an entity set that has parameters, you must expose the parameters as keys. If you do not need to have access to individual entries in an entity set, you can use the `key generate local` option to generate a pseudo key.

To enable navigation between an entity and a parameterized entity, you must perform the following steps:

1. Specify the parameters as part of the key of the target entity
2. Define the association between the entities

Enabling navigation between an entity and a parameterized entity is only possible if the parameters are part of the entity-type key in the OData service definition file. To make the parameters part of the key of the target entity, use the `via key` syntax, as illustrated in the following example:

```
service {
    "sap.test::calcview" key ("theKeyColumns") parameters via key and entity;
```

You also have to define an **association** between the source and target entities, for example, with additional entries introduced by the `via parameters` keyword, as illustrated in the following example:

```
service {
    "sap.test::table" as "Tab" navigates ("avp" as "ViewNav");
    "sap.test::calcview" as "View" key ("theKeyColumns") parameters via key and entity;

    association via parameters "avp"
        principal "Tab"("paramValue") multiplicity "*"
        dependent "View"("parameter") multiplicity "*";
}
```

### Note

The order of the property list of the dependent end is crucial.

The parameters you define in the dependent end of the association **must** be the first properties in the list. In addition, the parameters specified **must** be given in the same order as they are specified in the view, as illustrated in the following example:

```
association via parameters "avp"
  principal "Tab"("col1", "col2", "col3") multiplicity "*"
    dependent "View"("parameter1", "parameter2", "colA") multiplicity "*";
```

In the example immediately above, the principal “Tab” has three columns that contain the information that is required to navigate to the dependent “View” in the association.

- “col1”  
The value of “col1” should be set for “parameter1”
- “col2”  
The value of “col2” should be set for “parameter2”
- “col3”  
The parameter “col3” contains additional information that is not passed as an input parameter, but as part of a WHERE condition.

The generated SQL statement would look like the following:

```
select ... from "sap.test::calcview"(placeholder."$$parameter1$$=>?,
placeholder."$$parameter2$$=>?)
      where "colA"=?
```

#### i Note

This navigation property cannot be used in combination with the OData query options \$expand, \$filter and \$orderby.

### 6.1.1.4.9 OData ETag Support

This feature allows a service to define the fields that are to be included in the concurrency check.

You can now use entity tags (ETags) for optimistic concurrency control. If you choose to use this feature, then you must enable it per entity in the .xsodata file. Enabling this feature per entity allows for the concurrency control to be applied to multiple fields. The following code example provides information about how to do this.

#### Sample Code

```
service
{ entity "sap.test.odata.db.views::Etag" as "EtagAll"
  key ("KEY_00") concurrencytoken;
  entity "sap.test.odata.db.views::Etag" as "EtagNvarchar"
  key ("KEY_00") concurrencytoken ("NVARCHAR_01", "INTEGER_03");
}
```

If you specify `concurrencytoken` only, then all properties, except the key properties, are used to calculate the ETag value. If you provide specific properties, then only those properties are used for the calculation.

### Note

You **cannot** specify `concurrencytoken` on aggregated properties that use the `AVG` (average) aggregation method.

## 6.1.1.4.10 OData Nullable Properties

You can create a service to enable nullable properties in OData.

During the 'Create' phase, the XSODATA layer generates all entity properties automatically. Since the properties are not nullable, consumers of the code are forced to pass dummy values into them. However, OData supports `$filter` and `$orderby` conditions on the `null` value. This means that it is now possible to treat `null` as a value, if you enable it. You can enable this behavior for the entire service only, not per entity.

The following code example provides information about how you can do this.

### Sample Code

```
service {  
    ...  
}  
settings {  
    support null;  
}
```

If you enable this support, then `$filter` requests, such as `$filter=NVARCHAR_01 eq null`, are possible. Otherwise `null` is rejected with an exception. Of course, if you do not enable the support, then the default behavior applies. All `null` values are ignored in comparisons and the respective rows are removed from the result; this is common database behavior.

## 6.1.1.4.11 OData Configurable Cache Settings

You can create a service to configure the cache settings for the `$metadata` request to optimize performance.

When calling OData services, the services make repeated requests for the `$metadata` document. Since changes to the underlying entity definitions occurs rarely, SAP has enabled the option to configure caching for these `$metadata` documents. By configuring the cache, you can avoid many redundant queries to process the metadata.

The following code example provides information about how you can do this.

### Sample Code

```
service {  
    ...  
}  
settings {  
    metadata cache-control "no-store";  
}
```

```
    content cache-control "max-age=3600,must-revalidate";
}
```

## 6.1.1.4.12 Custom Exits for OData Requests

Define validation checks or modification operations during an XS OData write request.

The xsodata library supports custom exits to enable the application to define JavaScript functions which are called while processing the OData request. These exits can be implemented either as a JavaScript function or as a stored procedure. You can define exits for creating, updating, and deleting entity sets as well as for the database commit operation (for example, pre-commit and post-commit). It is not possible to define an exit for simple read requests.

### Sample Code

Exit definition in the xsodata Service-Definition file

```
service {
    "my_name_space::customers" as "Customers"
    keys("CustomerID")
    navigates("CustomerToProduct" as "Product" from principal)
    create forbidden
    update using "sap.abc.xyz:myExits.xsjslib::updateEntity"
        events( before "sap.abc.xyz:myLoggingInfrastructure.xsjslib::log" )
    delete forbidden;
}
```

## Exit Types

The following type of write exits are supported for OData write requests in SAP HANA XS:

- Validation Exit:

Validation exits allow the application to validate data either before or after the data is changed in the database. Any error returned from the application exit function stops any further processing of the request and a rollback is performed. For batch requests, the processing of further requests in the same change set is stopped.

- Modification Exit

Modification exits enable you to define custom logic that can be used to create, update, or delete an entry in an entity set. If defined, a modification exit is executed instead of the generic actions provided by the OData infrastructure. You use the `using` keyword to register the exit.

Table 50: XS OData Validation Exits

Write Request	Code Example	Explanation
Create	create events (before "<custom exit 1>", after "<custom exit 2>", ...)	The xsodata library calls the exit custom exit 1 before creating the entity, and the exit custom exit 2 after creating the entity
	create events (precommit "<custom exit 1>", postcommit "<custom exit 2>", ...)	The xsodata library calls the exit custom exit 1 before committing the data, and the exit custom exit 2 after committing
	create forbidden	Prohibits any create operation on the entity set
Update	update events (before "<custom exit 1>", after "<custom exit 2>", ...)	The xsodata library calls the exit custom exit 1 before creating the entity, and the exit custom exit 2 after updating the entity
	update events (precommit "<custom exit 1>", postcommit "<custom exit 2>", ...)	The xsodata library calls the exit custom exit 1 before committing the data, and the exit custom exit 2 after committing
	update forbidden	Prohibits any update operation on the entity set
Delete	delete events (before "<custom exit 1>", after "<custom exit 2>", ...)	The xsodata library calls the exit custom exit 1 before creating the entity, and the exit custom exit 2 after deleting the entity
	delete events (precommit "<custom exit 1>", postcommit "<custom exit 2>", ...)	The xsodata library calls the exit custom exit 1 before committing the data, and the exit custom exit 2 after committing
	delete forbidden	Prohibits the deletion of the entity

The following table lists the types of modification exits you can define with XS Odata:

Table 51: XS OData Modification Exits

Write Request	Code Example	Explanation
Create	create using "<custom exit>"	The xsodata library calls the exit <custom exit> instead of creating the entity directly
Update	update using "<custom exit>"	The xsodata library calls the exit <custom exit> instead of updating the entity directly
Delete	delete using "<custom exit>"	The xsodata library calls the exit <custom exit> instead of deleting the entity directly

XS Odata also allows you to combine both validation exits and modification exits in the same request, as illustrated in the following example:

#### Sample Code

```
create using "<custom exit>" events(before "<custom exit 1>", after "<custom exit 2>", ...)
```

## Exit Execution Order for XS OData Requests

For **single** OData requests, the following execution order applies for the custom exit:

Table 52: Exit Execution Order for Single OData Requests

Odata Request Element	Action	Exit Execution Order
Request #	Process the request	<ol style="list-style-type: none"><li>before</li><li>using (or implicit update happens)</li><li>after</li><li>precommit</li><li>Data is committed to the database</li><li>postcommit</li></ol>

The following table shows the execution order applied to custom exits for **batch** OData requests:

#### Note

This example assumes that there are two (2) requests and the batch request also includes a change set with 2 requests.

Table 53: Exit Execution Order for OData Batch Requests

ODATA Batch-Request Element	Action	Exit Execution Order
Request 1	Process the request	Single OData request execution order
Change set	Process request 1 in the change set	<ol style="list-style-type: none"><li>before</li><li>using</li><li>after</li></ol>
	Process request 2 in the change set	<ol style="list-style-type: none"><li>before</li><li>using</li><li>after</li></ol>

OData Batch-Request Element	Action	Exit Execution Order
	Process the commit operation	<ol style="list-style-type: none"> <li>1. precommit of the first request</li> <li>2. precommit of the second request</li> <li>3. Commit the requested data</li> <li>4. postcommit of the first request</li> <li>5. postcommit of the second request</li> </ol>
Request 2	Process request 2	Single OData request execution order

## Exit Implementation

You can choose to perform the exit in any of the following ways:

- [JavaScript Function \[page 446\]](#)
- [Node.js Function \[page 448\]](#)
- [Stored procedure \[page 448\]](#)

### XS Odata Custom Exit as an XS JavaScript Function

The JavaScript function needs to be implemented within an XS JavaScript library (\*.xsjslib artifact). There is no need to use the export mechanism of Node.js; a plain function is all you need, as illustrated in the following example::

#### Sample Code

```
//myLoggingInfrastructure.xsjslib
function my_create_before_exit(param) { }
```

The input parameter param is an object with the following content:

1. connection  
The SQL connection used in the OData request
2. beforeTableName  
The name of a temporary table with the single entry **before** the operation (UPDATE and DELETE events only)
3. afterTableName  
The name of a temporary table with the single entry **after** the operation (CREATE and UPDATE events only)

The content of the temporary tables can be fetched using the available SQL connection as illustrated in the following example:

## Sample Code

```
var stmt = 'select * from "' + param.afterTableName + '"',
xStmt = param.connection.prepareStatement( stmt ).executeQuery(),
data = xStmt._rows[0]; // {col1:val1, col2:val2, ...}
```

For the custom exit function, there are two possibilities to handle an error:

1. Throw an error

Use the function `throw "error message";`.

### Note

The response has the status code “500”, which cannot be modified.

2. Return an object

Return an object with the following structure:

## Sample Code

```
return{
    HTTP_STATUS_CODE: code, // e.g. 400, 500, etc.
    ERROR_MESSAGE: 'error message',
    DETAILS: 'more details'
};
```

### Note

This method enables you to add more details and control the status code of the response.

Within the xsodata files, exit functions are referenced with a colon (:) separated string, for example:

`<directory>:<file>::function`.

## Sample Code

```
sap.abc.xyz:myLoggingInfrastructure.xsjslib::my_create_before_exit
```

- `sap.abc.xyz`  
References a directory
- `myLoggingInfrastructure.xsjslib`  
References a XS JavaScript library (`xsjslib` file)
- `my_create_before_exit`  
References the exit function

The following example shows an XS JavaScript startup configuration with two root folders `/_SYS_REPO` and `/my_content`:

## Sample Code

```
//Start xsjs listener
xsjs({
    compress: false,
```

```

rootDirs: [
    path.join(__dirname, '_SYS_REPO')
    path.join(__dirname, 'my_content')
],
...}).listen(...)
```

In the example above, the file `myLoggingInfrastructure.xsjslib` must be stored in one of the following locations:

- `<app directory>/_SYS_REPO/sap/abc/xyz/myLoggingInfrastructure.xsjslib`
- `<app directory>/my_content/sap/abc/xyz/myLoggingInfrastructure.xsjslib`

### XS Odata Custom Exit as a Node.js Function

The JavaScript function needs to be implemented as node module as illustrated in the following example:

#### Sample Code

Custom OData Exit with Node.js

```
//myLoggingInfrastructure.js
exports.my_create_before_exit = function (param) {
```

The following example shows an XS JavaScript startup configuration with two root folders `/_SYS_REPO` and `/my_content`:

#### Sample Code

```
//Start xsjs listener
xsjs({
    compress: false,
    rootDirs: [
        path.join(__dirname, '_SYS_REPO')
        path.join(__dirname, 'my_content')
    ],
    ...}).listen(...)
```

For the example above, the file `myLoggingInfrastructure.js` must be stored in one of the following locations:

- `<app directory>/_SYS_REPO/sap/abc/xyz/myLoggingInfrastructure.xsjslib`
- `<app directory>/my_content/sap/abc/xyz/myLoggingInfrastructure.xsjslib`

### XS Odata Custom Exit as an SQLScript Stored Procedure

If you register a custom exit for an OData write request in the form of an SQLScript procedure, the signature of the registered script must follow specific rules, for example, depending on whether it is registered for `entity` or `link` write operations and depending on the operation itself. The signature must also have table-typed parameters for both input and output:

- Entity Write Operations
- Link Write Operations

For `entity` write operations, the methods registered for the CREATE operation are passed a table containing the new entry that must be inserted into the target table; the UPDATE operation receives the entity both

before and after the modification; the DELETE operation receives the entry that must be deleted. The table type of the parameters (specified with the `EntityType` keyword in the table below) corresponds to the types of the exposed entity.

Table 54: Entity Write Operations

Script Type	Create	Update	Delete
before, after, precommit, using	IN new EntityType, OUT error ErrorType	IN new EntityType, IN old EntityType, OUT error ErrorType	IN old EntityType, OUT error ErrorType
postcommit	IN new EntityType	IN new EntityType, IN old EntityType	IN old EntityType

For `link` write operations, all exits that are executed before the commit operation take two table-typed input parameters and one table-typed output parameter. The first parameter must correspond to the structure of the entity type at the **principal** end of the association; the second parameter must correspond to the **dependent** entity type.

Table 55: Link Write Operations

Script Type	Create, Update, Delete
before, after, precommit, using	IN principal PrincipalEntityType, IN dependent DependentEntityType, OUT error ErrorType
postcommit	IN principal PrincipalEntityType, IN dependent DependentEntityType

### i Note

Parameter types (`IN`, `OUT`) are checked during activation; the data types of table type columns are **not** checked.

The `OUT` parameter enables you to return error information. The first row in the `OUT` table is then serialized as `inner_error` in the error message. If no error occurs, the `OUT` table must remain empty. The structure of the table type `ErrorType` is not restricted. Any columns with special names `HTTP_STATUS_CODE` and `ERROR_MESSAGE` are mapped to common information in the OData error response. Content of columns with other names are serialized into the `inner_error` part of the error message that allows the return of custom error information.

Table 56: Error Message Content

Column Name	Type	Value Range	Error Response Information
<code>HTTP_STATUS_CODE</code>	INTEGER	400-417 (default: 400)	The HTTP response status code
<code>ERROR_MESSAGE</code>	NVARCHAR		The error message (<message>)

### i Note

If the SQLScript procedure throws an exception or writes an error messages to the `OUT` parameter table, the OData write operation is aborted. If more than one error message is added, only the content of the first row is returned in the resulting error message. Any scripts registered for the `postcommit` event must not

have an OUT parameter as the write operation cannot be aborted at such a late stage, even in the event of an error.

The following example illustrates a typical error-type table type, which is defined in a design-time file that must have the .hdbscript file suffix, for example `error.hdbscript`:

```
"sample.odata::error" AS TABLE (
    "HTTP_STATUS_CODE" INTEGER,
    "ERROR_MESSAGE" NVARCHAR(100),
    "DETAIL" NVARCHAR(100)
)
```

The following example shows how information is extracted from the error table if an error occurs during the execution of a create procedure for an OData write operation:

```
create procedure "sample.odata::createmethod"(IN new "sample.odata::table", OUT
error "sample.odata::error")
language sqlscript
sql security invoker as
    id INT;
begin
    select ID into id from :new;
    if :id < 1000 then
        error = select 400 as http_status_code,
                    'invalid ID' error_message,
                    'value must be >= 1000' detail from dummy;
    else
        insert into "sample.odata::table" values (:id);
    end if;
end;
```

## Related Information

[Tutorial: Creating a Validation Exit with SQLScript \[page 450\]](#)

[Tutorial: Creating a Modification Exit with XS JavaScript \[page 454\]](#)

### 6.1.1.4.13 Tutorial: Creating a Validation Exit with SQLScript

Use SQLScript to create a custom validation exit which runs server-side verification and data-consistency checks for an OData **update** operation.

## Prerequisites

To perform this task, you need the following objects:

- A table to expose, for example, `sample.odata::table.hdbscript`
- An error type, for example, `sample.odata::error.hdbscript`

## Context

In this tutorial, you see how to register an SQL script for an OData update operation; the script verifies, before the execution of the update operation, that the updated value is larger than the previous one. In the example shown in this tutorial, you define the table to be updated and a table type for the error output parameter of the exit procedure.

## Procedure

1. Create a table definition file using .hdbtable syntax.

The table to expose is defined in `sample.odata:table.hdbtable`, which should look like the following example:

```
COLUMN TABLE "table" (
    "ID" INTEGER NOT NULL,
    PRIMARY KEY ("ID")
);
```

2. Create a table type for the error output parameter of the exit procedure.

The error type file `sample.odata:error.hdbtabletype` should look like the following example:

```
"sample.odata::error" AS TABLE (
    "HTTP_STATUS_CODE" INTEGER,
    "ERROR_MESSAGE" NVARCHAR(100),
    "DETAIL" NVARCHAR(100)
)
```

3. Create a procedure that runs before the UPDATE event.

The procedure script for the `before UPDATE` event must have two table input parameters and one output parameter, for example:

- IN new "sample.odata::table"
- IN old "sample.odata::table"
- OUT error "sample.odata::error"

The procedure `sample.odata:beforeupdate.hdbprocedure` would look like the following example:

```
procedure "sample.odata::beforeupdate"
    (IN new "sample.odata::table", IN old "sample.odata::table", OUT error
    "sample.odata::error")
language sqlscript
sql security invoker as
    idnew INT;
    idold INT;
begin
    select ID into idnew from :new;
    select ID into idold from :old;
if :idnew <= :idold then
    error = select 400 as http_status_code,
        'invalid ID' error_message,
        'the new value must be larger than the previous' detail from dummy;
end if;
end;
```

4. Register the procedure to be executed at the `before` event.

You use the `update events (before "...")` keywords to register the procedure, as illustrated in the following example of an OData service file:

```
service {
    "sample.odata::table"
        update events (before "sample.odata::beforeupdate");
}
```

## 6.1.1.4.14 Tutorial: Creating a Validation Script with XS JavaScript

Use server-side JavaScript to write a script that you can register as a validation exit for an OData update operation for an entity.

### Prerequisites

To perform this task, you need the following objects:

- A table to expose for the OData create operation, for example, `sample.odata::table.hdbtable`

### Context

SAP HANA XS enables you to register custom code that handles the OData write operation. In this tutorial, you see how to use server-side JavaScript (XSJS) to write a script which you register as a validation exit for OData UPDATE operations for an entity. The script you register can verify the data to insert, continue to perform the intended operation, or return an error. To register an XS JavaScript function as an OData validation exit, perform the following steps:

### Procedure

1. Create a table definition file, for example, using the `.hdbtable` syntax.

The table you create in this step is used in the XS JavaScript function you create later in the tutorial. The table to expose is defined in `sample.odata::table.hdbtable`, which should look like the following example:

#### Sample Code

```
COLUMN TABLE "table" (
    "ID" INTEGER NOT NULL,
    PRIMARY KEY ("ID")
```

```
);
```

2. Create the XS JavaScript function that you want to register for OData modification events.

The function you register has one parameter, which can have the properties listed in the following table:

- connection
- beforeTableName
- afterTableName

### **Restriction**

The XS JavaScript function that you register for OData validation events must be created in the form of an XSJS library, for example, with the file extension `.xsjslib`; the XS JavaScript function you register for OData validation events cannot be an `.xsjs` file.

The XS JavaScript function `jsexit.xsjslib` should look something like the code illustrated in the following example:

#### **Sample Code**

XS JavaScript OData Validation Exit `jsexit.xsjslib`

#### **Sample Code**

```
function update_verify (param) {
    var after = param.afterTableName,
        stmt = 'select * from "' + after + '"';
    pStmt = param.connection.prepareStatement( stmt ),
    xStmt = pStmt.executeQuery(),
    cols = xStmt._columnNames,// [col1, col2, ...]
    data = xStmt._rows[0];// {col1:val1, col2:val2, ...}
    pStmt.close();
    if (data['ID']>50) {
        console.log('** Err');

        return { HTTP_STATUS_CODE: 400, ERROR_MESSAGE: 'ID is beyond the
range', DETAILS: 'ID can not be greater than 50'};
    } else {
        console.log('** Continue to update');
    }
}
```

3. Bind the XS JavaScript function to the entity specified in the OData service definition.

To bind the XS JavaScript function to a specified entity, use the syntax:

`<Package.Path>:<file>.<suffix>::<XSJS_FunctionName>`, as illustrated in the following example:

#### **Sample Code**

OData Service Definition

```
service {
    "sample.odata::table" as "Table"
    update events (before "sap.test:jsexit.xsjslib::update_verify");
}
```

## Related Information

[Tutorial: Creating a Validation Exit with SQLScript \[page 450\]](#)

[Tutorial: Creating a Modification Exit with XS JavaScript \[page 454\]](#)

[Defining OData v2 Services for XS Advanced JavaScript Applications \[page 408\]](#)

### 6.1.1.4.15 Tutorial: Creating a Modification Exit with XS JavaScript

You can use server-side JavaScript to write a script which you register as a modification exit for an OData `update` operation for an entity.

#### Prerequisites

To perform this task, bear in mind the following prerequisites:

- A table to expose for the OData create operation, for example, `sample.odata::table.hdbtable`

#### Context

SAP HANA XS enables you to register custom code that handles the OData write operation. In this tutorial, you see how to use server-side JavaScript (XSJS) to write a script which you register as a modification exit for OData `UPDATE` operations for an entity. The script you register verifies the data to insert, and throws a defined error string in the event of an error, for example, `Could not update table; check access permissions.`

To register an XS JavaScript function as an OData modification exit, perform the following steps:

#### Procedure

1. Create a table definition file, for example, using the `.hdbtable` syntax.

The table you create in this step is used in the XS JavaScript function you create later in the tutorial. The table to expose is defined in `sample.odata::table.hdbtable`, which should look like the following example:

```
COLUMN TABLE "table" (
    "ID" INTEGER NOT NULL,
    PRIMARY KEY ("ID")
);
```

2. Create the XS JavaScript function that you want to register for OData modification events.

**i Note**

The XS JavaScript function that you want to register for OData modification events must be created in the form of an XSJS library, for example, with the file extension `.xsjslib`; the XS JavaScript function cannot be an `.xsjs` file.

The function you register has one parameter, which can have the properties listed in the following table:

Table 57:

Property	Type	Description
connection	Connection	The SQL connection used in the OData request
beforeTableName	String	The name of a temporary table with the single entry before the operation (UPDATE and DELETE events only)
afterTableName	String	The name of a temporary table with the single entry after the operation (CREATE and UPDATE events only)

The XS JavaScript function `jsexit.xsjslib` could look like the following example:

```
function update_instead(param) {
    $.trace.debug("entered function");
    let before = param.beforeTableName;
    let after = param.afterTableName;
    let pStmt = param.connection.prepareStatement('select * from "' + after
+ '"';
    // ...
    if (ok) {
        // update
    } else {
        throw "an error occurred; check access privileges"
    }
}
```

3. Bind the XS JavaScript function to the entity specified in the OData service definition.

To bind the XS JavaScript function to a specified entity, use the syntax:

`<Package.Path>:<file>.<suffix>:<XSJS_FunctionName>` as illustrated in the following example:

```
service {
    "sample.odata::table" as "Table" update using
"sap.test:jsexit.xsjslib::update_instead";
}
```

## 6.1.1.5 OData Service Definition Language Syntax (XS Advanced)

The OData Service Definition Language (OSDL) provides a set of keywords that enable you to set up an ODATA service definition file that specifies what data to expose, in what way, and to whom.

The following list shows the syntax of the OData Service Definition Language (OSDL) in an EBNF-like format; conditions that apply for usage are listed after the table.

```
definition          ::=service [annotations]
service            ::= 'service' [namespace] body
namespace          ::= 'namespace' quotedstring
quotedstring       ::= quote string quote
string             ::=UTF8
quote              ::= ''
body               ::= '{' content '}'
content            ::=entry [content]
entry              ::= ( entity | association ) ';'
entity             ::=object [entityset] [with] [keys] [navigates]
[aggregates] [parameters] [modification]
object             ::=['entity'] ( repoobject | catalogobject )
repoobject         ::=quote repopackage '/' reponame '.' repoextension quote
repopackage        ::=string
reponame           ::=string
repoextension      ::=string
catalogobject      ::=catalogobjectschema '.' catalogobjectname
catalogobjectschema ::=quotedstring
catalogobjectname ::=quotedstring
entityset          ::= 'as' entitysetname
entitysetname      ::=quotedstring
with               ::= ( 'with' | 'without' ) propertylist
propertylist       ::=(' columnlist ')
columnlist         ::=columnname [',' columnlist]
columnname         ::=quotedstring
keys               ::= 'keys' ( keylist | keygenerated )
keylist            ::=propertylist
keygenerated       ::= 'generate' ( keygenlocal )
keygenlocal         ::= 'local' columnname
navigates          ::= 'navigates' '(' navlist ')'
navlist            ::=naventry [',' navlist]
naventry           ::=assocname 'as' navpropname [fromend]
assocname          ::=quotedstring
navpropname        ::=quotedstring
fromend            ::= 'from' ( 'principal' | 'dependent' )
aggregates         ::= 'aggregates' 'always' [aggregatestuple]
aggregatestuple   ::=(' aggregateslist ')
aggregateslist    ::=aggregate [',' aggregateslist]
aggregate          ::=aggregatefunction 'of' columnname
aggregatefunction ::= ( 'SUM' | 'AVG' | 'MIN' | 'MAX' )
parameters         ::= 'parameters' 'via' [parameterskeyand] 'entity'
[parameterentitysetname] [parametersresultsprop]
parameterskeyand  ::= 'key' 'and'
parameterentitysetname ::=quotedstring
parametersresultsprop ::= 'results' 'property' quotedstring
modification       ::= [create] [update] [delete]
create             ::= 'create' modificationspec
update             ::= 'update' modificationspec
delete             ::= 'delete' modificationspec
modificationspec  ::= ( modificationaction [events] | events | 'forbidden' )
modificationaction ::= 'using' action
action             ::=quotedstring
events             ::= 'events' '(' eventlist ')'
eventlist          ::=eventtype action [',' eventlist]
eventtype          ::= ( 'before' | 'after' | 'precommit' | 'postcommit' )
```

```

association           :=associationdef [assocrefconstraint] principalend
dependentend [ ( assocable | storage | modification ) ]
associationdef        :='association' assocname
assocrefconstraint   :='with' 'referential' 'constraint'
principalend          :='principal' end
dependentend          :='dependent' end
end                  :=endref multiplicity
endref                :=endtype [joinpropertieslist]
entitysetname          :=entitysetname
joinpropertieslist     :=(' joinproperties ')
joinproperties         :=columnlist
multiplicity           :='multiplicity' quote multiplicityvalue quote
multiplicityvalue      :=( '1' | '0..1' | '1..*' | '*' )
assocable              :='over' repoobject overprincipalend overdependentend
[modification]
overprincipalend       :='principal' overend
overdependentend       :='dependent' overend
overend                :=propertylist
storage                :=( nostorage | storageend [modification] )
nostorage              :='no' 'storage'
storageend              :='storage' 'on' ( 'principal' | 'dependent' )
annotations            :='annotations' annotationsbody
annotationsbody         :='{ annotationscontent }'
annotationscontent      :=annotationconfig [annotationscontent]
annotationconfig        :='enable' annotation
annotation             :='OData4SAP'

```

### **i Note**

Support for OData annotations is currently not available in SAP HANA XS Advanced.

## Conditions

The following conditions apply when using the listed keywords:

1. If the `namespace` is not specified, the schema namespace in the EDMX metadata document will be the repository package of the service definition file concatenated with the repository object name. E.g. if the repository design time name of the `.xsodata` file is `sap.hana.xs.doc/hello.xsodata` the namespace will implicitly be `sap.hana.xs.doc.hello`.
2. `keylist` must not be specified for objects of type 'table'. They must only be applied to objects referring a view type. `keygenerated` in turn, can be applied to table objects.
3. If the `entityset` is not specified in an entity, the EntitySet for this object is named after the repository object name or the `catalogobjectname`. For example, if object is "sap.hana.xs.doc/odata\_docu", then the `entitysetname` is implicitly set to `odata_docu`, which then can also be referenced in associations.
4. The `fromend` in a `naventry` must be specified if the `endtype` is the same for both the `principalend` and the `dependentend` of an association.
5. The number of `joinproperties` in the `principalend` must be the same as in the `dependentend`.
6. Ordering in the `joinproperties` of ends is relevant. The first `columnname` in the `joinproperties` of the `principalend` is compared with the first `columnname` of the `dependentend`, the second with the second, and so on.
7. The `overprincipalend` corresponds to the `principalend`. The number of properties in the `joinproperties` and the `overproperties` must be the same and their ordering is relevant. The same statement is true for the dependent end.

8. aggregates can only be applied in combination with keygenerated.
9. If aggregatestuple is omitted, the aggregation functions are derived from the database. This is only possible for calculation views and analytic views.
10. Specifying parameters is only possible for calculation views and analytic views.
11. The default parameterentitysetname is the entitysetname of the entity concatenated with the suffix "Parameters".
12. If the parametersresultsprop is omitted, the navigation property from the parameter entity set to the entity is called "Results".
13. Support for OData annotations is currently under development. For more information about the SAP-specific metadata annotations that become available with the enable OData4SAP statement in an .xsodata file, see the Related Links below. Note that not all annotations allowed by OData are supported by SAP HANA XS.

## Related Information

[SAP Annotations for OData](#)

[Open Data Protocol](#)

### 6.1.1.6 OData Service Definition: SQL-EDM Type Mapping (XS Advanced)

During the activation of the OData service definition, the SAP HANA SQL types are mapped to the required OData EDM types according to the rules specified in a mapping table.

The following mapping table lists how SAP HANA SQL types are mapped to OData EDM types during the activation of an OData service definition.

**i** Note

The OData implementation in SAP HANA XS supports only those SQL types listed in the following table.

Table 58: SAP HANA SQL to OData EDM Type Mapping

SAP HANA SQL Type	OData EDM Type
Time	Edm.Time
Date	Edm.DateTime
SecondDate	Edm.DateTime
LongDate	Edm.DateTime
Timestamp	Edm.DateTime
TinyInt	Edm.Byte
SmallInt	Edm.Int16

SAP HANA SQL Type	OData EDM Type
Integer	Edm.Int32
BigInt	Edm.Int64
SmallDecimal	Edm.Decimal
Decimal	Edm.Decimal
Real	Edm.Single
Float	Edm.Single
Double	Edm.Double
Varchar	Edm.String
NVarchar	Edm.String
Char	Edm.String
NChar	Edm.String
Binary	Edm.Binary
Varbinary	Edm.Binary

## Example SQL Type Mapping

The following examples shows how SAP HANA SQL types (name, integer, Varchar) of columns in a table are mapped to the OData EDM types in the properties of an entity type.

SAP HANA SQL:

```
{name = "ID"; sqlType = INTEGER; nullable = false;},
{name = "RefereeID"; sqlType = VARCHAR; nullable = true;}
```

The following example illustrates how the SAP HANA SQL types illustrated in the previous example are mapped to EDM types:

```
<Property Name="ID" Type="Edm.Int32" Nullable="false"/>
<Property Name="RefereeID" Type="Edm.String" Nullable="true"/>
```

### 6.1.1.7 OData Batch Requests (XS Advanced)

The OData standard allows the collection of multiple individual HTTP requests into one single batched HTTP request.

Clients using a defined OData service to consume exposed data can collect multiple, individual HTTP requests, for example, retrieve, create, update and delete (GET, POST, PUT, DELETE), in a single “batch” and send the batched request to the OData service as a single HTTP request. You can compile the batch request manually (by creating the individual requests in the batch document by hand) or automatically, for example, with an AJAX call that adds requests to a queue and loops through the queues to build the batch request. In both

cases, the OData standard specifies the syntax required for the header and body elements of a valid batch request document.

SAP HANA XS supports the OData \$batch feature out-of-the-box; there is nothing to configure in SAP HANA XS to use \$batch to perform operations in SAP HANA using an OData service. To understand how the \$batch feature works, you need to look at the following phases of the operation:

- Batch Request
- Batch Response

A batch request is split into two parts: the request header and the request body. The body of a batch request consists of a list of operations in a specific order where each operation either retrieves data (for example, using the HTTP GET command) or requests a change. A change request involves one or more insert, update or delete operations using the POST, PUT, or DELETE commands.

#### Note

A change request must not contain either a **retrieve** request or any nested change requests.

The batch request must contain a Content-Type header specifying the value “multipart/mixed” and a boundary ID `boundary=batch_#`; the batch boundary ID is then used to indicate the start of each batch request, as illustrated in the following example.

```
POST /service/$batch HTTP/1.1
Host: host
Content-Type: multipart/mixed;boundary=batch_8219-6895          // Define batch ID
--batch_8219-6895
  Content-Type: multipart/mixed; boundary=changeset_a4e3-a738 // Define
  changeset ID
  --changeset_a4e3-a738                                         // Changeset 1
start
  Content-Type: application/http
  Content-Transfer-Encoding: binary
  [PUT...]
  --changeset_a4e3-a738                                         // Changeset 2
start
  Content-Type: application/http
  Content-Transfer-Encoding: binary
  [POST...]
  --changeset_a4e3-a738--                                       // Changeset (all)
end
--batch_8219-6895
start
  Content-Type: application/http
  Content-Transfer-Encoding:binary
  [GET...]
--batch_8219-6895--                                           // Batch (all) end
```

Within the batch request, changeset is defined by another boundary ID (for example, `boundary=changeset_123`), which is then used to indicate the start and end of the change requests. The batch request must be closed, too.

#### Note

In the following example of a simple OData batch request, some content has been removed to emphasize the structure and layout.

```
POST http://localhost:8002/sap/sample/odata/syntax.xsodata/$batch HTTP/1.1
```

```

Host: localhost:8002
Connection: keep-alive
Content-Length: 471
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/30.0.1599.101 Safari/537.36
Cache-Control: no-cache
Content-Type: multipart/mixed; boundary=batch_123
Accept: */
Accept-Encoding: identity
Accept-Language: en-US,en;q=0.8
x-sap-request-language: en-US
--batch_123
Content-Type:multipart/mixed;boundary=changeset_456
Content-Transfer-Encoding:binary
--changeset_456
Content-Type:application/http
Content-Transfer-Encoding:binary
POST BatchSample HTTP/1.1
Content-Type:application/json
Content-Length:11
{"ID" : 14}
--changeset_456
Content-Type:application/http
Content-Transfer-Encoding:binary
POST BatchSample HTTP/1.1
Content-Type:application/json
Content-Length:11
Accept: application/json
{"ID" : 15}
--changeset_456--
--batch_123--

```

The batch response includes a response for each of the retrieve or change operations included in the corresponding batch request. The order of the responses in the response body must match the order of requests in the batch request. In the context of the batch response, the following is true:

- The response to a retrieve request is always formatted in the same way regardless of whether it is sent individually or as part of batch.
- The body of the collected response to a set of change-requests is one of the following:
  - A response for **all** the successfully processed change requests within the change set, in the correct order and formatted exactly as it would have appeared outside of a batch
  - A single response indicating the failure of the entire change set

The following example shows the form and syntax of the OData batch response to the request illustrated above.

```

HTTP/1.1 202 Accepted
content-type: multipart/mixed; boundary=0CDF14D90919CC8B4A32BD0E0B330DA10
content-length: 2029
content-language: en-US
cache-control: no-cache
expires: Thu, 01 Jan 1970 00:00:00 GMT
--0CDF14D90919CC8B4A32BD0E0B330DA10
Content-Type: multipart/form-data; boundary=0CDF14D90919CC8B4A32BD0E0B330DA11
Content-Length: 1843
--0CDF14D90919CC8B4A32BD0E0B330DA11
Content-Type: application/http
Content-Length: 1118
content-transfer-encoding: binary
HTTP/1.1 201 Created
Content-Type: application/atom+xml; charset=utf-8
location: http://localhost:8002/sap/sample/odata/syntax.xsodata/BatchSample(14) /
Content-Length: 943

```

```

<?xml version="1.0" encoding="utf-8" standalone="yes"?><entry xml:base="http://localhost:8002/sap/sample/odata/syntax.xsodata/" xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices" xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" xmlns="http://www.w3.org/2005/Atom"><id>http://localhost:8002/sap/sample/odata/syntax.xsodata/BatchSample(14)</id><title type="text"></title><author><name /></author><link rel="edit" title="BatchSample" href="BatchSample(14)"/><link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Ref" type="application/atom+xml;type=entry" title="Ref" href="BatchSample(14)/Ref"/></link><category term="sap.sample.odata.syntax.BatchSampleType" scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" /><content type="application/xml"><m:properties><d:ID m:type="Edm.Int32" m:null="true">14</d:ID><d:SELFID m:type="Edm.Int32" m:null="true"></d:SELFID></m:properties></content></entry>
--0CDF14D90919CC8B4A32BD0E0B330DA11
Content-Type: application/http
Content-Length: 427
Content-Transfer-Encoding: binary
HTTP/1.1 201 Created
Content-Type: application/json
location: http://localhost:8002/sap/sample/odata/syntax.xsodata/BatchSample(15)
Content-Length: 271
{"d": {"__metadata": {"uri": "http://localhost:8002/sap/sample/odata/syntax.xsodata/BatchSample(15)", "type": "sap.sample.odata.syntax.BatchSampleType"}, "ID": 15, "SELFID": null, "Ref": {"__deferred": {"uri": "http://localhost:8002/sap/sample/odata/syntax.xsodata/BatchSample(15)/Ref"}}}}
--0CDF14D90919CC8B4A32BD0E0B330DA11--
--0CDF14D90919CC8B4A32BD0E0B330DA10--

```

## OData Batch Requests in SAPUI5 Applications

If you are developing a UI client using SAPUI5, you can make use of the ODataModel tools to ensure that the data requests generated by the various UI controls bound to an OData service are collected and sent in batches. The SAPUI5 ODataModel toolset includes a large selection of tools you can use to configure the use of the OData batch feature, for example:

- `setUseBatch`  
Enable or disable batch processing for all requests (read and change)
- `addBatchChangeOperations`  
Appends the change operations to the end of the batch stack, which is sent with the `submitBatch` function
- `addBatchReadOperations`  
Appends the read operations to the end of the batch stack, which is sent with the `submitBatch` function
- `submitBatch`  
Submits the collected changes in the batch which were collected via `addBatchReadOperations` or `addBatchChangeOperations`.

## Related Information

[Open Data Protocol](#)

[SAPUI5 ODataModel Reference](#)

## 6.1.2 Defining OData v4 Services for XS Advanced Java Applications

Set up an OData (v4) service for use by a Java application in an XS advanced.

An OData v4 service for use with an XS advanced Java application is described in a design-time OData artifact and referenced in the corresponding CDS document (by means of an `@OData.publish` annotation) that exposes the specified tables or entities in the annotated context. The definition of your OData service should be placed in the `java/odata/v4/` folder of your Java application project module. The following example shows the project structure for an XS advanced Java application which consumes an OData version 4.0 service.

### Example

#### Sample Code

```
Java-AppName
|- db/
|   |- package.json
|   |- src/
|       |- .hdiconfig
|           \- myDataModel.hdbcards      # @OData v4 annotations
|- web/
|   |- xs-app.json                 # Route for Java Odata v4 module
|   |- package.json
|   \- resources/
|- java/                          # Enable DB/OData v4 support
|   |- package.json
|   |- src/
|       \- odata/v4/              # OData v4 service resources
|- security/                      # OData v4 user authorization scopes
|   \- xs-security.json
\- mta.yaml                         # Requires Java/OData v4 module
```

## Related Information

[Maintaining OData Services in XS Advanced \[page 407\]](#)

[Tutorial: Create OData v4 Services for XS Advanced Java Applications \[page 464\]](#)

## 6.1.2.1 Tutorial: Create OData v4 Services for XS Advanced Java Applications

Step-by-step instructions for creating OData v4 services in an XS advanced Java application.

### Context

To create an OData v4 service from an annotated CDS context using tools available in SAP Web IDE for SAP HANA, perform the following steps:

### Procedure

1. Create an MTA project.

For more information about using SAP Web IDE for SAP HANA to set up a Multi-Target Application (MTA), see *Related Links* below.

2. Add an HDB module to the newly created MTA project.

In SAP Web IDE for SAP HANA, right-click the root folder of the XS advanced MTA project and choose **New > SAP HANA Database Module**.

3. Add the required CDS files to the HDB module of your Java MTA.

You can add CDS files to your MTA's HDB module in either of the following ways:

- Import existing CDS artifacts from the local file system
- Create new CDS artifacts manually, for example, using the CDS Graphical Editor.

In SAP Web IDE for SAP HANA, expand the XS advanced application's DB module, right-click the `src/` folder and choose **New > CDS Artifact**. For more information about creating a CDS file, see *Related Links* below.

4. Annotate the required CDS contexts with the annotation `@OData.publish : true`.

This step is required to indicate which CDS context must be exposed as an OData v4 service.

#### Note

You can publish only a child context. If your CDS file contains both a parent as well as a child context, only the child context can be published. For more information, see *Related Links* below.

#### Sample Code

CDS Context exposed as an OData v4 Service

```
namespace "your"."namespace";
@OData.publish : true
context ContextName{
    entity MyEntity{
        key myID : Integer
```

```
    };
```

5. Create a Java module in the same MTA.

In SAP Web IDE for SAP HANA, right-click the root folder of the XS advanced MTA project and choose

► [New > Java Module](#). While creating the new Java module, make sure you enable the following checkbox options:

- *Enable HANA DB support*
- *Enable OData support*

6. Write extensions within the Java module by using the API provided (required only for modify operations).

You can use the extensions framework to enhance functionality of your OData service to support data modification operations, for example, CREATE, UPDATE, and DELETE. For more information on the Java-based OData extension framework, see *Related Links* below.

 **Note**

At this point you have a working MTA that exposes an OData v4 service from the annotated CDS contexts defined in the database module. A productive application may require an application router, which routes requests to the Java module, and a security configuration, which performs authentication and authorization checks during execution of a query. For configuring security and application router, follow the below steps.

The following steps show you how to create an application router in the HTML 5 module and configure the `xs-security.json` file. For more information about configuring application security, see *Related Links* below.

7. Add an HTML5 module to your MTA project.

In SAP Web IDE for SAP HANA, right-click the root folder of the XS advanced MTA project and choose

► [New > Basic HTML5 Module](#).

8. Configure the `mta.yaml` file to expose your Java module.

To expose your OData service, perform the steps indicated in the following procedure:

- a. Configure the `mta.yaml` development descriptor file to expose the URL for the Java module.

Initially, the relevant section of the `mta.yaml` file would look similar to the code sample below; you need to verify that the Java module **provides** its URL for consumption by other modules. With this change, the configuration would look like the following example:

 **Sample Code**

```
- name: MyJavaModule
  type: java
  path: MyJavaModule
  requires:
    - name: hdi-container
      properties:
        JBP_CONFIG_RESOURCE_CONFIGURATION: [tomcat/webapps/ROOT/META-INF/
context.xml:{service_name_for_defaultDB" : "~{hdi-container-name}" }]
  provides:
    - name: MyJavaModule_api
```

```
properties:  
  service_url: ${default-url}
```

- b. Configure the development descriptor (`mta.yaml`) such that the HTML5 module **requires** the above URL exposed from the Java module. Change the configuration of HTML5 module to match the following example:

#### Sample Code

```
- name: MyHTML5Module  
  type: html5  
  path: MyHTML5Module  
  requires:  
    - name: MyJavaModule_api  
      group: destinations  
      properties:  
        name: MyJavaModule_be  
        url: ~{service_url}
```

- c. Configure the application descriptor (`xs-app.json`) located in the `MyHTML5Module` directory to add a route to the Java module under the source `"/java/odata/v4"`. You also need to specify the authentication type as `xsuaa`, as illustrated in the following example:

#### Sample Code

```
{  
  "welcomeFile: "index.html",  
  "authenticationMethod: "route",  
  "routes": [  
    {  
      "source": "/java/odata/v4",  
      "authenticationType": "xsuaa",  
      "destination": "MyJavaModule_be"  
    }]  
}
```

#### Note

The `"destination":` value `"MyJavaModule_be"` must match the value specified in the application's development descriptor (`mta.yaml`)

9. Create a security descriptor (`xs-security.json`) file at the project level.

For more information about creating the security descriptor (`xs-security.json`), see *Related Links* below. If you do not need to create any more scopes than those that are required by default (for example, due to security constraints in the `web.xml`), you can use the following example as a template.

#### Tip

The example below defines a single authorization scope (`ODATASERVICEUSER`) and a role template (`GenericODataAccessRole`) that has this scope.

#### Sample Code

```
{
```

```

"xsappname": "YourAppName",
"scopes": [
    "name": "$XSAPPNAME.ODATASERVICEUSER",
    "description": "Enter"
],
"role-templates": [
    "name": "GenericODataAccessRole",
    "scope-references": [
        "$XSAPPNAME.ODATASERVICEUSER"
    ]
}

```

- Add a dependency to the User Account and Authorization (UAA) service in the HTML5 (Web) module and the JAVA module of the development descriptor (`mta.yaml`), as illustrated in the following example:

#### Sample Code

```

- name: MyJavaModule
  type: java
  path: MyJavaModule
  requires:
    - name: MY_uaa
    - name: hdi-container
      properties:
        JBP_CONFIG_RESOURCE_CONFIGURATION:
          '[tomcat/webapps/ROOT/META-INF/context.xml:
            {"service_name_for_DefaultDB" : "~{hdi-container-name}"]'
  provides:
    - name: MyJavaModule_api
      properties:
        service_url: ${default-url}
- name: MyHtml5Module
  type: html5
  path: MyHtml5Module
  requires:
    - name: MY_uaa
    - name: MyJavaModule_api
      group: destinations
      properties:
        name: MyJavaModule_be
        url: ~{service_url}
        forwardAuthToken:true

```

Remember to add `MY_uaa` to the resource segment, too, as illustrated in the following example:

#### Sample Code

```

resources:
  - name: hdi-container
    properties:
      hdi-container-name: ${service-name}
      type: com.sap.xs.hdi-container
  - name: MY_uaa
    type: com.sap.xs.uaa
    parameters:
      path: ./xs-security.json

```

- Create a new User Account and Authentication (UAA) service instance using the XS command-line interface.

- a. Log in to your XS advanced system from the command line interface using the `xs login` command.
- b. Download the security descriptor you created (`xs-security.json`) to your local file system.
- c. In the command shell, change directory to the directory where the `xs-security.json` file is located.
- d. Run the following command:

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

11. Build the MTA Java application.

You can now deploy your application and test the OData service. However, you need to ensure that users (for example, developers) have the authorization scopes required to access the application. To set up the required authorization, perform the following steps:

- a. Build the Multi-Target Application (MTA).
- b. Request the administrator to create a role for the `GenericODataAccessRole`. This is the role for which a role-template was defined in the `xs-security.json` security descriptor.

If you have the required administration authorization, you can create a new role, which must then be added to the developer user's role collection. For more information about roles and role collections, see *Related Links*.

12. Run the Java module and the HTML5 module.

This generates a `<Base URL>` for your application, which you can find at the top right corner of the console in SAP Web IDE for SAP HANA.

Check the availability of the OData service you created; you can access the OData service using the URLs listed in the following table:

Table 59: Sample OData service URLs

Scenario	Sample URL
Single context with namespace:	<code>&lt;Base URL for the HTML5 module&gt;/java/odata/v4/&lt;namespace&gt;._&lt;context&gt;</code>
Single context without namespace:	<code>&lt;Base URL for the HTML5 module&gt;/java/odata/v4/&lt;context&gt;</code>
Child context with namespace:	<code>&lt;Base URL for the HTML5 module&gt;/java/odata/v4/&lt;namespace&gt;._&lt;OuterContext&gt;.&lt;InnerContext&gt;</code>
Child context without namespace:	<code>&lt;Base URL for the HTML5 module&gt;/java/odata/v4/&lt;OuterContext&gt;.&lt;InnerContext&gt;</code>

You can clean the cache of your application using the `clearCache` tool, for example, with either of the following URLs:

- With namespace: `<Base URL for the HTML5 module>/java/odata/clearCache/<namespace>._<context>`
- Without namespace: `<Base URL for the HTML5 module>/java/odata/clearCache/<context>`

### Note

Cache cleaning is required only when the database module of an XS advanced application is updated and deployed individually; that is, without building and deploying the complete MTA to which the database module belongs. If you are deploying a complete, updated MTA, cache cleaning is not required. To perform a clean-cache operation, you need the authorization scope **ODATASERVICEADMIN**.

## Related Information

[SAP HANA Administration Guide \(Building Roles for XS Advanced Applications\)](#)

[Create the Security Descriptor for Your XS Advanced Application \[page 763\]](#)

[Setting Up Application Projects \[page 989\]](#)

[CDS Contexts in XS Advanced \[page 234\]](#)

[Configure the XS Advanced Application Router \[page 717\]](#)

[Using the OData Extension Framework \[page 482\]](#)

[Publishing CDS Contexts as OData Services \[page 469\]](#)

[Setting Up Security Artifacts \[page 757\]](#)

### 6.1.2.1.1 Publishing CDS Contexts as OData Services

A CDS context defined in a CDS document can be published as an OData service.

To publish a CDS context as an OData V4 service by annotating the CDS context in the CDS document with the annotation `@OData.publish = true`.

### Restriction

The use of the `@OData.publish` annotation in a CDS document has the following restrictions.

- It is only possible to publish a CDS context as an OData v4 service using the `@OData.publish` annotation. If any other CDS artifact is extended with the `@OData.publish` annotation, for example, an entity or a view, the annotation is ignored and the CDS artifact is not considered for creating an OData service.
- An annotated context that includes a nested context cannot be published as an OData service.

### Tip

You can annotate a nested context for exposure as an OData service. However, the nested context must not, itself, contain any nested (child) contexts.

- If the annotated context includes a navigation to an external entity or view (outside the annotated context), no OData service can be created.

To navigate from one context to another, you must create a view of the target entity and create a navigation to that view in the annotated context. However, it is allowed to include a reference to a complex

type, inferred type, and scalar type from the annotated context to another context. In that scenario, every OData service creates the type inside its service and creates an internal reference.

- Direct access to the artifacts defined in an annotated nested context is possible, for example, using the following syntax in the query URL:
  - With namespace: <Base URL>/java/odata/v4/<namespace>.\_.<OuterContext>.<InnerContext>
  - Without namespace: <Base URL>/java/odata/v4/<OuterContext>.<InnerContext>

## Contexts as OData v4 Services

All top-level contexts defined in a CDS document (or belonging to a specified name space) and appearing under the annotation `@OData.publish` are published as OData v4 services. There is no restriction on the number of CDS contexts that can be annotated as `OData.publish : true`.

### Sample Code

Publish a CDS Context (and Contents) as an OData v4 Service

```
namespace acme.com;
@OData.publish : true
context ContextA {
    MyEntity1 {
        key ID : Integer;
    };
    MyEntity2 {
        key ID : Integer;
        type Address1
    };
};
```

### Note

An annotated context that includes a nested (sub) context cannot be published as an OData service. However, you **can** annotate the nested context itself for exposure as an OData service. However, the nested context must not contain any nested contexts.

## Subcontexts as OData v4 Services

In the following example, the CDS artifacts defined in the subcontexts `SubContextA1` and `SubContextA2` are exposed as OData v4 services.

### Sample Code

Publish CDS Subcontexts as OData v4 Services

```
namespace acme.com;
context ContextA {
    @OData.publish : true
```

```
context ContextA1 {
    MyEntity1 {
        key ID : Integer;
    };
};

@OData.publish : true
context ContextA2 {
    MyEntity2 {
        key ID : Integer;
        type Address1
    };
};
};
```

## Related Information

[CDS Contexts in XS Advanced \[page 234\]](#)

[Tutorial: Create OData v4 Services for XS Advanced Java Applications \[page 464\]](#)

### 6.1.2.1.2 Supported CDS-OData v4 Features in the XS Advanced Java Stack

This document lists the OData v4 and CDS features supported by the XS advanced Java framework.

The information in this topic covers the following areas:

- [Supported OData v4 Features \[page 472\]](#)
- [Supported CDS Features \[page 476\]](#)

#### Code Syntax

Example CDS Document (`Sample.hdbcards`) with OData Service Enabled

```
@OData.publish : true
context Sample {
    type custid : Integer;
    entity Customer{
        key CustomerID : custid;
        key Type : String(20);
        CustomerName : String(100);
        CustAddress : Address;
        SalesOrders : association[0..*] to SalesOrderHeader {CustomerID,Type};
    };

    type Address{
        Street : String(100);
        Area : String(100);
        City : String(30);
        State : String(30);
        Country : String(100);
    };

    entity SalesOrderHeader{
        key SalesOrderId : String(36);
```

```

Note : hana.VARCHAR(300);
CurrencyCode : String(5);
NetAmount : Decimal(10,2);
TaxAmount : Integer;
OrderDate : LocalDate;
CallTime : LocalTime;
DeliveryETA : UTCDateTime;
CashOnDelivery : Boolean;
CustomerID : customerid;
Type : CustomerType;
ToCustomer : association[1] to Customer {CustomerID,Type};
LineItems : association[0..*] to SalesOrderLineItem {SalesOrderID};
};

entity SalesOrderLineItem{
    key SalesOrderID : String(36);
    key Position : Integer;
    ProductId : Integer;
    Note : String(255);
    Quantity : Integer;
    SalesOrder : association to SalesOrderHeader;
};

}
;

```

## Supported OData Features

The following table lists the OData version 4.0 features supported by the XS advanced Java framework:

Table 60: Supported OData Features

Feature	Supported Scenario	Sample Query
Read	Read an entity set	/Customer
	Read a particular entity	/Customer(CustomerID=1,Type='Enterprise')
	Filter operation on entity set	/Customer?\$filter=(Name eq 'Rowan Fairbrother')
Navigation	One to many	/Customer(CustomerID=1,Type='Enterprise')/SalesOrders
	One to one	/SalesOrderHeader(1)/ToCustomer
	Navigation with filter	/Customer(CustomerID=1,Type='Enterprise')/SalesOrders?\$filter=NetAmount gt 5
	Cross service referencing	

Feature	Supported Scenario	Sample Query
\$expand	Expand a subset	/Customer?\$expand=SaleOrders
	Multi-level expand	/Customer?\$expand=(SalesOrders?\$expand=(LineItems))
	Expand more than one subset	/SalesOrderHeader?\$expand=LineItems,ToCustomer
	Multi-level expand with filter operations	/Customer?\$expand=(SalesOrders?\$expand=(LineItems?\$filter=Position eq 1))
	Query operation with multi-level expand	/SalesOrderHeader?\$expand=LineItems(\$select=quantity) & \$select=deliveryeta
\$top & \$skip	Request a particular page of items (\$top) by excluding (\$skip) selected items.  Number of results can be restricted under expanded entities.	/Customer?\$top=10&\$skip=5  /Customer?\$expand=SalesOrders(\$top=2,\$orderby=NetAmount desc) Expands SalesOrders for a customer only with the top 2 highest netamount
Pagination	You must treat the URL of the <a href="#">Next</a> link as “opaque”.  You cannot append system query options to the URL of a <a href="#">Next</a> link.  OData services may use the reserved system query option \$skiptoken when building next links.	N/A
\$batch	Read and Query options	
	Change sets	
\$apply	Aggregate	/SalesOrderHeader?\$apply=aggregate(NetAmount with sum as total)
	GroupBy	/SalesOrderHeader?\$apply=groupby(CustomerID)

Feature	Supported Scenario	Sample Query
	Filter	<pre>SalesOrderHeader? \$apply=filter(NetAmount add TaxAmount lt 10)</pre>
	Compute	<pre>/SalesOrderHeader? \$apply=compute(NetAmount add TaxAmount as TaxInclusive)</pre>
\$count	As a resource path segment	<pre>/SalesOrderHeaders/\$count</pre> <p><b>⚠ Restriction</b> Not supported with navigation</p>
	As a system query option	<pre>/SalesOrderHeaders?\$count=true</pre> <p><b>⚠ Restriction</b> Works only at the parent level</p>

Feature	Supported Scenario	Sample Query
\$filter	<p>Lambda operators:</p> <div style="background-color: #ffffcc; padding: 10px;"> <b>⚠ Restriction</b> <ul style="list-style-type: none"> <li>• Works only at the parent level</li> <li>• Only comparison operators are supported</li> </ul> </div> <p>Comparison operators:</p> <div style="background-color: #ffffcc; padding: 10px;"> <b>⚠ Restriction</b> <p>The 'has not' operator is not supported.</p> </div> <p>Logical operators:</p> <div style="background-color: #ffffcc; padding: 10px;"> <b>⚠ Restriction</b> <p>The 'not' operator is not supported.</p> </div> <p>Query functions:</p> <div style="background-color: #ffffcc; padding: 10px;"> <b>⚠ Restriction</b> <p>Only string functions supported. In string functions, 'indexof' and 'substring' are not supported.</p> </div>	/SalesOrderHeaders?\$filter=LineItems/any(d:d/Quantity gt 200)

## OData v4 Analytics

In OData version 4, the `Aggregate` option enables data aggregation in OData without compromising general OData principles; it avoids the creation of any superfluous Edm data types that are required for showing aggregated data. Without aggregation, you must create Edm types, each of which has the structure of the desired view. This is not a development model that is scalable or maintainable, and it would require a lot of effort to set up and implement.

You can combine options with the `$apply` query, as illustrated in the following examples:

- `compute/aggregate`  

```
SalesOrderHeader?$apply=compute(NetAmount add TaxAmount as TaxInclusive) / aggregate(TaxInclusive with sum as Totality)
```
- `filter/aggregate`  

```
SalesOrderHeader?$apply=filter(NetAmount add TaxAmount gt 10) / aggregate(TaxAmount with sum as Tottax)
```

- filter/aggregate/compute  

```
SalesOrderHeader?$apply=filter(NetAmount add TaxAmount gt 10) /  
aggregate(TaxAmount with sum as Tottax) /compute(Tottax mul 3 as TriTottax)
```
- filter/groupby(aggregate)  

```
SalesOrderHeader?$apply=filter(NetAmount add TaxAmount gt 10) /  
groupby((CustomerID), aggregate(TaxAmount with average as AvgTx))
```
- groupby(aggregate)/filter  

```
SalesOrderHeader?$apply=groupby((CustomerID), aggregate(NetAmount with sum as  
total) /filter(total gt 20000))
```
- filter/compute  

```
SalesOrderHeader?$apply=filter(NetAmount add TaxAmount gt 10) /compute(TaxAmount  
mul 2 as DT) &$expand=LineItems
```

## Supported CDS Features

The following table lists the CDS features supported by the XS advanced Java framework in combination with OData v4 :

Table 61: Supported CDS Features

Feature	Comments
Entities	Entities without a key are not supported
Views	Views without keys are not supported
Data types:	<ul style="list-style-type: none"> <li>• Primitive</li> <li>• Scalar</li> <li>• Inferred</li> <li>• Complex</li> <li>• Complex types within complex types</li> <li>• Inferred types within complex types</li> </ul>

Feature	Comments
UI annotations	<ul style="list-style-type: none"> <li>• Hidden</li> <li>• Identification</li> <li>• LineItem</li> <li>• Mandatory</li> <li>• Masked</li> <li>• MultiLineText</li> <li>• Optional</li> <li>• ReadOnly</li> <li>• StatusInfo</li> <li>• Chart</li> <li>• Contact</li> <li>• FieldGroup</li> <li>• DataPoint</li> <li>• SelectionFields</li> <li>• HeaderInfo</li> </ul>
Additional metadata annotations	<ul style="list-style-type: none"> <li>• Consumption.semanticObject</li> <li>• Consumption.filter</li> <li>• ObjectModel.semanticKey</li> <li>• ObjectModel.virtualElement</li> <li>• ObjectModel.createEnabled</li> <li>• ObjectModel.updateEnabled</li> <li>• ObjectModel.deleteEnabled</li> <li>• Semantics.quantity.unitOfMeasure</li> <li>• Semantics.amount.currencyCode</li> <li>• Semantics.address</li> </ul>

### i Note

To use UI annotations, you need to add `using sap.common::UI;` in your CDS files. In addition, it is necessary to add a `package.json` to the XS advanced application's HDB module or modify the existing one. The `package.json` file in the HDB module must specify the additional dependency to the “`@sap/ui-annotations`”, as illustrated in the following example:

### Sample Code

#### Using UI Annotations

```
{
  "name": "deploy",
  "dependencies": {
    "@sap/hdi-deploy": "3.1.2",
    "@sap/ui-annotations": "2.0.3"
  }
}
```

```

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

```

## Related Information

[Defining OData v4 Services for XS Advanced Java Applications \[page 463\]](#)

[Tutorial: Create OData v4 Services for XS Advanced Java Applications \[page 464\]](#)

[CDS to OData v4 Datatype Mapping \[page 478\]](#)

[Streaming Data for OData Queries \[page 489\]](#)

[Enable Server Driven Pagination \[page 491\]](#)

### 6.1.2.1.2.1 CDS to OData v4 Datatype Mapping

This document lists how the XS advanced Java framework interprets the CDS artifacts defined in CDS documents (.hdbcds) and converts them to OData metadata.

The information in this topic covers the following areas:

- [CDS-OData v4 Datatype Mapping \[page 478\]](#)
- [CDS Artifact to OData v4 Datatype Mapping \[page 479\]](#)

### CDS Data Types as OData v4 Data Types

The following table illustrates how CDS data types are mapped to OData V 4.0 data types:

Table 62: Datatypes Mapping

CDS Datatype	OData Edm Datatype
String (n)	Edm.String
LargeString	Edm.Stream
Binary (n)	Edm.Binary
LargeBinary	Edm.Stream
Boolean	Edm.Boolean
Integer	Edm.Int64

CDS Datatype	OData Edm Datatype
Integer64	Edm.Int64
Decimal(p,s)	Edm.Decimal
DecimalFloat	Edm.Decimal
BinaryFloat	Edm.Double
LocalDate	Edm.Date
LocalTime	Edm.TimeOfDay
UTCDateTime	Edm.DateTimeOffset
UTCTimeStamp	Edm.DateTimeOffset

## CDS Artifact as OData v4 Datatypes

The following table illustrates how CDS artifacts, for example, entities or views, are mapped to OData V 4.0 data types:

Table 63: Artifacts Mapping

CDS Artifacts	OData Type
Context	Schema
Entity (with keys)	Entity type
Entity (without keys)	<b>Not exposed</b> ; can be exposed as functions
Views (with keys)	Entity type
Views (without keys)	<b>Not exposed</b> ; can be exposed as functions
Column properties	Edm properties
Complex type	Complex type
Scalar type	Type definition
Association (0...1)	Navigation type=<entity type> Nullable="true"
Association (1...1)	Navigation type=<entity type> Nullable="false"
Association (m...*)	Navigation type=<collection>
Association to complex type	

CDS Artifacts	OData Type
Calculated fields	
Enumeration	
Constant	
Default values(Not supported by CDS)	

## Related Information

[Tutorial: Create OData v4 Services for XS Advanced Java Applications \[page 464\]](#)

[Defining OData v4 Services for XS Advanced Java Applications \[page 463\]](#)

### 6.1.2.1.3 Using Cross Service Referencing

This document explains you how to use cross service navigation and referencing.

Cross service navigation enables inter communication of services. With this, navigation properties of entities of one service can reach entities of another service in a service group.

#### Sample Code

**Context: product\_sales:**

```
using product_category;
@OData.publish : true
context product_sales
{
  entity products
  {
    key id : Integer;
    name : String(50);
    category : Association to product_category.category;
  };
}
```

**Context: product\_category**

```
using product_details;
@OData.publish : true
context product_category
{
  entity category
  {
    key id : Integer;
    name : String(50);
    description : product_details.description;
  };
}
```

### Context: product\_details

```
@OData.publish : true
context product_details
{
  type description
  {
    short_description : String(40);
    long_description : String(100);
  };
}
```

**Navigation:** product\_sales → product\_category

**Referencing:** product\_category → product\_details

Now, when each of the mentioned contexts is published as OData service, their metadata would look like:

### Sample Code

#### Context: product\_sales

```
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx" Version="4.0">
  <edmx:Reference Uri="../../product_category/$metadata">
    <edmx:Include Namespace="product_category"/>
  </edmx:Reference>
  <edmx:DataServices>
    <Schema xmlns="http://docs.oasis-open.org/odata/ns/edm"
      Namespace="product_sales">
      <EntityType Name="products">
        <Key>
          <PropertyRef Name="id"/>
        </Key>
        <Property Name="id" Type="Edm.Int32" Nullable="false"/>
        <Property Name="name" Type="Edm.String" MaxLength="50"/>
        <NavigationProperty Name="category" Type="product_category.category"/>
      </EntityType>
      <EntityContainer Name="B0235B494D214C34B40169193F1F00A5">
        <EntitySet Name="products" EntityType="product_sales.products">
          <NavigationPropertyBinding Path="category"
            Target="product_category.B0235B494D214C34B40169193F1F00A5/category"/>
        </EntitySet>
      </EntityContainer>
    </Schema>
  </edmx:DataServices>
</edmx:Edmx>
```

#### Context: product\_category

### Sample Code

```
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx"
  Version="4.0">
  <edmx:Reference Uri="../../product_details/$metadata">
    <edmx:Include Namespace="product_details"/>
  </edmx:Reference>
  <edmx:DataServices>
    <Schema xmlns="http://docs.oasis-open.org/odata/ns/edm"
      Namespace="product_category">
      <EntityType Name="category">
        <Key>
          <PropertyRef Name="id"/>
        </Key>
```

```

<Property Name="id" Type="Edm.Int32" Nullable="false"/>
<Property Name="name" Type="Edm.String" MaxLength="50"/>
<Property Name="description" Type="product_details.description"/>
</EntityType>
<EntityContainer Name="B0235B494D214C34B40169193F1F00A5">
  <EntitySet Name="category" EntityType="product_category.category"/>
</EntityContainer>
</Schema>
</edmx:DataServices>
</edmx:Edmx>

```

Context: **product\_details**

```

<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx" Version="4.0">
  <edmx:DataServices>
    <Schema xmlns="http://docs.oasis-open.org/odata/ns/edm"
    Namespace="product_details">
      <ComplexType Name="description">
        <Property Name="short_description" Type="Edm.String" MaxLength="40"/>
        <Property Name="long_description" Type="Edm.String" MaxLength="100"/>
      </ComplexType>
      <EntityContainer Name="B0235B494D214C34B40169193F1F00A5"/>
    </Schema>
  </edmx:DataServices>
</edmx:Edmx>

```

#### Operation:

- Here, product\_sales has a navigation to an entity, "category" that is in another context, product\_category. The category entity in product\_category refers to a type description from another context product\_sales.
- The Namespace of the reference is the namespace of the schema it is referring to. Suppose the schema at URL: [https://host:port/java/odata/v4/product\\_details/\\$metadata](https://host:port/java/odata/v4/product_details/$metadata) refers to a namespace, product\_category, then the URL of the referred service would be: [https://host:port/java/odata/v4/product\\_category/\\$metadata](https://host:port/java/odata/v4/product_category/$metadata).

#### **i** Note

- If a navigated context is not published as an OData service, then the navigation fails and throws an appropriate error.
- If a service is referencing to a type in a context which is not published, then the type is copied to the metadata that uses the type.

### 6.1.2.1.4 Using the OData Extension Framework

Extend the OData v4 service to enable modification actions.

The OData services you create are by default Read-only services. To support write operations such as CREATE, UPDATE, DELETE, you have to write extensions, which are custom Java code containing annotated methods that perform the required operations. There is no restriction in terms of the package where the extension classes must be located. However, typically you create the class in the default package created within the XS advanced application's Java module. An `extensionTemplate` class is created by default when you create a Java application module with OData support enabled.

The OData v4 extension framework provides access to a set of objects which enables you to obtain the values required to implement the extension, as illustrated in the following table:

Table 64: OData v4 Extension Objects and Access

Action	Request Payload	Request Header	Key Info
CREATE	✓	✓	-
UPDATE	✓	✓	✓
DELETE	-	✓	✓

The object `DataSourceParams` provides any parameters required that are specific to the named data source. In the context of CDS, this object enables access to the connection object for the schema in which the current application is deployed.

➔ Tip

If you have to write extensions where multiple exposed OData services have an entity with the same name, then that extension is called for the corresponding entity in both OData services. To resolve this ambiguity and maintain individual extensions for each of these entities, use the parameter `serviceName` with a value that is the name of the OData service to which the entity belongs.

## Using the OData Extension Framework

You can use the extension framework to write code for OData methods, which are currently not supported by the framework, for example: `CREATE`, `UPDATE`, and `DELETE`. You can also use the extension for `READ`, for example, if it is necessary to perform the following actions:

- Completely override the framework handling for `READ`
- Perform some pre- or post-processing in addition to the framework's default implementation.

i Note

The extension framework is **not** supported for the `QUERY` method.

For each (Entity Set, OData Method) combination for which you want to extend the default functionality, it is necessary to write a method. For example, if you want to handle `CREATE` or `UPDATE` for the entity set "Customers" and `DELETE` functionality for the entity set "SalesOrders", you must create three additional methods. These methods must be annotated with the annotation `ExtendDataProvider`, for which the following parameters are available:

- `entitySet`  
The value for the entity set for which the extension is being written
- `requestTypes`  
Typically it has a single method as the value. However, if you want to handle multiple OData methods within a single extension method, you can provide additional values
- `serviceName` (optional)

In case you have to write extensions when multiple exposed OData services have an entity with the same name, then that extension is called for the corresponding entity in both OData services. To resolve this ambiguity and maintain individual extensions for each of these entities, you need to use the parameter **serviceName**. This should have value as the name of the OData service to which the entity belongs to.

The method signatures would look like the following example:

#### Sample Code

```
@ExtendDataProvider(entitySet = { "Customers" }, requestTypes =
RequestType.CREATE)
public void createCustomer(ExtensionContext ecx)
@ExtendDataProvider(entitySet = { "Customers" }, requestTypes =
RequestType.UPDATE)
public void updateCustomer(ExtensionContext ecx)
@ExtendDataProvider(entitySet = { "SalesOrders" }, requestTypes =
RequestType.DELETE)
public void createSalesOrders(ExtensionContext ecx)
```

- The method should be annotated properly with `ExtendDataProvider` annotation.
- The method should return `void`.
- The method should take a parameter of type `ExtensionContext`.

#### Note

Currently, update through patch is not supported.

## Related Information

[Tutorial: Create OData v4 Services for XS Advanced Java Applications \[page 464\]](#)

[The ExtensionContext APIs \[page 484\]](#)

[Methods Under DataProviderExtensionContext Class \[page 486\]](#)

### 6.1.2.1.4.1 The ExtensionContext APIs

The `ExtensionContext` object is used to retrieve the information required to write extensions for a particular OData method.

With the `CREATE` method, you must gain access to the request payload. For the `READ` method, you need a way to invoke the default framework implementation. For all the OData methods, there is only a single `ExtensionContext` object. For this reason, all methods are made available in extensions written for any OData method.

#### Caution

It is essential to use the correct methods for the particular scenario for which the extension is being written.

The following table lists the methods available, and where possible, provides some code examples:

Table 65: ExtensionContext API Methods

API Method	Description	Example
getDSParams ()	Used to get the data-source-specific parameters which might be useful for writing the extensions. For CDS, there is only the connection object which is accessed through this object.	Connection conn = ((CDSDSParams) ecx.getDSParams()).getConnection();
getODataRequest ()	Used to get the current OData request object. This object can be used for getting the request headers. The following example shows how to check the Prefer header for return preference:	Preferences.Return returnPreference = OData.newInstance().createPreferences(ecx.getODataRequest().getHeaders(Header.PREFER)).getReturn();
performDefault ()	This method can be used to invoke default implementation of the OData method for which the current extension is being written. This only makes sense with the READ method as the framework does not provide default implementation for CREATE,UPDATE, and DELETE methods.	N/A
getUriInfo ()	Returns the UriInfo object that contains information about the URI, Query Options, etc. This is mandatory with DELETE and UPDATE where the key predicate is required.	UriInfo uriInfo = ecx.getUriInfo();  UriResourceEntitySet entity = (UriResourceEntitySet)uriInfo.getUriResourceParts().get(0); List<UriParameter> kp = entity.getKeyPredicates();
asDataProviderContext ()	Called to “specialize” the ExtensionContext so that additional methods are available for other use-cases. When this method is called, the result is an object of DataProviderExtensionContext which is a subclass of ExtensionContext.	N/A

## Related Information

[Tutorial: Create OData v4 Services for XS Advanced Java Applications \[page 464\]](#)

## 6.1.2.1.4.2 Methods Under DataProviderExtensionContext Class

A list of the methods available for the DataProviderExtensionContext class.

All the methods under `ExtensionContext` are also available under `DataProviderExtensionContext`. The following table lists and describes the additional methods that are available under `DataProviderExtensionContext`:

**i Note**

HANA DB returns Boolean values as 1 or 0. These values should be transformed to TRUE or FALSE while creating the response entity

Table 66: DataProviderExtensionContext Class Methods

Method	Description	Examples
<code>getDeserializerResult()</code>	Used to retrieve the request payload. It returns an object of <code>DeserializerResult</code> on which you have to call <code>getEntity</code> method which will give the entity object corresponding to the request payload.	<pre>DataProviderExtensionContext dpCtx =     ecx.asDataProviderContext(); DeserializerResult payload =     dpCtx.getDeserializerResult(); Entity customerEntity =     payload.getEntity();</pre>
<code>setResultEntity()</code>	Used to set the response entity which can be the result entity for CREATE, UPDATE or READ. This method should <b>not</b> be called if <code>setEntityToBeRead()</code> method is used to perform a READ operation.	N/A
<code>setEntityToBeRead()</code>	Used for CREATE and UPDATE requests, if you want to extend the framework to handle the READ part of the operation.	For the OData CREATE (and UPDATE) methods, the entity has to be created in the back end and returned as the response. However, the response part can be handled by the framework since the READ functionality is already supported. As a result, after finishing the creation step, rather than reading and forming the entity object in the extension itself, you can call this API, to make the framework handle the rest.

Method	Description	Examples
<code>setEntityToBeRe ad(Map&lt;String, String&gt; keys)</code>	<p>Similar to the <code>setEntityToBeRead()</code> except that you can specify the key values of the entity to be read. This is useful in a case of an auto-generated key from the database in cases where the framework has no way of knowing which entity to read from the back-end.</p> <div style="background-color: #f2e0c7; padding: 5px;"> <p><b>i Note</b> Works for a deep insert, too.</p> </div>	N/A

## Related Information

[Tutorial: Create OData v4 Services for XS Advanced Java Applications \[page 464\]](#)

[Using the OData Extension Framework \[page 482\]](#)

[The ExtensionContext APIs \[page 484\]](#)

### 6.1.2.1.4.3 Using Extension Framework for Actions and Function Imports

This document details you about extending the metadata to add actions and functions to your OData v4 application.

Operations allow execution of custom logic on parts of a data model. Functions are operations that do not have side-effects and can support further composition e.g. additional filter operations, functions or an action.

Actions are operations that allow side effects, such as data modification, and cannot be further composed in order to avoid non-deterministic behavior. Actions and functions can be categorized as:

1. Bound (called as members of an instance of that type)
2. Unbound (called as static operations)

Action imports and function imports enable unbound actions and functions to be called from the service root. To develop the required extension, follow the below steps

1. **Creating the metadata**

To add functions and actions in your application, you need to write a method that takes no parameters and returns `List<CSDLOperation>`. Then you have to annotate the method with the annotation `@ExtendMetadata` and set the service name.

➔ Tip

- Create all your actions and functions, and return the list.
- The developer can annotate as many methods with the annotation.
- If service name is not mentioned, then it is added to all the published OData service in the application.

↳ Sample Code

```
Sample Code:  
@ExtendMetadata(serviceName="SalesOrder")  
public List<CSDLOperation> myOperations()  
{  
}  
@ExtendMetadata  
public List<CSDLOperation> myOperations()  
{  
}
```

## 2. Processing the data request for the operation

Now, you need to write the code to be executed when the corresponding action or function import is called. You can reuse the already available DataProvider extension and add a parameter called `operationName`.

The guidelines for writing the code are same as writing extension for CRUD operations. The method name can be anything, but the parameter should be `ExtensionContext`.

➔ Tip

- If no `serviceName` is mentioned, it is applicable on all services in the application.
- The logic remains the same as usual Extension for CUD operations.
- You should take care to set the proper return type as mentioned in the Action or Function return type. To set the return object, you can use either of the following two:
  - `DataProviderExtensionContext dpCtx = ectx.asDataProviderContext();`
  - `dpCtx.setResultEntityCollection(<Olingo EntityCollection type>); or`  
`dpCtx.setResultEntity(<Olingo Entity type>); or dpCtx.setResult(<Object>);`

↳ Sample Code

```
@ExtendDataProvider(serviceName = "SalesOrder", operationName =  
"countcategories")  
public void count_cat(ExtensionContext ectx)  
{  
}  
@ExtendDataProvider(operationName = "countcategories")  
public void count_cat(ExtensionContext ectx)  
{  
}
```

## Related Information

[Action Imports and Function Imports](#) ↗

### 6.1.2.1.4.4 Streaming Data for OData Queries

Streaming is an optional feature that you can use to receive data in chunks. Streaming is typically used to improve memory and time performance on the server side. There is very low memory footprint on the server side if a huge collection of data is being fetched. Clients can process the incoming streaming data. If the client doesn't support streaming, the transport layer takes care of accumulating the data and passing it on to the client.

#### Note

Streaming is enabled by default.

The following restrictions apply to streaming for OData queries:

- The method `isStreaming()` is mandatory for a `DataProvider` and by default is set to `false`. To enable streaming for the `DataProvider`, you must set `isStreaming()` to `true`.
- By default, `CDSDataProvider` always streams data. However, you can override the `DataProvider` and `isStreaming()` method.

You can switch streaming on or off according to your requirement. To turn off streaming, you can extend `CDSDataProvider` and add the `isStreaming()` method as illustrated in the following example:

#### Sample Code

```
@Override  
public Boolean isStreaming()  
{  
    return false;  
}
```

#### Restriction

Streaming is not supported for `$batch` and `$apply` calls.

## Related Information

[Defining OData v4 Services for XS Advanced Java Applications \[page 463\]](#)

[Tutorial: Create OData v4 Services for XS Advanced Java Applications \[page 464\]](#)

## 6.1.2.1.4.5 Use Batch Operations with Change Set

In XS advanced the Java framework for OData v4 provides support for processing change sets within a batch request. To process a change set, the framework reuses the functionality provided for CRUD operation through an extension mechanism. To control the transaction, the framework monitors operations on the connection object provided to the extension developer. This ensures a seamless integration of existing extension functionalities with the new changes in the framework.

### Context

The extension developer is expected to use traditional transaction logic while writing the extension, as follows:

### Procedure

1. Get the connection object from the extension context, as illustrated in the following example:

#### Sample Code

```
Connection conn = ((CDSParams) ecx.getDSParams()).getConnection()
```

2. Write the required transaction logic.
3. Close the connection, for example, using the close-connection command: `Conn.close()`

When a change set is requested, the framework internally ensures that the following calls on the connection object are ignored:

- `connection.setAutoCommit (Boolean)`
- `connection.rollback()`
- `connection.rollback(Savepoint)`
- `connection.commit()`
- `connection.close()`

### Related Information

[Batch Requests](#) ↗

## 6.1.2.1.5 Enable Server Driven Pagination

Set limits for the amount of data to be fetched and sent over the network by OData queries.

### Context

Server side pagination is used to limit the amount of data to be fetched and sent over the network to the client, in case generic query options yield huge amounts of data. You can request for responses that include only a partial set of the items identified by the request. For this, on the server side, you have to specify the number of entities to be returned in the response. The response contains a link, next link that allows you to retrieve the next partial set of items. You must treat the URL of the next link as opaque and must not append system query options to the URL of a next link.

To enable server-side pagination for your XS advanced Java application, perform the following steps:

### Procedure

1. Create a new DataProvider class which extends CDS DataProvider.

Once the project is created via of the archetype, add the Java class `MyDataProvider.java` to the Java module of your project.

2. Add your required page size in the `getPageSize()` method. As you are provided with `UriInfo` object, you can return different page size for different entity sets.

The following code snippet shows how to turn off streaming and set the server-side page size:

#### Sample Code

```
package groupID.MySalesOrder;
import org.apache.olingo.commons.api.edm.provider.CsdlEdmProvider;
import org.apache.olingo.server.api.uri.UriInfo;
import com.sap.gateway.v4.rt.cds.CDSDataProvider;
import com.sap.gateway.v4.rt.cds.ODataToCDSProcessor;
public class MyDataProvider extends CDSDataProvider {
    public MyDataProvider(CsdlEdmProvider edmProvider) {
        super(edmProvider);
    }
    public MyDataProvider(ODataToCDSProcessor jdbcProcessor, CsdlEdmProvider
        edmProvider) {
        super(jdbcProcessor, edmProvider);
    }
    /*To disable streaming, return false*/
    @Override
    public Boolean isStreaming()
    {
        return false;
    }
    /*To set the server side page size, return a long value. To disable server
     * side paging,
     * return getPageSize() as null*/
    @Override
    public Long getPageSize(UriInfo uriInfo) {
```

```
    return (long)1000;
}
}
```

3. Change your web.xml file to update the value of the context parameter DPC to your new class.

### Sample Code

```
<context-param>
<param-name>DPC</param-name>
<param-value>package.MyDataProvider</param-value>
</context-param>
```

The framework takes care to append the NextLink in case it is required.

## Next Steps

Regulations imposed when you request a particular page size through header `odata.maxpagesize()`.

- When page size is null and server side pagination is switched off, this request is ignored.
- When requested page size is lesser than or equal to that set by developer, then framework services the new page size requested by you.
- When requested page size is greater than that set by developer, then framework ignore the request and serves the one set by the developer.
- In all cases when server driven pagination is active, the page size requested used in the collection query is present in the response header Preference-Applied : `odata.maxpagesize`.

## Related Information

[Defining OData v4 Services for XS Advanced Java Applications \[page 463\]](#)

[Tutorial: Create OData v4 Services for XS Advanced Java Applications \[page 464\]](#)

[Server-Driven Paging !\[\]\(f10a7d24c1a1171e3bd262ab22e502b2\_img.jpg\)](#)

## 6.1.2.1.6 Customize Applications with the User's Locale

In XS advanced the Java framework enables you to personalize your application with the user's locale.

### Context

The locale can be retrieved from two headers in the following order:

1. Custom `x-sap-request-language` HTTP header
2. `Accept-Language` HTTP header

Check if the locale is set in the request headers. If it is, set it for the connection so that the database can perform locale specific operations. If there is no locale, do not set anything in the connection.

#### Note

This functionality is always on.

To use the locale personalization, follow the below procedure:

### Procedure

1. Create a new `MyDataProvider` class that extends `CDSDataProvider`.

The implementation for setting the locale is already included in `CDSDataProvider`. The connection object comes from the `CDSDataProvider` class, you need not do anything else.

2. Change the `web.xml` file to update the value of the context parameter (`context-param`) to `DPC` in your new class (new `DataProvider`).

#### Sample Code

```
<context-param>
    <param-name>DPC</param-name>
    <param-value>package.MyDataProvider</param-value>
</context-param>
```

## 6.1.2.1.7 Performance Statistics for OData Requests

Enable access to performance statistics for OData requests.

In order to facilitate the measurement of performance metrics for OData end points provisioned in XS advanced, the application provides support for getting the same via query parameters and headers without the need to set up performance tools explicitly. This feature also helps in identifying the bottleneck area during network communication, which you can later troubleshoot.

You can access SAP performance statistics from the framework for each OData request. To obtain the SAP performance statistics, you can add `?sap-statistics-xsa-java-odatav4=true` at the end of the request URL, regardless of whether it is an HTTP method of type GET, POST, PUT, MERGE, PATCH or DELETE. Alternatively, you can enter `name=sap-statistics-xsa-java-odatav4, value=true` in the HTTP request header. The setting in the URL is treated with higher priority than setting it in the HTTP header.

The XS Advanced OData Provisioning framework (XS advanced framework for provisioning of OData) provides performance statistics to consumers in the HTTP response header in the following format:

- HTTP Header Name: `sap-statistics-xsa-java-odatav4`
- HTTP Header Value: `true`

The following table lists and describes the HTTP header values:

Table 67: HTTP Header Values

HTTP Header Value	Description
total	The total processing time for executing an OData request starting from the time the JAVA application received the call to the time it sent back the first response
fw	The time taken by OData framework excluding the time taken to retrieve data from back-end; this includes time taken before sending request to back-end and after receiving the data from back-end. It also includes time taken for extension code to complete.
ext	The time taken by the custom code written for extensions to execute
db	The time taken to retrieve data from the back-end including the network round trip to the back-end.

**i Note**

- Due to SAP security policies, OData provisioning provides performance statistics only if the OData request is processed successfully.
- All times are calculated in milliseconds.

## Related Information

[Defining Web-based Data Access \[page 407\]](#)

[Defining OData v4 Services for XS Advanced Java Applications \[page 463\]](#)

## 6.2 Data Access with XMLA in SAP HANA XS

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

You can map the persistence and consumption models with XML for Analysis (XMLA). With XMLA, you write multi-dimensional-expressions (MDX) queries wrapped in an XMLA document. An XML for Analysis (XMLA) application running in SAP HANA application services (SAP HANA XS) is used to provide the consumption model for client applications exchanging MDX queries (wrapped in XMLA documents) with the SAP HANA database.

XMLA uses Web-based services to enable platform-independent access to XMLA-compliant data sources for Online Analytical Processing (OLAP). XMLA enables the exchange of analytical data between a client application and a multi-dimensional data provider working over the Web, using a Simple Object Access Protocol (SOAP)-based XML communication application-programming interface (API).

Applications running in SAP HANA XS enable you to control the flow of data between the presentational layer, for example, in the Web browser, and the data-processing layer in SAP HANA itself, where the calculations are performed, for example in SQL or SQLScript. If you develop and deploy an XMLA service running in SAP HANA XS, you can take advantage of the access to SAP HANA that SAP HANA XS provides to improve end-to-end performance.

### Note

The XS advanced application `xmla` is installed by default in the SAP space. However, if you are using multiple tenant databases, you must install the `xmla` application in another space (or spaces), for example, `DEV`. In addition, the target space must already be mapped to a tenant database **before** you deploy the `xmla` application. You can map an XS advanced space to a tenant database using the [SAP HANA Service Broker Configuration](#) tool that is included in the [XS Advanced Administration](#) tools.

## Related Information

[Defining Web-based Data Access \[page 407\]](#)

[Execute MDX Queries in the SAP HANA Database Explorer \[page 77\]](#)

## 6.2.1 Define the Data to Expose with an XMLA Service in XS Advanced

Define the tables and views to expose with the XMLA service in XS advanced.

### Prerequisites

- Data to expose  
If you already have a data model containing tables or views that can be exposed, you do not need to create additional elements. You can use the tables and views that are already available.
- Access to data  
Ensure that the suitable permissions are granted to the users and applications requiring access
- Access to the XMLA service in XS advanced  
Only XS advanced users have access to the XMLA service; you can clone existing SAP HANA users to XS advanced, for example, using the XS Advanced Administration Tools

### Context

An XMLA service exposes data stored in database tables for analysis and display by client applications. However, first of all, you need to ensure that the tables and views to expose as an XMLA service actually exist and are accessible.

To define the data to expose using an XMLA service, you must perform at least the following tasks:

### Procedure

1. Create a simple database schema.
2. Create a simple database table to expose with an XMLA service.
3. If required, create a simple database view to expose with an XMLA service.
4. Grant select privileges to the tables and views to be exposed with the XMLA service.

### Related Information

[Data Access with XMLA in SAP HANA XS \[page 495\]](#)

[Set up and Use the XMLA Interface in XS Advanced \[page 497\]](#)

## 6.2.2 Set up and Use the XMLA Interface in XS Advanced

Enable and test the XMLA interface in XS advanced.

### Prerequisites

Before you set up and use the XMLA interface for XS advanced, note the following prerequisites:

- If you are using multiple tenant databases, you must deploy the `xm1a` application in a space (or spaces) other than the default space, SAP, for example, DEV.
- The target space must already be mapped to a tenant database **before** you deploy the `xm1a` application.

#### Restriction

Administrator permissions are required to access the *SAP HANA Service Broker Configuration* tool you need to use to complete this mapping task.

### Context

An XMLA service for XS advanced is configured by deploying the multi-target application (MTA) `xm1a` to the XS advanced run-time environment. After successful deployment of `xm1a` to the XS advanced run-time environment, the XMLA interface is available at the following URL:

`https://<SERVERNAME>:<PORT>/sap/hana/xm1a/requestbasic`

Where:

- `<SERVERNAME>`  
Is the name of the host where your XS advanced instance is running
- `<PORT>`  
Is the number of the port the XMLA service is listening on, for example:  
`https://host1.acme.com:51028/sap/hana/xm1a/requestbasic`

To set up and use the XMLA interface in XS advanced, perform the following steps:

### Procedure

1. Check if the XMLA application is already running in the XS advanced run-time environment.

The XS advanced application `xm1a` is installed by default in the SAP space. However, if you are using multiple tenant databases, you must install the `xm1a` application in another space (or spaces), for example, DEV.

To display a list of running applications in XS advanced, use the `xs apps` command, as illustrated in the following example:

### Note

Check the state of the application is **STARTED** as displayed in the *requested state* column.

```
xs apps
Getting apps in org "acmeorg" / space "SAP" as dev1...
Found apps:
name           requested state  instances   memory   disk
urls
-----
auditlog-db    STOPPED        0/1        1.00 GB  <unlimited>
<none>
auditlog-server STARTED      1/1        512 MB   <unlimited>
https://host1.acme.com:51002
auditlog-broker STARTED      1/1        64.0 MB  <unlimited>
https://host1.acme.com:51003
[...]
xsa-admin-backend STARTED    1/1        1.00 GB  <unlimited>
https://host1.acme.com:51010
xsa-admin       STARTED      1/1        1.00 GB  <unlimited>
https://host1.acme.com:51009
xmla           STARTED      1/1        1.00 GB  <unlimited>
https://host1.acme.com:51028
[...]
```

### Tip

To change the target space, use the command `xs target -s <SpaceName>`, for example, `xs target -s DEV`.

2. If you are using multiple tenant databases, map the XS advanced run-time space where you want to deploy `xmla` to the tenant database to which you want the XMLA interface to connect.

You can deploy `xmla` to more than one space, and map each space to a different tenant database.

### Tip

You can map an XS advanced space to a tenant database using the *SAP HANA Service Broker Configuration* tool that is included in the *XS Advanced Administration* tools. For more information about the *XS Advanced Administration* tools, refer to the *Application Run-time Services* section (for XS advanced) in the *SAP HANA Administration Guide*.

3. Deploy the XMLA application.

If you are using multiple tenant databases, you must install the `xmla` application in a space (or spaces) other than the default `SAP`, for example, `DEV`.

### Tip

The target space must already be mapped to a tenant database **before** you deploy the `xmla` application.

To deploy an XS advanced application, use the `xs deploy` command, as illustrated in the following example:

```
xs deploy <XMLA_MTA_ARCHIVE> --no-namespaces-for-services
```

4. Display the URL where the XMLA application is running.

In the following example, the XMLA interface is available at the following URL: `https://host1.acme.com:51028`

```
xs app xmla
Showing status and information about "xmla"
  name:          xmla
  requested state:  STARTED
  instances:      1
  memory:         1.00 GB
  disk:           <unlimited>
  buildpack:      <default>
  urls:           https://host1.acme.com:51028
Instances of droplet 1 created at Sep 21, 2016 5:17:09 PM
index   created           state    host           internal
port   os user

-----
1       Sep 30, 2016 2:43:10 AM   RUNNING   host1.acme.com:51028
50032      acmexsa
```

5. Send an MDX query to the XMLA interface.

The MDX query should be sent in the form of a POST-request using the following URL:

```
https://<servername>:<PORT>/sap/hana/xmla/requestbasic
```

The POST-request you send should include the following HEADER elements:

- `SOAPAction = "urn:schemas-microsoft-com:xml-analysis:Execute"`
- `Content-Length = <length>`
- `Content-Type = text/xml`
- `Authorization = Basic <base64-encoded String "<Username>:<Password>">`

The request BODY must be a valid XMLA request, as illustrated in the following example:

#### Sample Code

```
<Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/">
<Header>
<Version Sequence="100" xmlns="http://schemas.microsoft.com/
analysisservices/2003/engine/2" />
</Header>
<Body>
<Execute xmlns="urn:schemas-microsoft-com:xml-analysis">
<Command>
  <Statement>
    SELECT { [K3].[K3].[All].[All] } on 0, {Measures.M1} on 1 FROM
    MDX_TEST_03_CV3J
  </Statement>
</Command>
<Properties>
  <PropertyList />
</Properties>
</Execute>
</Body>
</Envelope>
```

### Caution

The request you send **must** have a body. If you send an empty request (without a body), the XMLA service displays an error message, for example, "TypeError: Cannot read property 'substring' of undefined".

## Related Information

[Define the Data to Expose with an XMLA Service in XS Advanced \[page 496\]](#)

[The XMLA Service in XS Advanced \[page 500\]](#)

[Multidimensional Expressions \(MDX\) \[page 502\]](#)

### 6.2.2.1 The XMLA Service in XS Advanced

The XMLA service is defined by deploying the Node.js application `xmla`.

The XMLA service is defined by deployment of the Node.js application `xmla`. Exposed data is available for analysis and display by client applications, for example, a Web browser that uses functions provided either by the XMLA service running in SAP HANA XS or by an XMLA client library running on the client system.

To expose data to XS advanced applications using XMLA, you first need to define database views that provide the data with the required granularity. The XMLA service is used to control access to the exposed data.

### Restriction

XS advanced supports XMLA version 1.1, which you can use to send MDX queries to the database.

If the `xmla` is deployed to (and running in) the XS advanced run-time environment, the XMLA interface is available at the following URL:

`https://<SERVERNAME>:<PORT>/sap/hana/xmla/requestbasic`

Where `<SERVERNAME>` is the name of the host where your XS advanced instance is running, and `<PORT>` is the number of the port the XMLA service is listening on, for example:

`https://host1.acme.com:51028/sap/hana/xmla/requestbasic`

In XS advanced, you can use the `xs apps` command to list details of all XS advanced applications running in the current space; the information displayed includes the URL and port number to use for access to the application. For more information about a specific application running in XS advanced, use the `xs app <appName>`, as illustrated in the following example:

```
xs app xmla
Showing status and information about "xmla"
  name:          xmla
  requested state: STARTED
  instances:      1
```

```

memory:          1.00 GB
disk:            <unlimited>
buildpack:       <default>
urls:            https://host1.acme.com:51028
Instances of droplet 1 created at Sep 21, 2016 5:17:09 PM
index   created           state      host           internal
port    os     user
-----
1      Sep 30, 2016 2:43:10 AM    RUNNING    host1.acme.com:51028
50032          acmexsa

```

 **Tip**

If the `xmla` is not running, you need to deploy it, for example, with the `xs deploy` command.

Currently, access to the XMLA service in XS advanced is only granted to users registered with the XS advanced run time. You can clone to XS advanced all SAP HANA users that need to be able to use the XS advanced XMLA service, for example, with the *User Management* tool included in the XS Advanced Administration Tools. The basic authentication request is performed with this name and password of the XS advanced user.

## Related Information

[Define the Data to Expose with an XMLA Service in XS Advanced \[page 496\]](#)

[Set up and Use the XMLA Interface in XS Advanced \[page 497\]](#)

[The XS Command-Line Interface Reference \[page 848\]](#)

## 6.2.2.2 XML for Analysis (XMLA)

XML for Analysis (XMLA) uses Web-based services to enable platform-independent access to XMLA-compliant data sources for Online Analytical Processing (OLAP).

XMLA enables the exchange of analytical data between a client application and a multi-dimensional data provider working over the Web, using a Simple Object Access Protocol (SOAP)-based XML communication application-programming interface (API).

Implementing XMLA in SAP HANA enables third-party reporting tools that are connected to the SAP HANA database to communicate directly with the MDX interface. The XMLA API provides universal data access to a particular source over the Internet, without the client having to set up a special component. XML for Analysis is optimized for the Internet in the following ways:

- **Query performance**

Time spent on queries to the server is kept to a minimum

- **Query type**

Client queries are stateless by default; after the client has received the requested data, the client is disconnected from the Web server.

In this way, tolerance to errors and the scalability of a source (the maximum permitted number of users) is maximized.

## XMLA Methods

The specification defined in XML for Analysis Version 1.1 from Microsoft forms the basis for the implementation of XML for Analysis in SAP HANA.

The following list describes the methods that determine the specification for a stateless data request and provides a brief explanation of the method's scope:

- **Discover**

Use this method to query metadata and master data; the result of the `discover` method is a rowset. You can specify options, for example, to define the query type, any data-filtering restrictions, and any required XMLA properties for data formatting.

- **Execute**

Use this method to execute MDX commands and receive the corresponding result set; the result of the `Execute` command could be a multi-dimensional dataset or a tabular rowset. You can set options to specify any required XMLA properties, for example, to define the format of the returned result set or any local properties to use to determine how to format the returned data.

## Related Information

[Data Access with XMLA in SAP HANA XS \[page 495\]](#)

### 6.2.2.3 Multidimensional Expressions (MDX)

Multidimensional Expressions (MDX) is a language for querying multidimensional data that is stored in OLAP cubes.

MDX uses a multidimensional data model to enable navigation in multiple dimensions, levels, and up and down a hierarchy. With MDX, you can access pre-computed aggregates at specified positions (levels or members) in a hierarchy.

**i Note**

MDX is an open standard. However, SAP has developed extensions to MDX to enable faster and more efficient access to multidimensional data; for example, to serve specific SAP HANA application requirements and to optimize the result set for SAP HANA clients.

MDX is implicitly a hierarchy-based paradigm. All members of all dimensions must belong to a hierarchy. Even if you do not explicitly create hierarchies in your SAP HANA data model, the SAP HANA modeler implicitly generates default hierarchies for each dimension. All identifiers that are used to uniquely identify hierarchies, levels and members in MDX statements (and metadata requests) embed the hierarchy name within the identifier.

In SAP HANA, the standard use of MDX is to access SAP HANA models (for example, analytical and attribute views) that have been designed, validated and activated in the modeler in the SAP HANA studio. The studio provides a graphical design environment that enables detailed control over all aspects of the model and its language-context-sensitive runtime representation to users.

MDX in SAP HANA uses a runtime cube model, which usually consists of an analytical (or calculation) view that represents data in which dimensions are modeled as attribute views. You can use the analytical view to specify whether a given attribute is intended for display purposes only or for aggregation. The attributes of attribute views are linked to private attributes in an analytic view in order to connect the entities. One benefit of MDX in SAP HANA is the native support of hierarchies defined for attribute views.

#### i Note

MDX in SAP HANA includes native support of hierarchies defined for attribute views. SAP HANA supports level-based and parent-child hierarchies and both types of hierarchies are accessible with MDX.

SAP HANA supports the use of variables in MDX queries; the variables are an SAP-specific enhancement to standard MDX syntax. You can specify values for all mandatory variables that are defined in SAP HANA studio to various modeling entities. The following example illustrates how to declare SAP HANA variables and their values:

```
MDX
Select
From [SALES_DATA_VAR]
Where [Measures].[M2_1_M3_CONV]
SAP VARIABLES [VAR_VAT] including 10,
               [VAR_K2] including 112,
               [VAR_TARGET_CURRENCY] including 'EUR',
```

### 6.2.2.3.1 MDX Functions

MDX in SAP HANA supports a variety of standard MDX functions.

The following MDX functions are supported:

- Aggregate
- Ancestor
- Ancestors
- Ascendants
- Avg
- BottomCount
- Children
- ClosingPeriod
- Count
- Cousin
- Crossjoin
- CurrentMember
- DefaultMember
- Descendants
- Dimension
- Dimensions
- Distinct
- DistinctCount

DrillDownLevel  
DrillDownLevelBottom  
DrillDownLevelTop  
DrillDownMember  
DrillDownMemberBottom  
DrillDownMemberTop  
DrillUpLevel  
DrillUpmember  
Except  
Filter  
FirstChild  
FirstSibling  
Generate  
Head  
Hierarchize  
Hierarchy  
Instr  
Intersect  
IsAncestor  
IsGeneration  
IsLeaf  
IsSibling  
Item  
IIF  
Lag  
LastChild  
LastPeriods  
LastSibling  
Lead  
Leaves  
Left  
Level  
Levels  
Max  
Member\_caption  
Members  
MembersAscendantsDescendants  
Mid  
Min  
MTD  
Name  
NextMember  
NOT  
OpeningPeriod  
OR

Ordinal  
ParallelPeriod  
Parent  
PeriodsToDate  
PrevMember  
Properties  
QTD  
Range  
Right  
Siblings  
StrToMember  
StrToSet  
StrToTuple  
StrToValue  
Subset  
Sum  
Tail  
TopCount  
Union  
UniqueName  
WTD  
YTD

For more information about these functions, see Microsoft's Multidimensional Expressions (MDX) Reference.

### 6.2.2.3.2 MDX Extensions

SAP HANA supports several extensions to the MDX language, including additional predefined functions and support for variables.

#### 6.2.2.3.2.1 Sibling\_Ordinal Intrinsic Property

The object `Member` includes a property called `Sibling_Ordinal`, that is equal to the 0-based position of the member within its siblings.

##### Example

```
WITH
  MEMBER [Measures].[Termination Rate] AS
    [Measures].[NET_SALES] / [Measures].[BILLED_QUANTITY]
SELECT
{
  [Measures].[NET_SALES],
  [Measures].[BILLED_QUANTITY],
  [Measures].[Termination Rate]
} ON COLUMNS,
```

```

Descendants
(
    [DISTRIBUTION_CHANNEL].[DISTRIBUTION_CHANNEL].[All].[(all)],
    1,
    SELF_AND_BEFORE
)
DIMENSION PROPERTIES SIBLING_ORDINAL ON ROWS
FROM SALES_DATA

```

## 6.2.2.3.2.2 MembersAscendantsDescendants Function

SAP HANA includes the `MembersAscendantsDescendants` function that enables you to get, for example, all ascendants and descendants of a specific member.

This function improves on the standard MDX functions `Ascendants` and `Descendants`.

The function can be called as follows:

```
MembersAscendantsDescendants (<set>, <flag>)
```

- ***set***: A set of members from a single hierarchy
- ***flag***: Indicates which related members to return, and can be one of the following:
  - `MEMBERS_AND_ASCENDANTS_AND_DESCENDANTS`
  - `MEMBERS_AND_ASCENDANTS`
  - `MEMBERS_AND_DESCENDANTS`
  - `ASCENDANTS_AND_DESCENDANTS`
  - `ONLY_ASCENDANTS`
  - `ONLY_DESCENDANTS`

### Example

```

SELECT
{ [Measures].[SALES] }
ON COLUMNS,
NON EMPTY
{ Hierarchize( MembersAscendantsDescendants([SALES_DATA_TIME].[TimeHier].[QUARTER].[3]:[SALES_DATA_TIME].[TimeHier].[QUARTER].[4], MEMBERS_AND_ASCENDANTS_AND_DESCENDANTS) ) }
ON ROWS
FROM [SALES_DATA]

```

### Example

```

SELECT
{ [Measures].[SALES] }
ON COLUMNS,
NON EMPTY
{ Hierarchize( MembersAscendantsDescendants([SALES_DATA_TIME].[TimeHier].[QUARTER].[3]:[SALES_DATA_TIME].[TimeHier].[QUARTER].[4], ONLY_ASCENDANTS) ) }
ON ROWS
FROM [SALES_DATA]

```

### 6.2.2.3.2.3 Variables in MDX

An MDX SELECT statement in SAP HANA enables you to send values for variables defined within modeling views.

Analytic and calculation views can contain variables that can be bound to specific attributes. When calling the view, you can send values for those variables. These variables can be used, for example, to filter the results.

SAP HANA supports an extension to MDX whereby you can pass values for variables defined in views by adding an SAP Variables clause in your SELECT statement. Here is the syntax for a SELECT statement:

```
<select_statement>:  
[WITH <formula_specification> ]  
SELECT [ <axis_specification>[,<axis_specification>...]]  
    FROM <cube_specification>  
    [WHERE <slicer_specification>  
  
SAP VARIABLES: <sap_variable> [ [,] <sap_variable>...]  
<sap_variable>: <variable_name> <sign> [<option>] <variable_value>  
<sign>: INCLUDING | EXCLUDING  
<option>: = | > | >= | < | <= | <>  
<variable_value>:  
    <unique_member_name>  
    | <unsigned_numeric_literal>  
    | <string_value_expression>  
    | <member> : <member>  
    | <character_string_literal> : <character_string_literal>  
    | <unsigned_numeric_literal> : <unsigned_numeric_literal>
```

#### Example

The following statement specifies a single value for variables VAR\_VAT, VAR\_K2, and VAR\_TARGET\_CURRENCY.

```
SELECT  
FROM [SALES_DATA_VAR]  
WHERE [Measures].[M2_1 M3_CONV]  
SAP VARIABLES [VAR_VAT] including 10,  
              [VAR_K2] including 112,  
              [VAR_TARGET_CURRENCY] including 'EUR'
```

#### Example

The following specifies an interval for variable VAR\_K2.

```
SELECT NON EMPTY  
{  
    [K2].[K2].Members  
} ON ROWS  
FROM [SALES_DATA_VAR_SIMPLE]  
WHERE [Measures].[M3_CONV]  
SAP VARIABLES [VAR_K2] including [K2].[K2].&[122]:[K2].[K2].&[221]
```

## Metadata on Variables in Views

SAP HANA includes the following set of tables that contain information about the variables defined for views:

- BIMC\_VARIABLE
- BIMC\_VARIABLE\_ASSIGNMENT
- BIMC\_VARIABLE\_VALUE

The tables enable, for example, an application to retrieve the variables defined for a view and create a user interface so the user can enter values.

# 7 Writing the XS Advanced Application Code

Add business logic to your XS advanced application to enable it to work with the database artifacts.

The business logic is used to help retrieve data from the database, for example, using OData services. The data is requested from and displayed in the client user interface.

The SAP HANA XS, advanced model, run-time platform provides a number of services for managing the various container instances and their application run times. Containers are used to manage run times and allow for isolation, resource management, and shared service injection. The XS advanced application run times are lightweight processes that are invoked over HTTP and communicate remotely with the database. Execution environments such as JavaScript, including Node.js, and Java are supported.

The application services can be used to expose the database data model, with its tables, views and database procedures, to clients. This can be done in a declarative way using OData services or by writing native application-specific code that runs in the SAP HANA context. You can also use SAP HANA XS advanced model to build dynamic HTML5 UI applications.

## JavaScript Applications

For applications written in JavaScript (Node.js or XS JavaScript):



```
Sample Code

JavaScript-AppName
|- db/
|   |- package.json
|   |- src/
|       |- .hdiconfig
|       \- mytable.hdbdd
|- web/
|   |- xs-app.json
|   |- package.json
|   \- resources/
|- js/                      # JavaScript artifacts
|   |- start.js              # JavaScript application entry point
|   |- package.json          # JavaScript application details/dependencies
|   \- src/                  # JavaScript source code
|- security/
|   \- xs-security.json
\- mtad.yaml
```

- Add a package description file (package.json) to your JavaScript resource-files folder with details of your application's contents and dependencies.
- Add a start-up file for your JavaScript application, for example, main.js.
- Create express routes.
- Create JavaScript source files (.js or .xsjs). These files must be placed in the corresponding application module's folder, for example, js/src/\*.js or xsjs/src/\*.xsjs.

## Java Applications

For applications written in Java, a multi-target application would have the following design-time structure:

### Sample Code

```
Java-AppName
|- db/
|   |- package.json
|   |- src/
|       |- .hdiconfig
|       \- mytable.hdbdd
|- web/
|   |- xs-app.json          # Application-router configuration
|   \- resources/
|- java/
|   |- pom.xml              # Java artifacts
|   \- src/                  # Java project object model (Maven)
|       |- security/         # Java source code
|           \- xs-security.json
\- mtad.yaml
```

- Write the Java application's code. These files must be placed in the corresponding application module's folder, for example, `java/src/*`
- Look up the required data-source.
- Build the WAR file based on the project object model file (`pom.xml`), for example, with Maven.
- Use the Java Persistence API (JPA) to work with CDS entities (if required).

## Related Information

[Create the XS Advanced Application Package Descriptor \[page 652\]](#)

[XS Advanced Application Resource Files \[page 110\]](#)

[The SAP HANA XS Advanced JavaScript Run Time \[page 510\]](#)

[The SAP HANA XS Advanced Java Run Time \[page 589\]](#)

## 7.1 The SAP HANA XS Advanced JavaScript Run Time

SAP HANA XS advanced model provides a JavaScript run time environment to which you can deploy your Node.js and XS JavaScript applications.

SAP HANA XS advanced (XS advanced) enables you to build and deploy applications on Node.js. XS advanced makes no assumption about the frameworks you are using. However, it is recommended to connect to (and use) the SAP HANA deployment infrastructure (HDI) container and validate using JSON Web Tokens (JWT).

Authentication for node applications in XS advanced relies on a special usage of the OAuth 2.0 protocol by the User Account and Authorization service (UAA). The UAA vouches for the authenticated user's identity using a JSON web token (JWT) as OAuth 2.0 token. This is a signed text-based token in JSON syntax. The node.js

application is specified in the related manifest file. For more information, see the README.md file of the `@sap/xssec` module in the XS advanced Container Security API, for example, for your node.js application.

## Node.js Modules for Download

A collection of node.js packages developed by SAP is provided as part of XS advanced; the Node.js packages are available for download either from the SAP NPM public registry at `npm.sap.com` or, for customers and partners who have the appropriate access authorization, from the SAP Service Marketplace (SMP). Search for the software component named `XS_JAVASCRIPT`; it includes packages for the application router, job scheduling, logging, text mining and analysis, and so on. The following list shows a selection of the more commonly used packages included:

- `@sap/approuter`
- `@sap/hdbext`
- `@sap/jobs-client`
- `@sap/logging`
- `@sap/textbundle`
- `@sap/textanalysis`
- `@sap/textmining`
- `@sap/xsenv`
- `@sap/xsjs`
- `@sap/xsssecure`
- `@sap/xssec`

### ➔ Tip

For more information about all the Node.js packages included in the latest version of the component `XS_JAVASCRIPT` or more information about configuring and using the SAP NPM Registry, see *Related Links*.

To install an `npm` module use the following command:

```
npm install --save @sap/xsenv
```

To use a module, for example `@sap/xsenv`:

```
var xsenv = require('@sap/xsenv');
```

## SAP HANA Database Connections

To establish a connection to the SAP HANA database, a JavaScript application must first look up the connection properties from the bound services and create an “`hdb`” client. For Node.js applications, the easiest way to do this is to add a dependency to `@sap/hdbext` to the application’s `package.json` file, as illustrated in the following example:

## Note

You might need to modify the version of `@sap/hdbext` to reflect the version required by your application.

## Sample Code

```
"dependencies": {  
  "@sap/hdbext": "4.0.1"  
},
```

Next, the application has to create a connection, as illustrated in the following example:

## Sample Code

```
var hdbext = require('@sap/hdbext');  
var hanaConfig = { ... };  
hdbext.createConnection(hanaConfig, function(error, client) {  
  if (error) {  
    return console.error(error);  
  }  
  client.exec(...);  
});
```

## Sample Code

```
var xsConnection = require("@sap/hdbext");  
var hdbcclient = xsConnection.createConnection(function(error) { ... });
```

## Connection Pools

The `@sap/hdbext` module includes a simple method for pooling connections. To use the connection-pool feature, you must first create the connection pool, as illustrated in the following example:

```
var pool = hdbext.getPool(hanaConfig, poolConfig);
```

Then you can acquire a connection client from the pool; the client is delivered in a callback, as illustrated in the following example:

```
pool.acquire(function(err, client) {});
```

If the connection client is no longer needed, it should be released back to the pool:

```
pool.release(client);
```

## Security

For Node.js, the client security library is an npm module called `@sap/xssec`. If you want your application to use the security library `@sap/xssec`, you must add the dependency to `@sap/xssec` in the `dependencies` section of your application's package description file, `package.json`, as illustrated in the following example:

### Sample Code

```
"dependencies": {  
    "@sap/xssec": "^1.0.5"  
},
```

## Outbound Connectivity

If a Node.js application needs to call external applications or services, it must do so by performing HTTP requests to the required services. The service can either be located in the same system (for example, an SAP HANA XS advanced system) or on an external system on the Internet. For connections to remote systems, Node.js applications can make use of the “`request`” module. For outbound connections, you need to consider the following:

- Proxy connections  
If your Node.js applications runs behind an HTTP proxy, the details of the proxy connection must be provided to the `request` module. Request can discover this information either from environment variables set in the application's manifest file or directly from the options object of the HTTP request.
- Authentication  
If the application needs to call a service that performs authentication by means of Security Assertion Markup Language (SAML) and JWT (authentication), and the calling service does so as well, then the SAML token can be taken from the incoming request and attached to the outgoing request.

### Sample Code

```
var options = {  
    url: remoteURL,  
    headers: {  
        'authorization': req.headers.authorization  
    }  
};
```

### Note

For this so called “principal propagation” to work, both the calling and the called application must be bound to the same User Account and Authentication (UAA) service. Otherwise the SAML token attached to the request will not be recognized and the request cannot be authenticated.

## Related Information

[Writing the XS Advanced Application Code \[page 509\]](#)

[The SAP HANA XS Advanced Java Run Time \[page 589\]](#)

[Standard Node.js Packages for XS Advanced \[page 519\]](#)

[The SAP NPM Registry \[page 560\]](#)

### 7.1.1 Secure Programming with JavaScript

Web-based applications for deployment to the XS advanced run-time must adhere to the general standard for Web application security as defined by the W3C.

To ensure the best possible protection against attacks, Web-based SAP HANA applications must implement not only **primary** security measures such as input validation or output encoding, but also secondary measures, for example a Content Security Policy (CSP) that helps defend against script-based injection vulnerabilities. In addition to any other security measures, developers of SAP HANA XS advanced applications must ensure that the Web-based applications comply with the following basic rules:

- Avoid using inline `JavaScript()` functions in a Web page.
- Avoid using the `eval()` command.
- Avoid using strings (that represent code to be executed) in place of simple functions in code generating-functions such as `setTimeout()`, `setInterval()`, or `function()`.
- Ensure that platforms provide HTTP header-setting functions which perform the following actions:
  - Enable CSP headers that can be maintained or customized (for example, as a configuration option). This is possible in either of the following ways:
    - Indirectly  
CSP settings are derived from higher-level configurations, for example, system landscape data.
    - Directly  
CSP settings are derived from modifications to the configuration in more simple environments.
  - Enable applications to set CSP headers (for example, as an API option).
- Make use of functions for controlling frame inclusion (for example, frame “ancestors”).
- Ensure the implementation of primary protection measures such as “input validation” and “output encoding”.

#### i Note

A CSP compliant application will only work if the underlying platform is also CSP compliant. For example, a CSP-compliant Web browser will not execute instructions that contradict the defined security policy.

## Recommended Coding Rules for JavaScript Applications

It is recommended to adhere to the following rules when coding JavaScript applications that are rendered by HTML in a Web browser, for example: SAPUI5-based Fiori applications, mobile applications using HTML technology, JSP applications, Web Dynpro, BSP applications, EP iViews:

- Security policy:

The CSP should be as close as possible to the following example:

```
Content-Security-Policy: default-src 'self'; style-src 'self' 'unsafe-inline'; img-src 'self' data:
```

### Note

Certain adjustments might be needed to help ensure functionality, for example, adding source lists for script-src.

- <script>

Loading scripts from a white-listed source is allowed using the <script> tag, for example:

```
<script src = "foo.js" type = "text/javascript" > </script >
```

- onClick()

To handle events, you can set the event handler attributes of elements, or call element.addEventListener(), from a script that has been loaded from a white-listed site, as illustrated in the following example:

```
element.onclick = someFunction;
element.addEventListener( "click" , someFunction, false ) ;
```

- setInterval()

```
window.setInterval( myFunc, 10000 ) ;
```

- setTimeout()

```
self.timers.invite2xxTimer = window.setTimeout(function()
{invite2xxRetransmission(retransmissions)}, timeout);
```

## Coding Practices to Avoid in JavaScript Applications

It is recommended to avoid the following commonly used practices when coding Web-based JavaScript applications for SAP HANA XS advanced:

- <script>

Avoid including any executable code in an HTML string:

```
<script type = "text/javascript" > /* executable code */ </script >
```

- onClick()

Avoid including any executable code in an HTML string:

```
<a onclick = "another_evil_script()" > Click this for some awesome fun! </a >
```

Avoid specifying a JavaScript command as part of an HTML attribute; this is forbidden by the CSP policy:

```
<a href = "javascript:do_something_evil()" > Click here for some awesome fun!
</a >
```

- `setInterval()`

Avoid including any executable code in an HTML string:

```
window. setInterval( "evil_code()", 10000 ) ;
```

- `setTimeout()`

In the following short example, if “`invite2xxRetransmission`” returns a string instead of a pointer, it will be still evaluated using the `eval()` function:

```
self.timers.invite2xxTimer =
window.setTimeout(invite2xxRetransmission(retransmissions), timeout);
```

## Related Information

[Getting Started with Application Development in XS Advanced \[page 25\]](#)

[The SAP HANA XS Advanced JavaScript Run Time \[page 510\]](#)

## 7.1.2 Download and Consume SAP Node.js Packages

A selection of SAP-specific and ready-to-use Node.js packages is available for download from the SAP public NPM registry at `npm.sap.com` or from the SAP Service Marketplace.

## Context

A Node.js application often depends on packages that are present in the public `npm` registry or on specific SAP modules. A selection of SAP packages for Node.js is available for use in your Node.js applications. If your Node.js application depends on any of the SAP NPM packages, you need to configure a source to resolve the dependencies. The SAP NPM packages can be found either in the SAP public NPM registry, which is the recommended source, or in the `xs_JAVASCRIPT` archive, which you can download from SAP Service Marketplace (`service.sap.com`).

### ➔ Tip

For information about how to configure your Node.js application to use the SAP NPM registry at `npm.sap.com` to resolve package requirements and dependencies, see *The SAP NPM Registry* in *Related Information* below.

The following example shows a setup where the XS\_JAVASCRIPT was downloaded and the required content copied from the downloaded archive to the application's design-time infrastructure:

```
JavaScript-hello-world
|- db/
|- web/
|- js/
|   |- .cignore
|   |- .gitignore
|   |- start.js
|   |- package.json
|   |- node_modules/
|   |   \- @sap
|   |       \- xsjs/
|   |           \- package.json
|   \- src/
|
|- xs-security.json
\- mtad.yaml
```

# Node.js artifacts  
# Filter for JS build (optional)  
# Filter for GIT configuration (optional)  
# Node.js application entry point  
# Node.js application details/dependencies  
# Downloaded SAP Node.js packages (XS\_JAVASCRIPT)  
  
# Installed SAP Node.js package (@sap/xsjs)  
# This is the package.json file for @sap/xsjs  
# JavaScript source code

The following instructions explain how to download the XS\_JAVASCRIPT and configure the dependency to the extracted contents:

## Procedure

1. Check that a directory named `node_modules` exists in the `js/` module of your Node.js application.  
If the directory `node_modules` does not exist, you must create it. The `node_modules` directory must exist in the same location as the Node.js application's core `package.json` file, for example, `node-hello-world/js/node_modules`.
2. Check any exclusion filters (for example, for builds or version control).  
If you have set up any exclusion filters (for example, in the `.cignore` or `.gitignore` files), make sure that the filter lists do not include the `node_modules` directory.
3. Download the software component "XS\_JAVASCRIPT" from the SAP Service Marketplace.

### Restriction

Access to the SAP Marketplace is only permitted to authenticated users with the appropriate authorization.

4. Copy the relevant packages from XS\_JAVASCRIPT into your Node.js application's design-time infrastructure.  
Select the packages you want to use in your XS advanced Node.js application and copy each package directory from the XS\_JAVASCRIPT archive into the `node_modules/` directory of your Node.js application. For example if you need `@sap/xsjs` in your application, copy the `@sap/xsjs` directory to `node_modules/@sap/xsjs`.

### Note

It is essential to retain the original package directory name in the target location, for example, copy the folder `@sap/xsjs` to `node_modules/@sap/xsjs`.

- Check that the names and versions of the downloaded packages match those specified in the `package.json` file located in the root folder of the application's JavaScript module.

Add the name and version of each of the new installed packages to the Node.js application's `package.json` (js/`package.json`) and ensure that the names and versions match the installed packages.

For example, if you installed the SAP Node.js module `@sap/xsjs`, then check the contents of the file `node_modules/@sap/xsjs/package.json`. If the file `node_modules/@sap/xsjs/package.json` contains the following name and version information:

#### Sample Code

```
node_modules/@sap/xsjs/package.json
```

```
"name": "@sap/xsjs",
"version": "1.2.5"
```

Then, add the following line to the Node.js package-dependencies file `js/package.json`

#### Sample Code

```
js/package.json
```

```
"@sap/xsjs": "1.2.5"
```

- Download any additional dependent packages (from the public `npm` registry).

Since the `node_modules` directory is present in your application's design-time infrastructure, the Node.js build pack does not run the `npm install` command during deployment of the application to the XS advanced JavaScript run time. As a result, any missing dependencies are not downloaded.

#### Remember

Run the `npm install` locally before deploying the Node.js application with `xs push`.

## Related Information

[Standard Node.js Packages for XS Advanced \[page 519\]](#)

[The SAP NPM Registry \[page 560\]](#)

[The SAP HANA XS Advanced JavaScript Run Time \[page 510\]](#)

## 7.1.2.1 Standard Node.js Packages for XS Advanced

A collection of Node.js packages developed by SAP is provided to help you develop Node.js applications for SAP HANA XS advanced.

SAP HANA includes a selection of standard Node.js packages, which are available for download and use either from the SAP NPM public registry at `npm.sap.com` or, for customers and partners who have the appropriate access authorization, from the SAP Service Marketplace (SMP). For more information about the SAP NPM public registry, see *Related Information* below; to download and use Node.js packages from the SAP Service Marketplace, log on to the SMP and search for the software component `XS_JAVASCRIPT`, which is an archive that contains the most commonly used SAP NPM packages. The following table lists the SAP Node.js packages that are currently available and indicates where you can find them. For more details about the contents of each Node.js packages as well as any configuration tips, see the `README` file in the corresponding package.

**i Note**

Some packages are available for download but are neither listed nor documented here, for example, packages that are included in the archive or registry only because they are required by other packages.

Table 68: SAP Node.js Packages and Contents

Package Name	XS_JAVA-SCRIPT Archive	SAP NPM Registry	Description
<code>@sap/audit-logging</code>	✓	✓	Utilities for audit logging
<code>@sap/approuter</code>	✓	✓	The application router is the single entry point for the (business) application. It serves static content, authenticates users, rewrites URLs, and routes proxy requests to other micro services as well as propagating user information.
<code>@sap/cds</code>	✓	✓	Core Data Services (CDS) client for Node.js
			<p><b>i Note</b></p> <p>The <code>@sap/cds</code> library is now considered feature complete. It will remain fully supported but will not receive any further enhancements in future releases.</p>
<code>@sap/dwf-deploy</code>	✓	✓	Node.js client for SAP HANA Datawarehousing Foundation 2.0 (HANA DWF 2.0) to deploy designtime artifacts
<code>@sap/dwf-generator</code> [page 523]	✓	✓	Node.js client for SAP HANA Datawarehousing Foundation 2.0 (HANA DWF 2.0) to generate designtime artifacts

Package Name	XS_JAVA-SCRIPT Archive	SAP NPM Registry	Description
<a href="#">@sap/dwf-dws-client [page 524]</a>	✓	✓	Node.js client for SAP HANA Datawarehousing Foundation 2.0 (HANA DWF 2.0) to host task types for the Task Orchestration Engine
<a href="#">@sap/e2e-trace</a>	✓	✓	Utilities for end-to-end tracing (in particular the handling of SAP Passports)
<a href="#">@sap/hdb-connection</a>	✓	-	Utility functions for access to SAP HANA in Node.js
			<p><b>i Note</b></p> <p>The functions included in the <code>@sap/hdb-connection</code> library have been superseded; use <code>@sap/hdbext</code> instead.</p>
<a href="#">@sap/hdbext</a>	✓	✓	Extends the functionality of the <code>hdb</code> package, which is a JavaScript client for SQLDBC.
			<p><b>i Note</b></p> <p>The functionality included in the <code>@sap/hdbext</code> library replaces and supersedes the functions previously provided by the <code>@sap/hdb-connection</code> library.</p>
<a href="#">@sap/hdi-deploy</a>	✓	✓	The Node.js based "HDI deploy" application (also known as "DeployApp") which is based on HDI's SQL interface
<a href="#">@sap/hdi</a>	✓	✓	A Node.js-based client library for SAP HANA DI (HDI)
<a href="#">@sap/instance-manager</a>	-	✓	Node.js package for creating and deleting service instances per tenant within an XS advanced application at runtime.
<a href="#">@sap/jobs-client</a>	✓	✓	Integrate jobs in your Node.js application
<a href="#">@sap/logging</a>	✓	✓	Provides logging and tracing functions for Node.js applications. Logs and traces are written in the standard SAP format.
<a href="#">@sap/node-vsi</a>	-	✓	A Node.js client that contains the Virus Scanning Interface (VSI) binding for Node.js
<a href="#">@sap/sds-deploy</a>	✓	-	Node.js client for the SAP HANA Streaming Analytics option
<a href="#">@sap/site-entry</a>	✓	✓	SAP Portal site entry point

Package Name	XS_JAVA-SCRIPT Archive	SAP NPM Registry	Description
<a href="#">@sap/site-content-deployer</a>	✓	✓	Client for portal site content deployer
<a href="#">@sap/site-app-server</a>	✓	✓	An application server for the independent apps scenario on the SAP portal
<a href="#">@sap/textanalysis</a>	✓	✓	A utility for performing text analysis on unstructured text
<a href="#">@sap/textmining</a>	✓	✓	A utility for performing text mining on unstructured text
<a href="#">@sap/textbundle</a>	✓	✓	A simple tool for text internationalization in Node.js.
<a href="#">@sap/ui-annotations</a>	✓	-	Annotations for OData version 4 in Core Data Services (CDS)
<a href="#">@sap/xsenv</a>	✓	✓	Utility for easy setup and access of XS Advanced environment variables and services
<a href="#">@sap/xsjs</a>	✓	✓	Compatibility layer for SAP HANA extended application services, classic model (SAP HANA XS Classic) applications to run on JavaScript runtime in SAP HANA extended application services, advanced model (SAP HANA XS Advanced).
<a href="#">@sap/xsjs-test</a>	✓	✓	Unit test framework for the compatibility layer (XS classic runtime)
<a href="#">@sap/xssec</a>	✓	✓	XS advanced security API for Node.js, including the XS advanced container security API for Node.js
<a href="#">@sap/xss-secure</a>	✓	✓	Utilities for protection against cross-site scripting
<a href="#">@sap/xb-msg</a>	✓	-	Provides messaging capabilities with a message broker.
<a href="#">@sap/xb-msg-amqp-v091</a>	✓	✓	Provides a client implementation for AMQP v0.9.1
<a href="#">@sap/xb-msg-mqtt-v311</a>	✓	✓	Provides a client implementation for MQTT v3.1.1

## @sap/approuter

The application router is the single entry point for the (business) application. The main tasks of the application router are to serve static content, authenticate users, rewrite URLs, and proxy requests to other micro services while propagating user information. For more information about the Node.js application router and how to configure it for your XS advanced application, see the section *Maintaining XS Advanced Application Routes and Destinations* in this guide or the *Related Links* section below.

## @sap/cds

The Core Data Services for Node.js (@sap/cds) comprise a JavaScript client library for Core Data Services that enable Node.js applications to consume CDS artifacts natively in Node.js applications.

### Usage

#### Sample Code

```
var cds = require('@sap/cds');
```

CDS entities are imported by name. The import function takes a callback that is invoked when all imports have completed. Additional fields and overrides can be supplied for each entity.

#### i Note

The import operation is a regular asynchronous Node.js function; you can import entities at any point in time, and in as many calls as you want.

#### Sample Code

```
cds.importEntities([
  { $entity: "xsds.test.cds::ds_test.e1" },
  { $entity: "xsds.test.cds::ds_test.e2",
    $fields: {
      a: { $association: "xsds.test.cds::ds_test.e2",
            $viaBacklink: "b" }
    }
  }
], callback);
function callback(error, entities) {
  var E1 = entities["xsds.test.cds::ds_test.e1"];
  var E2 = entities["xsds.test.cds::ds_test.e2"];
  // ...
}
```

## @sap/dwf-deploy

This package includes a Node.js client for SAP HANA Datawarehousing Foundation (HANA DWF). The following table describes the file-system layout required by the `HANA_DWF_Client` application:

Table 69: HANA DWF Client File System Layout

File	Mandatory	Description
<code>package.json</code>	Yes	Used by the Node.js package manager ( <code>npm</code> ) to boot and start the application.
<code>src/</code>	Yes	Subdirectory containing DWF deployment definition files (optional, additional subfolders are allowed).
<code>cfg/</code>	No	Additional deployment subdirectories ...

**i Note**

All DWF-related files in the `root` directory are ignored by the `HANA_DWF_Client` application.

## @sap/dwf-generator

This package includes a Node.js client for SAP HANA Datawarehousing Foundation 2.0 (HANA DWF 2.0) to generate designtime artifacts. The following table describes the file-system layout required by the `HANA_DWF_Client` application:

Table 70: HANA DWF Client File System Layout

File	Mandatory	Description
<code>package.json</code>	Yes	Used by the <code>dwf-generator</code> to discover the <code>dwf-config</code> .
<code>src/</code>	Yes	Subdirectory containing DWF deployment definition files (optional, additional subfolders are allowed).
<code>src/dlm_generated</code>	Yes	Subdirectory containing XSA or DWF deployment definition files (optional, additional subfolders are allowed).
<code>cfg/</code>	No	Additional deployment subdirectories, which are created in the designtime

File	Mandatory	Description
cfg/dlm_generated	No	Additional deployment subdirectories, which are generated, based on the deployment definition files that have been created in the designtime.

### i Note

All DWF-related files in the `root` directory are ignored by the `HANA DWF Client` application.

## @sap/dwf-dws-client

This package includes a Node.js client for SAP HANA Datawarehousing Foundation 2.0 (HANA DWF 2.0) to host task types for the Task Orchestration Engine. The module does not imply any requirements to the file system structure.

## @sap/hdb-connection

This package comprises a small library that enables you to establish a connection to an SAP HANA server using the credentials either from stand-alone environments or, if required, from a Cloud Foundry environment, for example defined in the “`VCAP_SERVICES`” environment variables.

### ⚠ Restriction

The functions included in the `@sap/hdb-connection` library have been superseded; use `@sap/hdbext` instead.

## @sap/hdbext

This is a small Node.js package that extends the functionality of the publicly available `hdb` package; `hdb` is a JavaScript client for Node.js, which implements the SAP HANA Database SQL Command Network Protocol. The functionality included in the `@sap/hdbext` library replaces and supersedes the functions previously provided by the `@sap/hdb-connection` library.

### Usage

#### Sample Code

```
var hdbext = require('@sap/hdbext');
```

The following code example shows how to create a connection to the SAP HANA database:

#### Sample Code

```
var hanaConfig = {
  host : 'hostname',
  port : 30015,
  user : 'user',
  password : 'secret'
};
hdbext.createConnection(hanaConfig, function(error, client) {
  if (error) {
    return console.error(error);
  }
  client.exec(...);
});
```

If you are using table parameters, with the new API you can pass an array of objects and it will auto convert it into a table parameter, as illustrated in the following example:

#### Sample Code

```
create procedure PROC_DUMMY (in a int, in b int, out c int, out d DUMMY, out e
TABLES)
language sqlscript
reads sql data as
begin
  c := :a + :b;
  d = select * from DUMMY;
  e = select * from TABLES;
end
```

With the `loadProcedure()` API, you can make the following call:

#### Sample Code

```
hdbext.loadProcedure(client, 'MY_SCHEMA', 'PROC_DUMMY', function(err, sp) {
  sp({ A: 3, B: 4 }, function(err, parameters, dummyRows, tableRows) {
    if (err) {
      return console.error(err);
    }
    console.log('C:', parameters.C);
    console.log('Dummies:', dummyRows);
    console.log('Tables:', tableRows);
  });
});
```

To use connection pooling, first create the pool, as illustrated in the following example:

#### Sample Code

```
var pool = hdbext.getPool(hanaConfig, poolConfig);
```

To acquire a client from the connection pool, use the `pool.acquire` function, as illustrated below:

#### Sample Code

```
pool.acquire(options, function(err, client) {});
```

#### Note

Use the `options` object if you need to change the settings configured for the pooled connection.

If the client is no longer needed, release it to the connection pool, as illustrated in the following example:

#### Sample Code

```
pool.release(client);
```

Alternatively, you can close the client connection, as illustrated below:

#### Sample Code

```
client.close();
```

## @sap/hdi-deploy

`@sap/hdi-deploy` is the Node.js deploy application (known as “`DeployApp`”) for the SAP HANA Deployment Infrastructure (HDI), which is based on the HDI’s SQL interface. The following table describes the file-system layout required by the `DeployApp` application:

Table 71: HDI Deployer File System Layout

File	Mandatory	Description
<code>package.json</code>	Yes	Used by the Node.js package manager ( <code>npm</code> ) to boot and start the application.
<code>src/</code>	Yes	Subdirectory containing <code>.hdiconfig</code> and HDI deployment definition files (optional, additional subfolders are allowed).
<code>cfg/</code>	No	Additional deployment subdirectories ...

#### Note

All HDI-related files in the `root` directory are ignored by the `DeployApp` application.

For more information about setting up and using the Node.js-based SAP HANA Deployment Infrastructure (HDI) Deployer for XS advanced, see *Configuring the HDI Deployer* in the *Related Links* below.

## Environment Variables

The following table lists the environment variables that can be used with the DeployApp application:

Table 72: HDI Deployer Environment Variables for XS Advanced Applications

Variable	Mandatory	Description
TARGET_CONTAINER	Yes	Service name that specifies the HDI target container (required if more than one service is bound to the DeployApp application).
SERVICE_REPLACEMENTS	No	<p>JSON-structured list of service replacements,</p> <pre>[   {     "key": "logical-service-name-1",     "service": "real-service-name-1"   },   {     "key": "logical-service-name-2",     "service": "real-service-name-2"   } ]</pre> <p>The <b>logical</b> service names refer to the names in the HDI content; the <b>real</b> service names refer to the services which are bound to the HDI Deployer via VCAP_SERVICES. If the HDI content references a service name which is not listed in the replacements, this name is used as a real service name.</p>

The structure of the <`SERVICE_REPLACEMENTS`> environment variable illustrated in the following example is intended to enable and facilitate MTA group assignments.

### Code Syntax

Service Replacements list in the `manifest.yml` file

```
applications:  
- name: app-db  
  path: db  
  services:  
    - app-database  
    - real-grantor-service  
    - real-external-service  
env:  
  TARGET_CONTAINER: app-database  
  SERVICE_REPLACEMENTS:  
  [  
    {  
      "key" : "logical-grantor-service",  
      "service" : "real-grantor-service"  
    },  
    {  
      "key" : "logical-external-service",  
      "service" : "real-external-service"  
    }  
  ]
```

## Removing Deployed SAP HANA Node.js Artifacts

The DeployApp deletes (recursively) all artifacts found in the folder `src/` (and `cfg/`, if the optional subfolder exists) before the deployment of changed or new artifacts, but deleted artifacts are not removed; they remain deployed. To remove deployed artifacts, you have the following options:

- Specify the deployed files to remove in an `undeploy.json` file placed in the root directory of the deployment artifacts; the file lists the paths to and the name of the artifacts to remove:

### Sample Code

```
[  
  "src/AddressBook.hdbcds"  
]
```

- Start the DeployApp with the `--autoUndeploy` option  
`node deploy --autoUndeploy`

## @sap/hdi

This module provides a Node.js-based client library for SAP HANA DI (HDI). The client library provides the following asynchronous methods to enable access to HDI functionality:

- `connect`
- `disconnect`
- `configureDI`

### ⚠ Caution

Deprecated; use `configureDIParameters` instead.

- `configureDIParameters`
- `createContainer`
- `dropContainer`
- `configureContainer`

### ⚠ Caution

Deprecated; use `configureContainerParameters` instead.

- `configureContainerParameters`
- `listLibraries`
- `configureLibraries`
- `listConfiguredLibraries`
- `status`
- `read`
- `listDeployed`  
Read object-related metadata.
- `readDeployed`

Read object-related metadata **and** its content.

- `write`
- `delete`
- `make`
- `makeAsync`
- `grantContainerApiPrivileges`
- `grantContainerApiPrivilegesWithGrantOption`
- `revokeContainerApiPrivileges`
- `grantContainerSchemaPrivileges`
- `revokeContainerSchemaPrivileges`
- `grantContainerSchemaRoles`
- `revokeContainerSchemaRoles`

## Overview

After installing the module using `npm`, an application must create an instance of the client, for example:

```
hdi = new HDI(container, logger, credentials);
```

In this example:

- “`container`”  
The name of an existing HDI container
- “`logger`”  
A callback function used to write logging information
- “`credentials`”  
An object containing the host name, port, user, and password that are used to establish the connection to the database, for example:

```
{ host : '<myhost.kumasi.acme.com>',
  port : 30015,
  user : 'Kwame',
  password : 'etisei!131'}
```

The connection to the database is established with the `connect()` method, which takes only a callback function as parameter. When the application finishes, it should close the connection by calling the method:

```
hdi.disconnect();
```

## @sap/instance-manager

`@sap/instance-manager` is a Node.js package for creating and deleting service instances per tenant within an XS advanced application at runtime.

### Usage

The `@sap/instance-manager` package provides a client for the Instance Manager: a component that creates and deletes service instances (by means of a REST API) for a specified key. Instance Manager can be used in the context of multi-tenant applications where the key an instance is associated with is the tenant ID.

With the `@sap/instance-manager` package, a Node.js application can dynamically create and delete service instances per tenant at run time. An instance of a managed service of the desired type is first created and then bound to the application. For an SAP HANA database, the managed service is called "managed-hana". The service binding only provides parameters (for example, HTTP end points and credentials) which can later be used by the application for creating and deleting service instances of the desired type for each tenant.

## Application Programming Interface (API)

The following example shows how to use the instance manager API to manage service instances for the XS advanced application used by multiple tenants:

### Sample Code

#### XS Advanced Service-Instance Manager API

```
var createInstanceManager = require('@sap/instance-manager').create;
var options = { /* properties from service binding */ };
createInstanceManager(options, function (err, instanceManager) {
  if (err) {
    return console.log('Create instance manager error:', err.message);
  }
  instanceManager.create('my-tenant', function (err, instance) {
    if (err) {
      return console.log('Create error:', err.message);
    }
    // consume instance.credentials
    console.log(instance);
    instanceManager.get('my-tenant', function (err, instance) {
      if (err) {
        return console.log('Get error:', err.message);
      }

      // same instance console.log(instance);
      instanceManager.delete('my-tenant', function (err) {
        if (err) {
          return console.log('Delete error:', err.message);
        }
        console.log('Instance deleted');
      });
    });
  });
});
```

## Options

The managed service that is bound to the multi-tenant application (for example "managed-hana") provides credentials as well as REST-based end points of the Instance Manager, which is the component that handles the creation and deletion of services. The credentials and the end points are mandatory.

The service `create` and `delete` operations are executed asynchronously on the server side. This library also implements polling until an operation is finished, and the polling can be tuned by means of some optional properties. Since operations involve network activity, the package also caches the created instances. The cache options can also be provided by developers.

Table 73: Instance Manager Configuration Options

Property	Mandatory	Description	Default
user	✓	User name for authentication	-
password	✓	Password for "user"	-
post_managed_instance_url	✓	The REST end point used for creating a new service instance for a tenant	-
get_managed_instance_url	✓	The REST end point used for retrieving details of a specific tenant service instance	-
get_all_managed_instances_url	✓	The REST end point used for retrieving details of <b>all</b> instances (for <b>all</b> tenants)	-
delete_managed_instance_url	✓	The REST end point used for deleting a service instance	-
polling_interval_millis	-	The number of milliseconds to wait between requests in the polling phase	300ms
polling_timeout_seconds	-	The maximum amount of time (in seconds) that can be spent in polling.	120 secs
cache_max_items	-	The maximum size of the cache.	500
cache_item_expire_seconds	-	The maximum number of seconds after which a cache entry expires.	600 secs

### i Note

A managed service binding contains all of the mandatory properties mentioned above.

It is recommended to have a single instance-manager JavaScript object for each managed service that is bound to the application.

## Methods

The following table lists the methods available with the instance-manager API:

Table 74: Instance-Manager API Methods

Method	Description
create	Creates a service instance for the provided tenant. The method polls until the instance is successfully created and then invokes the callback. Reports error if an instance for this tenant already exists.

Method	Description
get	<p>Retrieves the corresponding service instance for the specified tenant either from the cache or from the server. A returned value of <code>null</code> indicates that no service instance exists for the specified tenant.</p> <p><b>⚠ Restriction</b></p> <p>The <code>get</code> method only polls if the service instance is in status <code>CREATION_IN_PROGRESS</code>. In all other cases it returns the service instance as it is on server. There is no guarantee that the <code>credentials</code> property on the <code>instance</code> object is included in the callback.</p>
delete	Removes the service instance for the provided tenant. The method polls until the instance is successfully deleted and then invokes the callback. An error is reported if no instance exists for the specified tenant.

### ➔ Tip

If the callback of a method is invoked with an error and the error is caused by an unexpected HTTP response code received from the server, then the error object will have a `statusCode` property with the same status code as the received HTTP response.

## Debug Logs

You can enable debug logs for the `@sap/instance-manager` package by adding `instance-manager` to the `<DEBUG>` environment variable.

## @sap/jobs-client

`@sap/jobs-client` is a small Node.js package to integrate jobs into your Node.js application. The package contains utilities that enable you to create REST calls conforming to the format expected by the SAP HANA XS advanced job scheduler service, which is used to register or unregister jobs, update job schedules, and display the job status.

This package works with job descriptor objects and uses properties defined according to the requirements of the corresponding service in Job Scheduler.

## Usage

### Sample Code

```
var jobsc = require('@sap/jobs-client');
var options = {
  host: 'localhost',
  port: 4242,
  timeout: 15000,
  user: 'username',
  password: 'password',
  baseURL: 'http://apphost:port/'};
};
```

```

var myJob = { /* according to job scheduler documentation */ };
var scheduler = new jobsc.Scheduler(options);
var scJob = { job: myJob };
scheduler.createJob(scJob, function (error, body) {
  if (error) {
    return console.log('Error registering new job %s', error);
  }
  // job was created successfully job.id = body._id;
});

```

## @sap/logging

Provides logging and tracing functions for Node.js applications; logs and traces are written using the standard SAP formats.

### Usage

#### Sample Code

```

var LoggingLib = require('@sap/logging');
var express = require('express');
var app = express();
var appContext = LoggingLib.createAppContext();
app.use(LoggingLib.expressMiddleware(appContext));
app.get('/demo', function (req, res) {
  var logger = req.loggingContext.getLogger('/Application/Network');
  var tracer = req.loggingContext.getTracer(__filename);
  logger.info('Retrieving demo greeting ...');
  tracer.info('Processing GET request to /demo');
  res.send('Hello World!');
});
app.listen(3000, function () {
  console.log('Server started');
});

```

You need to perform the following actions:

1. Create the application context by initializing the logging library with some application-wide options.
2. Specify the middleware to use to extract information about a specific request. The specified middleware is used to extract request-specific information and attach the `loggingContext` property to the request object.

#### Note

This middleware definition should come **before** the `request-user` is set.

3. Instantiate a logger and a tracer by using the `loggingContext` property of the request.
4. Log and trace whatever you need.

### Severity Levels

The following severity levels can be used in logging and tracing:

- Logging:  
“**info**” (default), “warning”, “error”, and “fatal”

- Tracing:  
“debug”, “path”, “info”, “warning”, “**error**” (default), and “fatal”

If you deploy your application to the XS advanced On-Premise run time, you can change the severity level of categories and script files dynamically at run time without needing to restart the deployed application. To set (or reset) the logging level at run time, use the `xs set-logging-level` command, as illustrated in the following example:

#### Sample Code

```
xs set-logging-level  
  
  xs set-logging-level <application-name> <category-or-path-to-file> <logging-level>
```

It is possible to set a maximum priority level for all loggers and tracers by using the environment variable `XS_APP_LOG_LEVEL`. Its value is a level from “debug” to “fatal”. The value from the environment variable (if valid) will be used instead of all already set levels. This is especially suitable for debugging purposes.

#### Note

If `XS_APP_LOG_LEVEL = none`, all logging and tracing is disabled. This setting is useful for automated testing.

## Using Loggers

The following example shows how to create a logger:

#### Sample Code

```
var logger = req.loggingContext.getLogger('/Application/Network');
```

The request context has the “`getLogger`” function that takes a single string argument; namely, the “category”. Categories define the names of functional areas in an application. It is recommended to prefix the name of your categories with “/Application”, for example, `/Application/Network`. The categories form a hierarchy with forward slash as the separator. The following code shows how to use the `getLevel()` command to retrieve the severity level (as a string) of a logger:

#### Sample Code

```
var level = logger.getLevel();
```

It is also possible to check whether an entry with a specific severity level will be logged with the current level configuration:

#### Sample Code

```
var willItBeLogged = logger.isEnabled('info');
```

The following logging entries are available: “info”, “warning”, “error”, and “fatal”.

## Sample Code

```
logger.info('Successful login of user %s - ', user, new Date());
logger.warning('Job did not complete successfully. An app admin must retrigger
the job.');
logger.error(new Error('Sorry! An error occurred'));
logger.fatal('A fatal error has occurred; the application has stopped!');
```

## Using Tracers

To obtain a tracer, use the `getTracer()` command, as illustrated in the following example:

## Sample Code

```
var tracer = req.loggingContext.getTracer(__filename);
```

In the same way as with loggers, you can also retrieve and check tracing instances, as illustrated in the following example:

## Sample Code

```
var level = tracer.getLevel();
var willItBeTraced = tracer.isEnabled('path');
// etc.
```

The tracing API provides the following methods to enable you to set up and use tracing in Node.js applications:

- Entering  
Record that a function has been entered in the program flow. You can pass all of the arguments of your function to the entering function so that they are traced.
- Exiting  
Typically used in combination with the “entering” method. If you provide an argument, it will be considered as the return value of your function.
- Throwing  
Used to trace when the code is about to throw an error; you can pass the error that is about to be thrown as an argument.
- Catching  
Used in catch blocks; you can pass the caught error as an argument.

## Sample Code

### Tracing Methods: Entering and Exiting

```
function myFunction(tracer, a, b ,c) {
  tracer.entering(a, b, c);
  var result = // some logic here ...
  tracer.exiting(result);
  return result;
}
```

## Sample Code

### Tracing Methods: Throwing and Catching

```
function func1(tracer) {
  var error = new Error('An error has occurred');
  tracer.throwing(error);
  throw err;
}
function func2(tracer) {
  try {
    func1(tracer);
  } catch (err) {
    tracer.catching(err);
    // logic for processing the error
  }
}
```

## @sap/textanalysis

The `@sap/textanalysis` package is a Node.js module on the XS advanced platform that supports text analysis. Text analysis performs linguistic analysis and entity extraction on unstructured text documents.

`@sap/textanalysis` is implemented as an interface to the SQL stored procedure `TA_ANALYZE` and provides the single API function: `analyze()`.

For more information about text analysis and the `TA_ANALYZE` stored procedure, see the *Text Analysis Developer Guide*.

### Usage

First, a connection must be established to the SAP HANA database. Then the client database object can be passed to the `analyze()` method along with the input parameters. The input parameters set the input variables for the `TA_ANALYZE` stored procedure.

## Sample Code

```
var ta = require('@sap/textanalysis');
var client;
async.series([
  function connect(callback) {
    client = hdb.createClient(options);
    client.connect(callback);
  },
  function analyze(callback) {
    var values = {
      DOCUMENT_TEXT: '<!DOCTYPE html><html><body><h1>My First Heading</h1><p>My first paragraph.</p></body></html>',
      LANGUAGE_CODE: 'EN',
      CONFIGURATION: 'EXTRACTION_CORE',
      RETURN_PLAINTEXT: 0
    };
    ta.analyze(values, client, function done(err, parameters, rows) {
      if (err) { return console.error('error', err); }
      callback();
    });
  },
],
```

```
        function end(callback) {
            client.end(callback);
        }, done
    );
}
```

## @sap/textmining

The `@sap/textmining` package is a Node.js module on the XS advanced platform that supports text mining. Text mining makes determinations about the content of unstructured text documents by examining the terms used within them.

`@sap/textmining` is implemented as an interface to the Text Mining SQL API in SAP HANA. The API functions and parameters follow the pattern of the Text Mining XS classic API:

- `categorizeKNN()`
- `getRelatedDocuments()`
- `getRelatedTerms()`
- `getRelevantDocuments()`
- `getRelevantTerms()`
- `getSuggestedTerms()`
- `initialize()`

For more information about the Text Mining SQL API, see the section about Advanced Data Processing in the *SAP HANA SQL and System Views Reference*; for information about `initialize()`, see the section *ALTER FULLTEXT INDEX*.

For more information about the Text Mining XS classic API, see the *SAP HANA Text Mining XS JavaScript API Reference*.

For more information about text mining in general, see the *SAP HANA Text Mining Developer Guide*.

## Usage

### Sample Code

```
var textmining = require('@sap/textmining');
var hdb = require('hdb');
var db = {
    "host": "HOST",
    "port": 3XX15,
    "user": "USERNAME",
    "password": "PASSWORD"
}
var client = hdb.createClient({
    host: db.host,
    port: db.port,
    user: db.user,
    password: db.password
});
var p = {
    inputTermText : "term",
    top : 10,
    threshold : 0.3
}
```

```

var config = {
  client : client,
  referenceTable : 'SCHEMA.TABLE',
  referenceColumn : 'COLUMN'
}
var tmd = new textmining(config);
client.connect(function(err){
  if(err)
  {
    client.end();
    throw err;
    return;
  }
  tmd.getSuggestedTerms(p, function(err, result) {
    if(err)
    {
      client.end();
      throw err;
      return;
    }
    console.log(result);
    client.end();
  });
});

```

## @sap/textbundle

This is a small Node.js package to help enable text for internationalization in Node.js. Based on the same concept as SAP UI5, this package works with UTF-8 encoded .properties files. The language default feature is also borrowed from SAP UI5 with the idea that both the UI and the server-side code use the same approach to text internationalization.

### Usage

Assuming that you have the following properties files in the folder ./test/properties:

- i18n\_en.properties

#### Code Syntax

```
greeting = Hello {0}, you are {1} years old.
```

- i18n\_de.properties

#### Code Syntax

```
greeting = Hallo {0}, Sie sind {1} Jahre alt.
```

#### Sample Code

```

var TextBundle = require('@sap/textbundle');
var bundle = new TextBundle({path: './test/properties/i18n', locale:
'en_EN' });
bundle.getText('greeting'); // returns 'Hello {0}, you are {1} years old.'

```

```
bundle.getText('greeting', ['Stefan']); // returns 'Hello Stefan, you are  
undefined years old.'  
bundle.getText('greeting', ['Stefan', '21']); // returns 'Hello Stefan, you  
are 21 years old.'  
// using DE locale  
bundle = new TextBundle({path: './test/properties/i18n', locale: 'de' } );  
bundle.getText('greeting', ['Stefan', '21']); // returns 'Hallo Stefan, Sie  
sind 21 Jahre alt.'
```

## Loading Text Bundles

Bundles can be loaded by providing the absolute path to the resource bundle or relative path. The example above shows how to load a bundle using a relative path. You can also load the bundle with an absolute path:

### Code Syntax

```
var TextBundle = require('@sap/textbundle');  
var path = require('path');  
var bundle = new TextBundle({path: path.resolve(__dirname, './test/properties/  
i18n'), locale: 'de' } );
```

The default file extension is `.properties`. If your file has a different extension, you must append it to the path you provide. The format of the file must be UTF-8, and the structure has to conform to the `.properties` file format.

### Code Syntax

```
var TextBundle = require('@sap/textbundle');  
var txtBundle = new TextBundle({path: './test/txt/i18n.txt'});
```

## @sap/xsenv

This is a small Node.js package to enable easy setup of (and access to) environment variables and services in XS advanced.

### Usage

The following example shows you how to look up the specific services bound to your application:

### Sample Code

```
var xsenv = require('@sap/xsenv');  
var services = xsenv.getServices({  
    hana: { tag: 'hdb' },  
    scheduler: function(service) { return service.label === 'jobs' }  
});  
var hanaCredentials = services.hana;  
var schedulerCredentials = services.scheduler;
```

The search criteria for the required services is specified in the query parameter to `getServices`. Each property of the query object specifies the query value for one service.

## Note

`getServices` can return only the requested service; it cannot return any of the standard SAP HANA XS advanced services (for example, `hana`, `uaa`, `jobs`) unless they are explicitly specified in the query parameter.

You can also pass a custom file from which you load a default service configuration. For example, the following example shows how to load defaults from the file `my-services.json`:

## Sample Code

```
var xsenv = require('@sap/xsenv');
var services = xsenv.getServices({
  hana: { tag: 'hana' },
  uaa: { tag: 'uaa' }
}, 'my-services.json');
```

## Service Queries

Both `getServices` and `filterCFServices` use the same service query values.

Table 75: Query Values for `getServices` and `filterCFServices`

Query Value	Description
{string}	Matches the service with the same service instance name ( <code>name</code> property). Same as { <code>name: '&lt;string&gt;' </code> }.
{object}	All properties of the given object should match corresponding service instance properties as they appear in <code>VCAP_SERVICES</code>
{function}	A function that takes a service object as argument and returns true only if it is considered a match

If an object is given as a query value, it may have the following properties:

Table 76: Object Properties for Query Values

Property	Description
<code>name</code>	The name of the service instance; the name you use to bind the service
<code>label</code>	Service name - the name shown by the command: "xs marketplace"
<code>tag</code>	Should match any of the service tags
<code>plan</code>	The name of the service instance plan specified in the command: "xs create-service"

## Loading SSL Certificates

If SSL is configured in the XS advanced on-premise run-time environment, it will provide one or more trusted CA certificates that applications can use to make SSL connections. If the certificates are available, the file

paths to these certificates are listed in the `<XS_CACERT_PATH>` environment variable separated by a path delimiter, for example, a colon (:) on Linux-based systems and a semi-colon (;) on Windows-based systems.

```
loadCertificates([certPath])
```

The `loadCertificates()` function enables you to load the certificates listed in the given path. If the `([certPath])` argument is not provided, `loadCertificates()` uses the `<XS_CACERT_PATH>` environment variable instead. If neither `([certPath])` nor `<XS_CACERT_PATH>` is defined, the `loadCertificates()` function returns "undefined".

### ➔ Tip

The `loadCertificates()` function is synchronous and returns an array even if only a single certificate is provided.

The following example shows how to load trusted CA certificates so they are used for all subsequent outgoing HTTPS connections:

#### Sample Code

```
var https = require('https');
var xsenv = require('@sap/xsenv');
https.globalAgent.options.ca = xsenv.loadCertificates();
```

The following example shows how to load SSL certificates for use in SAP HANA:

#### Sample Code

```
var hdb = require('hdb');
var xsenv = require('@sap/xsenv');
var client = hdb.createClient({
  host : 'hostname',
  port : 30015,
  ca : xsenv.loadCertificates(),
  ...
});
```

## @sap/xsjs

The package enables SAP HANA XS Classic applications to run in the compatibility layer in the JavaScript runtime for SAP HANA extended application services, advanced model (SAP HANA XS advanced).

### Usage

#### Sample Code

```
'use strict';
var xsenv = require('@sap/xsenv');
var xsjs = require('@sap/xsjs');
var port = process.env.PORT || 3000;
var options = xsenv.getServices({
```

```

        uaa: 'xsuaa',
        hana: 'hana-hdi',
        jobs: 'scheduler',
        mail: 'mail',
        secureStore: 'secureStore'
    });
xsjs(options).listen(port);
console.log('Node XS server listening on port %d', port);

```

The starting function takes an object that contains service credentials and application options.

Table 77: Property Options

Property	Default	Usage
rootDir	lib	Location of XS JavaScript files
rootDirs	-	Same as <code>rootDir</code> , but an array of directories can be provided; overrides <code>rootDir</code> if both are set
uaa	-	UAA configuration necessary to enable authentication by JSON Web Tokens (JWT) and business-user propagation to SAP HANA
hana	-	Object containing SAP HANA database connection parameters; used for database connectivity
secureStore	-	Object containing SAP HANA database connection parameters; used for secure store connectivity
jobs	-	Job scheduler connection parameters used to register jobs during application start up and later for updating job execution status when the job has finished
mail	-	Mail options, used by XS JavaScript <code>\$.net.Mail</code> API
anonymous	false	Enable anonymous access; that is, access without user credentials
formData	-	Special restrictions over form-data submitted to server
destinationProvider	-	Custom function, synchronous or asynchronous, to be used when <code>\$.net.http.readDestination</code> is called in XS JavaScript code.
ca	Certificates listed in the environment variable <code>&lt;XS_CACERT_PATH&gt;</code>	Trusted SSL certificates for any outgoing HTTPS connections, an array of loaded certificates
compression	true	By default text resources over 1,000 are compressed.
context	{ }	Extend the default context in <code>xsjs</code> scripts

SAP HANA XS advanced applications connect to SAP HANA with a fixed technical user provided by means of XS advanced services (environment variables). The actual (business) user of the application is retrieved from the JWT token and propagated to SAP HANA. The connection to the Job scheduler service is performed with a fixed technical user provided by the XS advanced service binding.

## hana

The `hana` object has the following properties:

Table 78: Property Options

Property	Mandatory	Usage
host	Yes	SAP HANA database host name
port	Yes	SAP HANA database port number
user	Yes	Technical user for database connections
password	Yes	Password for technical user specified in <code>user</code>
schema		If provided, sets current schema for database connections
connectWithLoggedUser		Possible values are <code>true</code> or <code>false</code> (default). If provided the database connection will be done made the SAML assertion contained in the JWT token of the logged user.  <b>⚠ Caution</b> This option is provided only for the SAP HANA Cockpit transition to SAP HANA XS advanced. In general, this option should be avoided.
sqlcc		Object containing all SQLCC configurations as properties with name after SQLCC name used in XS JavaScript code
ca		Trusted SSL certificates intended explicitly for use by SAP HANA connections, or an array of loaded certificates. If not provided, the certificate from the service binding is used. If no certificates are available, the SAP HANA connection cannot be encrypted.

The `SQLCC` property can be initialized from the bound services as shown in the following example:

### Sample Code

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

To use it in your XS JavaScript code:

### Sample Code

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

## secureStore

The `secureStore` object has the following properties:

Table 79: Property Options

Property	Mandatory	Usage
host	Yes	SAP HANA database host name
port	Yes	SAP HANA database port number
user	Yes	Technical user for database connections
password	Yes	Password for technical user specified in <code>user</code>
schema		If provided, sets current schema for database connections

## formData

The `formData` object has the following properties:

Table 80: Property Options

Property	Default	Usage
<code>maxFileSizeInBytes</code>	10485760	restricts the total size of all the uploaded files.

## mail

The `mail` object has the following properties:

Table 81: Property Options

Property	Mandatory	Usage
host	Yes	SMTP server host name
port	Yes	SMTP server port number
ignoreTLS	Yes	True or false (default). Defines if a STARTTLS command should be invoked if available by the mail server.
secure	Yes	True or false (default). Defines if the connection should be over TLS/SSL.
connectionTimeout		Connection timeout in ms. Default =60000.

Property	Mandatory	Usage
authMethod		Authentication method to use. Options are: "PLAIN", "LOGIN", or "CRAM-MD5"
auth		Authentication credentials, for example: { user: 'user', pass: 'pass' }; default is no authentication.

## destinationProvider

If your application requires different mechanisms for destination configuration, for example, dynamic configuration changes or dynamically adding new destinations to your application, you can provide your own function that retrieves these configurations from your storage. Support is provided for both synchronous and asynchronous destination provider function. The number of parameters your function has determines if the function is called synchronously or asynchronously. The signatures for both are as follows:

### Sample Code

```
function getDestinationSync(packagename, objectname, dtDescriptor) {
}
function getDestinationAsync(packagename, objectname, dtDescriptor, callback) {
}
```

Table 82: Parameter Options

Parameter	Description
packagename	The package of the destination supplied to <code>\$.net.http.readDestination</code>
objectname	The object name of the destination supplied to <code>\$.net.http.readDestination</code>
dtDescriptor	The object containing all properties contained in the corresponding HTTP destination configuration file ( <code>.xshttpdest</code> ), if available; otherwise "undefined"
callback	Provided only in the asynchronous case and should be called by your provider function to return the destination or report error

## @sap/xsjs-test

This client library provides the unit-test framework for the XS JavaScript compatibility layer (XS run time). To use the `@sap/xsjs-test` tools in your MTA project, you will need to perform the following steps:

- Declare a development dependency to `@sap/xsjs-test` in your XSJS application project (for example, in the `package.json`)
- Your tests are located in test folder (`test/`) parallel to `package.json` and `lib/`
- Configure a `xstest` script in your application's `package.json`
- Run the test tools, for example with the command `npm run xstest`

For the dependency to `@sap/xsjs-test`, you need to verify which version of `@sap/xsjs-test` is released and refer the released version. By establishing this developer dependency, you ensure that `@sap/xsjs-test` is installed only in the (local) development installation; it is not available for use in a productive installation. To avoid the necessity of any test dependencies, you can install the `@sap/xsjs-test` package on your development machine, for example, with the following command:

```
npm install -g @sap/xsjs-test
```

Normally the `xsjs` run-time files are located in the `xsjs/` folder, which means that the following paths are expected:

- `<>/xsjs/package.json`
- `<>/xsjs/lib/`
- `<>/xsjs/test/`  
Contains your test scripts; if you choose to put the test scripts in a different folder, a special configuration is required.

#### Test script

There is a normal binary script defined in bin folder. The normal way would be to define a script in the application `package.json`, as illustrated in the following example:

#### Sample Code

```
"scripts": {  
  "xstest" : "xstest"  
}
```

To start the XSJS unit tests from the command line, use the following command:

#### Sample Code

```
npm run xstest
```

## **@sap/xssec**

The client security library includes the XS advanced container security API for Node.js.

Authentication for node applications in XS advanced relies on a special usage of the OAuth 2.0 protocol, which is based on central authentication at the SAP HANA User Authentication & Authorization (UAA) server that then vouches for the authenticated user's identity by means of a so-called OAuth Access Token. The current implementation uses as access token a JSON web token (JWT), which is a signed text-based token formatted according to the JSON syntax.

In a typical deployment scenario, your node application will consist of several parts, which appear as separate application packages in your manifest file, for example:

- Database artifacts  
This application module (`<myAppName>/db/`) describes and defines the HANA database content
- Application logic

This application module (`<myAppName>/js/`) contains the application logic: code written in Node.js. This module can make use of this *XS Advanced Container Security API* for Node.js).

- UI client

This application module (`<myAppName>/web/`) is responsible for the UI layer; this module can make use of the application router functionality (defined in the file `xs-app.json`).

### Note

The application logic written Node.js and the application router should be bound to one and the same UAA service instance so that these two parts use the same OAuth client credentials.

To use the capabilities of the XS Advanced container security API, add the package “`@sap/xssec`” to the dependencies section of your application's `package.json` file. To enable tracing, set the environment variable `DEBUG` as follows: `DEBUG=xsssec:*`.

## Usage

To use the XS advanced container security API, it is necessary to provide a JWT token. The examples below rely on users and credentials that you should substitute with the ones in your context. The typical use case for calling this API lies from within a container when an HTTP request is received. The authorization header (with keyword “bearer”) already contains an access token; you can remove the prefix bearer and pass the remaining string to the API as “`access_token`”, as illustrated in the following example:

### Sample Code

```
xssec.createSecurityContext(access_token, xsenv.getServices({ uaa: 'uaa' }).uaa, function(error, securityContext) {
  if (error) {
    console.log('Security Context creation failed');
    return;
  }
  console.log('Security Context created successfully');
  var userInfo = securityContext.getUserInfo();
  console.log("User Info retrieved successfully");
});
```

The example above uses Node.js package `xsenv` to retrieve the configuration of the default services; the default services are read either from the environment variable `<VCAP_SERVICES>` or, if not set, from the default configuration file. However, only the required User Authentication & Authorization configuration (`uaa`) is passed to the method `createSecurityContext`.

## Container Security API

Table 83: Container Security API

API	Description
<code>createSecurityContext</code>	Creates the “security context” by validating the received access token against credentials put into the application's environment via the UAA service binding

API	Description
getUserInfo	<p>Returns a structure with the following properties:</p> <ul style="list-style-type: none"> <li>• <code>userInfo.logonName</code></li> <li>• <code>userInfo.firstName</code></li> <li>• <code>userInfo.lastName</code></li> <li>• <code>userInfo.email</code></li> </ul>
checkLocalScope	<p>Checks a scope that is published by the current application in the <code>xs-security.json</code> file</p>
checkScope	<p>Checks a scope that is published by an application</p>
getToken	<p>Returns a token that can be used to connect to the SAP HANA database. If the token that the security context has been instantiated with is a foreign token (meaning that the OAuth client contained in the token and the OAuth client of the current application do not match), “<code>null</code>” is returned instead of a token; the following attributes are available:</p> <ul style="list-style-type: none"> <li>• <code>namespace</code> Tokens can be used in different contexts, for example, to access the SAP HANA database, to access another XS advanced-based service such as the Job Scheduler, or even to access other applications or containers. To differentiate between these use cases, the namespace is used. In <code>lib/constants.js</code> we define supported name spaces (for example, <code>SYSTEM</code>).</li> <li>• <code>name</code> The name is used to differentiate between tokens in a given namespace, for example, “<code>HDB</code>” for the SAP HANA database or “<code>JOBSCEDULER</code>” for the job scheduler. These names are also defined in the file <code>lib/constants.js</code>.</li> </ul>
hasAttributes	<p>Returns “<code>true</code>” if the token contains any XS advanced user attributes; otherwise “<code>false</code>”.</p>
getAttribute	<p>Returns the attribute exactly as it is contained in the access token. If no attribute with the given name is contained in the access token, “<code>null</code>” is returned. If the token that the security context has been instantiated with is a foreign token (meaning that the OAuth client contained in the token and the OAuth client of the current application do not match), “<code>null</code>” is returned regardless of whether the requested attribute is contained in the token or not. The following attributes are available:</p> <ul style="list-style-type: none"> <li>• <code>name</code> The name of the attribute that is requested</li> </ul>

API	Description
isInForeignMode	Returns "true" if the token, that the security context has been instantiated with, is a foreign token that was not originally issued for the current application, otherwise "false".
getIdentityZone	Returns the identity zone that the access token has been issued for.

## @sap/xss-secure

This is a Node.js package that includes the cross-site scripting (XSS) security implementation taken from SAP UI5.

### Usage

#### Sample Code

```
var xsssecure = require('@sap/xss-secure');
```

The @sap/xss-secure package includes the following APIs:

- encodeCSS  
For Cascading Style sheets
- encodeHTML  
For HTML markup
- encodeJS  
For JavaScript code
- encodeURL  
For URL components, for example, parameter names or values
- encodeXML  
For XML code

Cross-site Scripting (XSS) is the name of a class of security vulnerabilities that can occur in Web applications. It summarizes all vulnerabilities that allow an attacker to inject HTML Markup or JavaScript into the affected Web application's front-end. XSS can occur whenever the application dynamically creates its HTML, JavaScript, or CSS (either in the back end or in the front-end part of the application in the user's Web browser), and values controlled by the attacker are used in this process. If this happens, the values are included in the generated HTML, JavaScript, or CSS without proper validation or encoding. As a result, an attacker is able to include arbitrary HTML, JavaScript, or CSS into an application's client front end; this code is, in turn, rendered by the victim's Web browser and, in this way, interpreted in the victim's current authentication context.

## @sap/e2e-trace

SAP Passports allow the identification of a specific request in an end-to-end scenario involving several components, which need to communicate with each other. The client sends a special header containing the

SAP Passport (a hex string with a special structure) to the first component. The client can send the SAP passport by means of a browser plug in or the front-end component of an SAPUI5 application. Whenever an application component receives an SAP Passport, this component should ensure that the unique identifiers of the SAP Passport are included in its log and trace files, update the SAP passport with component-specific data, and then forward it to the next system.

An application receives an SAP Passport in the `@sap/passport` header. The same header is used when the SAP Passport is forwarded to another system with the HTTP protocol. If the SAP passport is sent to SAP HANA, the SAP Passport should be set as the '`SAP_PASSPORT`' session variable of the database connection.

To load the `@sap/e2e-trace` package, use the following command:

#### Sample Code

```
var SAPPassport = require('@sap/e2e-trace').Passport;
```

The code in the following example shows how to create an instance of an SAP Passport:

#### Sample Code

```
function requestHandler(req, res) {
  var encodedPassport = req.headers[SAPPassport.HEADER_NAME];
  if (encodedPassport) {
    var passport = new SAPPassport(encodedPassport);
  }
}
```

The library provides the constant `SAPPassport.HEADER_NAME` for the '`@sap/passport`' header; the `passport` variable is an instance which you can use to read (or modify) the SAP Passport in your application component. The following code example shows how to read the unique identifiers of an SAP Passport:

#### Sample Code

```
var identifiers = passport.readUniqueIdentifiers();
```

The returned value (assigned to the `identifiers` variable) is an object that has the following properties: `transactionID`, `rootContextID`, `connectionID`, and `connectionCounter`. These fields of the SAP Passport must be present in the logs and traces of the application component. If you are using the `@sap/logging` library, refer to its documentation to check if the specific version of the library is capable of handling SAP Passports.

To see how to update the SAP Passport before forwarding it to the next application component, have a look at the following code example:

#### Sample Code

```
passport.update({
  previousComponent: 'my-application',
  connectionID: '00112233445566778899AABBCCDDEEFF',
  connectionCounter: 36
});
```

This method takes an object with the following **mandatory** properties:

Table 84: Mandatory SAP Passport Properties

Property	Value	Semantics
previousComponent	ASCII string (up to 32 characters)	The name of the current component which is about to make an outbound connection to another component.
connectionID	GUID (16-byte hex)	Every connection between the current component and the next component should have an ID which is passed as that property.
connectionCounter	Positive integer (up to 4 bytes)	The amount of time for the connection with the given connectionID is being reused.

### **Restriction**

The SAP HANA database has a limitation regarding the size of the SAP Passport. It is recommended to call the method `passport.compact()` before forwarding an SAP Passport to SAP HANA.

The following code example shows how to generate a hex string from the updated SAP Passport; the SAP Passport can be sent to other components in this format:

### **Sample Code**

```
var http = require('http');
var url = require('url');
var encodedPassport = passport.serialize();
var options = url.parse('http://my-host:1234/my/path');
options.headers = {};
options.headers[SAPPassport.HEADER_NAME] = encodedPassport;
var request = http.request(options);
request.on('error', function (err) { /* ... */ });
request.on('response', function (response) { /* ... */ });
request.end();
```

### **@sap/node-vsi**

The `@sap/node-vsi` package contains the Virus Scan Interface (VSI) binding for Node.js. The package also includes the native libraries required to run on Windows, Linux, and MacOS operating systems.

The following example shows how to use `@sap/node-vsi` to scan for the standard test file from the European Institute for Computer Antivirus Research at “[www.eicar.com](http://www.eicar.com)”. All anti-virus scanners must be able not only to find the file but also detect that it is a virus:

### **Sample Code**

```
var vsi = require('@sap/node-vsi');
var vsiProfile = vsi.vsiProfile;
var v = new vsiProfile("");
```

```
v.scanBytes("eicar.txt","X5O!P%@AP[4\\PZX54(P^)7CC)7}{$EICAR-STANDARD-ANTIVIRUS-TEST-FILE!$H+H*", 68);
console.log("\nResult of eicar scan is rc " + v.getLastErrorCode() + " (" +
v.getScanErrorName() + ") with error message: \n" + v.getLastErrorMessage() + "\n" );
```

## Getting Started

To start using the VSI @sap/node-vsi run the following command from your project directory:

```
$ var vsi = require('@sap/node-vsi');
```

## @sap/sds-deploy

The @sap/sds-deploy package includes the Node.js client for the deployment of application projects for the SAP HANA Streaming Analytics option. The @sap/sds-deploy library compiles the Continuous Computation Language (CCL) content of a Streaming Analytics project (consisting of a CCL file and an optional CCR file) and deploys the Streaming Analytics project to the Streaming Analytics service installed with the SAP HANA platform.

The Streaming Analytics application project is mapped to a Streaming Analytics module in the multi-target application (MTA). The Streaming Analytics application project consists of two configuration files:

- mySDS\_Project-Definition.ccl  
The Streaming Analytics project-definition file is written in Continuous Computation Language (CCL), which is the primary event-processing language of SAP HANA Streaming Analytics. You define a Streaming Analytics project using CCL, which is a language that is based on Structured Query Language (SQL), and adapted for stream processing.
- mySDS\_Project\_Configuration.ccr  
A project-configuration file (.ccr) is an XML document that is used to control specific run-time properties of a Streaming Analytics project, including the data stream's URI bindings as well as any adapter properties, parameter values, and advanced deployment options.

The following example of a simple .ccl file shows some code that runs a simple Tuple test:

### Sample Code

Example random\_tuple\_test2.ccl

```
CREATE INPUT WINDOW InputStream1 SCHEMA ( Column1 integer , Column2 integer ,
Column3 integer , Column4 string ) PRIMARY
KEY(Column1,Column2,Column3,Column4) KEEP 10000 ROWS;
ATTACH INPUT ADAPTER RandomTuples2 TYPE randomtuplegen_in TO InputStream1
PROPERTIES Rate =10000 , RowCount =0 , SecondDateFormat = '%Y-%m-%d %H:%M:
%S' , MsDateFormat = '%Y-%m-%d %H:%M:%S' ;
CREATE OUTPUT STREAM OutStream1 AS SELECT InputStream1.Column1
SimpleResultExpression2 , InputStream1.Column2 SimpleResultExpression1 FROM
InputStream1;
CREATE OUTPUT STREAM OutStream2 AS SELECT InputStream1.Column1
SimpleResultExpression2 , InputStream1.Column2 SimpleResultExpression1 FROM
InputStream1;
```

The following code example shows the contents of the corresponding .ccr file for the random\_tuple\_test2.ccl:

### Sample Code

Example random\_tuple\_test2.ccl

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration xmlns="http://www.sybase.com/esp/project_config/2010/08/">
  <Deployment>
    <Project ha="false">
      <Options>
        <Option name="time-granularity" value="5"/>
        <Option name="debug-level" value="7"/>
        <Option name="ignore-config-topology" value="false"/>
      </Options>
      <Instances>
        <Instance>
          <Failover enable="false"/>
          <Affinities/>
        </Instance>
      </Instances>
    </Project>
  </Deployment>
</Configuration>
```

## @sap/audit-logging

Provides audit logging functionality for Node.js applications.

Audit logging concerns writing entries in a specific format to a log storage. The entries written to the audit log concern any events of significant importance, for example, security events which may impact the confidentiality, the integrity, or the availability of a system. Access to sensitive personal data (both reading and altering) such as bank accounts, political opinion, health status is also an event that is written to the audit log. Audit logs are not the same as ordinary log files. Ordinary log files are used by the system administrator who need to keep track of the state of a system; audit logs are read by an auditor. Audit logs are subject to legal requirements, which in some countries are stricter than in others. In general, the events that are supposed to be recorded in the audit log can be grouped into the following categories:

- Changes to system configurations (which may have significant effect on the system itself)
- Access to sensitive data (related to data privacy)
- General security events (for example, starting or stopping a system, failed authorization checks, etc.)

### Tip

For more detailed information about the audit-log NPM package, see the README.md file included in the @sap/audit-logging package.

## @sap/ui-annotations

Enables the use of UI annotations in CDS; currently, this feature is supported by the XS advanced Java framework in combination with OData v4. The following UI annotations are currently supported:

- Hidden
- Identification
- LineItem
- Mandatory
- Masked
- MultiLineText
- Optional
- ReadOnly
- StatusInfo
- Chart
- Contact
- FieldGroup
- DataPoint
- SelectionFields
- HeaderInfo

To enable UI annotations, add the entry “`using sap.common::UI;`” to the start of the CDS documents which contain elements that you want to publish with OData v4.

### Sample Code

Enable UI Annotations in an OData v4 Service

```
namespace acme.com;
using sap.common::UI;
@OData.publish : true
context ContextA {
    MyEntity1 {
        key ID : Integer;
    };
    MyEntity2 {
        key ID : Integer;
        type Address1
    };
};
```

To use the features provided in the XS advanced NPM package `@sap/ui-annotations`, it is necessary to ensure that the `package.json` file in the XS advanced application's HDB module specifies the dependency to `@sap/ui-annotations`, as illustrated in the following example:

### Sample Code

Using UI Annotations

```
{
  "name": "deploy",
  "dependencies": {
```

```
    "@sap/hdi-deploy": "2.2.0"
    "@sap/ui-annotations": "1.0.0-1"
  },
  "scripts":
  {
    "start": "node node_modules/@sap/hdi-deploy/"
  }
}
```

## @sap/xs-msg

Provides messaging capabilities with a message broker. This package supports the RabbitMQ message broker with the following protocols:

- amqp v091
- mqtt v311

The following example shows how to create a new client, set it up to consume incoming messages, and connect the client:

### Sample Code

```
const client = new Client(options);
client.istream('in')
  .on('data', (message) => {
    console.log('received message ' + JSON.stringify(message));
  });
client.connect();
client
  .on('connected', () => {
    console.log('connected');
  })
  .on('error', (err) => {
    console.log(err);
  });
}
```

## @sap/xb-msg-amqp-v091

This package provides a client implementation for AMQP v0.9.1.

A single client instance represents one connection to the broker. Either TLS or NET socket is used depending on the options defined for the client. The API works completely asynchronous based on callbacks, often providing also “done” (resolve) and “failed” (reject) callbacks. This means it would be simple to use Promise objects in the application even if the client library so far does not use it. AMQP v0.9.1 defines classes and methods (like remote procedure calls). Unfortunately, some of them do not allow a key-based mapping of responses to requests. Hence, for those methods the client has to wait for the response before a second request can be sent. The client encapsulates this and behaves always asynchronously for the caller.

### Prerequisites

A message broker must be available, for example, RabbitMQ.

## Usage

The following example shows how to create a new client instance:

### Sample Code

```
const options = {
  tls: {
    host: 'localhost',
    port: 5671,
    ca: [
      fs.readFileSync('../.../truststore/cacert.pem'),
      fs.readFileSync('../.../truststore/cert.pem')
    ]
  },
  net: {
    host: 'localhost',
    port: 5672,
  },
  sasl: {
    user: 'guest',
    password: 'guest'
  },
  amqp: {
    vhost: '/',
  }
};
const client = new AMQP.Client(options);
```

### Note

Connection-callback options (for example, host, port, path, socket, etc.) must be provided for either "tls" or "net"; "tls" will be preferred.

## @sap/xb-msg-mqtt-v311

This package provides a client implementation for MQTT v3.1.1.

A single client instance represents one connection to the broker. Either TLS or NET socket is used depending on defined client connection options. The API works completely asynchronously based on callbacks, often providing also "done" (resolve) and "failed" (reject) callbacks. This means it would be simple to use Promise objects in the application even if the client library does not yet use it.

### Prerequisites

A message broker must be available, for example, RabbitMQ.

## Usage

The following example shows how to create a new client instance:

### Sample Code

```
const options = {
  net: {
```

```

        host: 'host.acme.com',
        port: 1883
    },
    mqtt: {
        user: 'admin',
        password: 'admin',
        clientId: 'myClient',
        keepAlive: 10000
    }
};
const client = new MQTT.Client(options);

```

### Note

Connection-callback options (for example, host, port, path, socket, etc.) must be provided for either “tls” or “net”; “tls” will be preferred.

## @sap/site-entry

This component is the Web entry for Fiori Launchpad portal sites. It includes an `approuter` component that serves as its Web server for requests from client-side resources and back-end calls. This component uses the portal-service that exists in the XS advanced environment.

### Usage

Create the folder structure shown in the following example:

#### Sample Code

Application Sources Folder Structure

```

<site-entry-folder-name>/
| - package.json
| - xs-app.json

```

The following code sample shows the contents of the application package descriptor (`package.json`):

#### Sample Code

`package.json`

```

{
  "engines": {
    "node": ">=4.0.0"
  },
  "name": "site-entry",
  "dependencies": {
    "@sap/site-entry": "^1.9.1"
  },
  "scripts": {
    "start": "node --harmony node_modules/@sap/site-entry/server.js"
  }
}

```

In the application descriptor `xs-app.json`, add the route to your SAPUI5 application's static resources:

### Sample Code

#### Application Descriptor (`xs-app.json`)

```
{  
  "source": "^/app1/sap/demo/(.*)",  
  "target": "$1",  
  "localDir": "app1/sap/demo/",  
  "cacheControl": "public, max-age=31536000,must-revalidate"  
}
```

The `mtad.yaml` file must contain the following information:

- The site-entry module needs to be bound to the `portal-service` service; it should require a resource of the following type: `com.sap.portal.site-host`.
- The path to the SAPUI5 that is deployed on the XS advanced model host
- The path to the XS User Account and Authentication (UAA) system

### Sample Code

#### Application Deployment Descriptor (`mtad.yaml`)

```
...  
modules:  
  - name: site-entry-<unique site id>  
    parameters:  
      memory: 64M  
    requires:  
      - name: sap-portal-services-host-<unique site id>  
      - name: myuua  
      - name: sapui5-provider  
        properties:  
          sapui5url: ~{url}  
    type: javascript.nodejs  
  - name: site-content-<unique site id>  
    parameters:  
      memory: 32M  
    requires:  
      - name: sap-portal-services-client-<unique site id>  
      - name: myuua  
    type: com.sap.portal.site-content  
resources:  
  - name: sap-portal-services-host-<unique site id>  
    parameters:  
      config:  
        siteId: <unique site id>  
    type: com.sap.portal.site-host  
  - name: myuua  
    parameters:  
      service-plan: space  
    type: com.sap.xs.uaa  
  - name: sapui5-provider  
    parameters:  
      provider-id: com.sap.ui5.dist.sapui5-dist-xsa.XSAC_UI5_FESV3:sapui5_fesv3  
      version: '>=1.42.0'  
      provider-nid: mta  
    type: configuration  
  - name: sap-portal-services-client-<unique site id>  
    parameters:  
      config:  
        siteId: <unique site id>
```

```
type: com.sap.portal.site-content
```

## @sap/site-content-deployer

This component is combined with the `@sap/site-entry` node module, which in turn acts as the Web-entry server for the Fiori Launchpad portal site. This component is used to deploy the Fiori Launchpad portal site configuration (for example, the configuration of tiles, groups and catalogs) into the XS advanced environment. This component uses the portal-service that exists in the XS advanced environment.

### Usage

Create the folder structure shown in the following code sample and copy the `manifest.json` file of your SAPUI5 application to your app folder, `MySAPUI5App`, as illustrated in the following example:

#### Sample Code

##### Application Sources Folder Structure

```
<site-content-deployer-folder-name>/
|- package.json
|- applications/
|   \- MySAPUI5App/
|       |- manifest.json
|       \- i18n/
\- src/
    |- site-content.json
    \- i18n/
```

The following code is an example of a `package.json` file:

#### Sample Code

##### Application Package Descriptor (`package.json`)

```
{
  "engines": {
    "node": "4.x"
  },
  "name": "site-content-deployer",
  "version": "1.9.1",
  "description": "portal deployer package",
  "dependencies": {
    "@sap/site-content-deployer": "^1.9.1"
  },
  "scripts": {
    "start": "node --harmony node_modules/@sap/site-content-deployer/deploy.js"
  }
}
```

The `site-content-deployer` module must be bound to the `portal-service` service. For example, in the `mtad.yaml` file, the `site-content-deployer` should require a resource of type “`com.sap.portal.site-content`”, as illustrated in the following example:

### Sample Code

Application Deployment Descriptor (`mtad.yaml`)

```
...
modules:
- name: site-content-deployer-<unique site id>
  type: javascript.nodejs
  requires:
    - name: sap-portal-services-client-<unique site id>
...
resources:
- name: sap-portal-services-client-<unique site id>
  type: com.sap.portal.site-content
parameters:
  config:
    siteId: <unique site id>
```

For more information about developing Fiori Launchpad modules, see *Related Information*.

## @sap/site-app-server

This component is a Web server for static resources used by applications that run in a Fiori Launchpad portal site. This component is coupled with the `@sap/site-entry` node module.

## Related Information

[Download and Consume SAP Node.js Packages \[page 516\]](#)

[The SAP NPM Registry \[page 560\]](#)

[Configuring the HDI Deployer \[page 188\]](#)

[Maintaining XS Advanced Application Routes and Destinations \[page 716\]](#)

[Developing SAP Fiori Launchpad Modules \[page 1080\]](#)

## 7.1.2.2 The SAP NPM Registry

SAP maintains a public NPM registry providing Node.js modules for use by application developers.

Whether you are using command-line tools or graphical tools to build your Node.js application, you can configure the build to use the packages that are available in registries which make Node Package Modules available for public consumption, for example, "`registry.npmjs.org`". SAP maintains a public NPM

registry that contains a selection of SAP-specific features and functions, which you can configure your build tools to consider at build time, as described below:

- [NPM Registry Configuration for Command-Line and Build Tools \[page 561\]](#)
- [NPM Registry Configuration for SAP Web IDE for SAP HANA \[page 561\]](#)

## NPM Registry Configuration for Command-Line and Build Tools

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

### Sample Code

NPM Registry Configuration in the `.npmrc` File

```
...  
@sap:registry=https://npm.sap.com  
...
```

You can also use the `npm` command to configure the target registry for the current session:

```
npm config set @sap:registry https://npm.sap.com  
npm install @sap/<node_package>
```

## NPM Registry Configuration for SAP Web IDE for SAP HANA Build Tools

If you are using the build tools included in SAP Web IDE for SAP HANA to build your Node.js applications for XS advanced, you can configure SAP Web IDE for SAP HANA to use one or more specific NPM registries to resolve the dependencies between the resources specified in the development and deployment descriptors.

To specify a specific NPM registry that will be used for the resolution of dependency requirements when developing and building applications with SAP Web IDE for SAP HANA, add the "UPSTREAM\_LINK" and (or) "SAPUPSTREAM\_LINK" properties to the `di-local-npm-registry` module in the MTA **extension** descriptor (for example, `sapwebide.mtaext`) that is used when deploying SAP Web IDE, as illustrated in the following example:

### Note

If a suitable MTA extension descriptor does not already exist for the deployment of SAP Web IDE, you must create one and add the suggested configuration details for the NPM registries.

### Sample Code

MTA Extension Descriptor: NPM Registry Configuration for SAP Web IDE Deployment

```
...
```

```
modules:  
  - name: di-local-npm-registry  
    properties:  
      UPSTREAM_LINK: "http://registry.npmjs.org"  
      SAPUPSTREAM_LINK: "https://npm.sap.com"  
  ...
```

## Related Information

[Download and Consume SAP Node.js Packages \[page 516\]](#)

[Standard Node.js Packages for XS Advanced \[page 519\]](#)

### 7.1.3 Download and Install JavaScript Data Services

Node.js Data Services (`node-cds`) provide a native JavaScript client for using Core Data Services functionality in the JavaScript (`node.js`) run time container on the XS advanced platform.

## Context

JavaScript Data Services (`node-cds` API) include a native JavaScript client and query builder for Core Data Services (CDS) for node.js on SAP HANA XS Advanced Model; `node-cds` enables programs to consume SAP-HANA-based records as native JavaScript objects. The `node-cds` library supports basic CRUD operations and complex queries on all data types (native SAP HANA and defined in CDS), and covers associations, transactions, and lazy navigation.

To use the `node-cds` API functions in your `node.js` programs, perform the following steps.

## Procedure

1. Add a dependency to the `node-cds` package in the `package.json` file of your Node.js application.

### Sample Code

```
{  
  "dependencies": {  
    "cds": "git://github.acme.corp/xs2/node-cds.git", ...  
  }, [...]
```

2. Import the `node-cds` module into your into your Node.js application.

Use the `require()` statement, as illustrated in the following example:

#### Sample Code

```
var cds = require("cds");
```

## Related Information

[Node.js Data Services \[page 563\]](#)

[Getting Started with Node.js Data Services \[page 564\]](#)

### 7.1.3.1 Node.js Data Services

Node.js Data Services (`node-cds`) are a native JavaScript client for using Core Data Services functionality in the node.js container on the SAP HANA XS Advanced model platform.

Core Data Services (CDS) are a cross-platform set of concepts and tools to define semantically rich data models for SAP HANA applications. A central part of CDS is the object-relational mapping that links semantically meaningful entities to their technical representation as records in the SAP HANA database.

The `node-cds` client frees the application developer from using the low-level `node-hdb` interface for interaction with SAP HANA data by embedding records as native JavaScript objects in the application logic. The `node-cds` module understands CDS meta-data such as associations and maintains consistency when manipulating entity instances as JavaScript objects. Advanced queries are constructed using a fluent query API that follows the CDS Query Language specification as closely as possible.

The `node-cds` library uses native SAP HANA and CDS functionality where available but provides its own implementation where the core implementation is still incomplete. `Node-cds` can also be used without CDS, for example, for legacy application data, while still benefiting from a similar level of convenience. The `node-cds` library supports basic CRUD operations and complex queries on all native SAP HANA and CDS-defined data types, and covers associations, transactions, and lazy navigation, too.

## Related Information

[Download and Install JavaScript Data Services \[page 562\]](#)

## 7.1.3.2 Getting Started with Node.js Data Services

A quick tour of basic node-cds API functionality.

To give you an idea of what you can do with the node-cds API, this short tour of the basic capabilities of the node-cds library covers the following areas:

- [Defining and Generating the JavaScript Objects \[page 564\]](#)
- [Connecting to the Database \[page 565\]](#)
- [Retrieving and Updating Entity Instances \[page 566\]](#)
- [Enabling Lazy Navigation \[page 567\]](#)
- [Using Advanced Queries \[page 568\]](#)
- [Extending node-cds Beyond CDS \[page 570\]](#)

### Defining and Generating the JavaScript Objects

The following example of a CDS document show the underlying tables for a bulletin board application that allows users to post short text messages and to reply to other users' posts. In the CDS data model shown here, we define two entities; namely, "user" and "post", as illustrated in the following example code:

#### Sample Code

##### CDS Document

```
namespace sap.hana.xs2.cds.examples.data;
context bboard {
    entity user {
        key uid: Integer not null;
        Name: String(63) not null;
        Email: String(63);
        Created: UTCDateTime;
    };
    type text {
        Text: String(255);
        Lang: String(2);
    };
    entity post {
        key pid: Integer not null;
        Title: String(63) not null;
        Text: text;
        Author: association [1] to user;
        Parent: association [0..1] to post;
        Rating: Integer;
        Created: UTCDateTime;
    };
}
```

If you use the node-cds module to import this CDS definition into your Node.js application code, a JavaScript object is generated for each of the entities defined (for example, `user` and `post`). These entity objects are then used to create and retrieve entity instances, which are native JavaScript objects that have all the properties of the underlying CDS data model.

#### Tip

Required type and key information will be inferred automatically.

## Sample Code

Node.js Application Importing a CDS Model

```
cds.$importEntities([
  { $entity: "sap.hana.xs2.cds.examples.data::bboard.post" },
  { $entity: "sap.hana.xs2.cds.examples.data::bboard.user" }
], main);
function main(error, entities) {
  if (error) {
    console.error(error);
  } else {
    var Post = entities["sap.hana.xs2.cds.examples.data::bboard.post"];
    var User = entities["sap.hana.xs2.cds.examples.data::bboard.user"];
    // ...
  }
}
```

## Connecting to the Database

Before you can work with entity instances, you need to get a database connection from a connection pool and open a transaction context:

## Sample Code

```
cds.$getTransaction(function (err, tx) {
  // ... transaction context
});
```

When the transaction is complete, close and release the connection by calling the `$close` method:

## Sample Code

```
tx.$close();
```

Alternatively, you can reuse an existing database connection, for example, a connection provided by some surrounding framework such as “express”:

### Note

The operation is still asynchronous, as the status of the database connection may be “disconnected”.

## Sample Code

```
cds.$getTransaction(dbconn, function (err, tx) {
  // ... transaction context ...
});
```

## Retrieving and Updating Entity Instances

The `$get` method of a transaction object retrieves an entity instance asynchronously by its key, as illustrated in the following example:

### Sample Code

```
tx.$get(Post, { pid: 101 }, function(error, instance) {
    // ...
});
```

The `instance` argument of the `$get` callback method receives a native JavaScript object whose JSON representation looks something like the following example:

### Sample Code

```
{
  pid: 101,
  Title: "First Post!",
  Rating: 1,
  Text: {
    Lang: "EN",
    Text: "This is my first post."
  },
  Author: {
    uid: 1,
    Name: "Kojo",
    Email: "kojo.mensah@acme.com",
    Created: "2014-06-26T11:44:00.000Z"
  },
  Created: "2014-06-26T11:44:00.000Z"
}
```

### Note

Associated instances such as the `Author` instance are retrieved automatically by default and become part of the `Post` instance.

Modifications to an entity instance remain local to the transaction context of the current Node.js application. To transfer any changes to the database, call the `$save` method of the modified instance:

### Sample Code

Update both a Post and a User

```
// update post and user
post.Rating++;
post.Author.Email = "kojo.mensah@acmehana.com";
tx.$save(post, function(error, instance) { /* ... */ });
```

To create a new instance, invoke `$save` with the entity type and the skeleton value for the new instance to create:

**i Note**

Unless you are using SAP HANA's sequence mechanism to generate new values automatically, it is necessary to supply your own key values.

 **Sample Code**

```
// create new post
tx.$save({
  $entity: Post,
  pid: 102,
  Title: "New Post",
  Text: { Text: "Hello BBoard!", Text: "EN" },
  Author: post.Author,
  Rating: 1,
  Created: new Date()
}, function (error, newPost) { ... });
```

## Enabling Lazy Navigation

The `node-cds` library enables you to perform a “lazy” loading of associated entity instances; lazy loading is normally used to help reduce the amount of data transferred to the application server.

 **Sample Code**

```
cds.$importEntity({
  $entity: "sap.hana.xs2.cds.examples.data::bboard.post",
  $fields: { Author: { $association: { $lazy: true } } }
}, function(error, entities) { ... });
```

If lazy loading is enabled, `node-cds` only retrieves those instances that the application actually needs to process. To retrieve the targets of a lazy association, we use the `$load` function, as illustrated in the following example:

 **Sample Code**

```
tx.$get(Post, { pid: 101 }, function (error, post) { // retrieve Post only
  if (post.Rating > 0) {
    post.Author.$load(function (error, author) { // now retrieve
      associated User
        var sender = author.Email;
        // ...
      });
    }
});
```

## ➔ Tip

Although lazy loading can cause additional round trips to the database, it can also reduce the total amount of data transferred if lazy associations are followed only infrequently.

## Using Advanced Queries

For complex ad hoc queries `node-cds` provides a query builder based on the CDS Query Language. The `$query` method of the entity returns a query object that enables you to build a complex query step-by-step:

### Sample Code

The `node-cds` `$query` Method

```
var qPosts = Post.$query(tx);
```

Queries can be refined with (or derived from) existing queries by using relational operators such as `$project`, `$where`, or `$matching`. The `$project` operator method specifies the fields which should be returned by the query:

### Sample Code

```
var qTitleAndAuthor = Post.$query().$project({
  Author: {
    Name: true,
    Email: true
  },
  Title: true
});
```

The argument to `$project` is a JavaScript object where the fields to be included are marked “`true`”. Without projection, all fields defined in the entity are returned by default. The `$where` operator method filters the result of the query by some conditional expression:

### Sample Code

```
var qImportantTitleAndAuthor = qTitleAndAuthor.$where(Post.Author.Name.
  $eq("Kojo"));
```

The `$matching` operator method provides a more concise, JSON-based syntax with a subset of the full expressions language supported by `$where`:

### Sample Code

```
qImportantTitleAndAuthor = qImportantTitleAndAuthor.$matching({ Rating: { $gt:
  2 } });
```

None of the query building steps shown above involves any kind of communication with the index server. To execute the query and retrieve the result, invoke the `$execute` method, as illustrated in the following example:

#### Sample Code

```
qImportantTitleAndAuthor.$execute(function (error, results) {  
    for (var r in results) { ... }  
});
```

`node-cds` creates the SQL query in the background, automatically inserting JOINs based on all the projection and selection operations defined in the query. The above sequence of statements results in the SQL query illustrated in the following example:

#### Sample Code

```
SELECT "t0.Author"."Name" AS "t0.Author.Name",  
       "t0.Author"."Email" AS "t0.Author.Email",  
       "t0"."Title" AS "t0.Title"  
  FROM "sap.hana.xs2.cds.examples.data::bboard.post" "t0"  
 JOIN "sap.hana.xs2.cds.examples.data::bboard.user" "t0.Author"  
    ON "t0"."Author.uid"="t0.Author.uid"  
 WHERE ("t0"."Rating" > 2) AND ("t0.Author"."Name" = 'Kojo')
```

The following example shows the probable query result:

#### Sample Code

```
[{  
    Author: {  
        Name: "Kojo",  
        Email: "kojo.mensah@acme.com"  
    },  
    Title: "Fail"  
}]
```

#### Note

The result returned by an advanced query is a plain value, not a managed entity instance. The main difference between a plain value and an entity instance is the handling of updates and associated entities.

You may freely convert between plain values and managed instances, however, or you could use the `$find` method (instead of `$matching`):

#### Sample Code

```
tx.$find(Post, {  
    Rating: { $ge: 1, $lt: 5 },  
    Parent: { $null: false }  
}, function (error, posts) { /* ... */ });
```

## Extending node-cds Beyond CDS

The node-cds library enables you to import existing CDS entities, but it can also transform entity definitions and derive new entities. This extensibility allows you to work with legacy SQL data models, for example, by enriching the data model to make implicit associations between plain database tables **explicit**:

### Sample Code

```
cds.$importEntities([
  $schema: "SCHEMA",
  $table: "BBOARD_POSTS",
  $name: "bbposts",
  $fields: {
    Author: {
      $association: { $entity:
        "sap.hana.xs2.cds.examples.data::bboard.user" },
      uid: { $column: "POSTED_BY" }
    }
  }
}], function (error, entities) { /* ... */});
```

### Note

node-cds imports every table column as an entity property. However, since plain SQL lacks the metadata for the associations, you must supply the information about the associations manually.

## Related Information

[Download and Install JavaScript Data Services \[page 562\]](#)

[Node.js Data Services \[page 563\]](#)

### 7.1.3.3 Node.js Data Services in XSJS Compatibility Mode

An extension of Node.js Data Services that provides compatibility with XS Classic Data Services (XSDS) library.

If you have code to run in the XS JavaScript (XSJS) compatibility mode, you can use an extension to the XS Advanced Node.js data services that is compatible with the basic XS Classic data services library (XSDS).

### ⚠ Restriction

This extension is intended to be used only within the XSJS Compatibility Layer run-time container.

To initialize the XS Advanced Node.js extension from an XSJS file, it needs to be invoked as follows:

## Sample Code

```
var conn = $.db.getConnection();
var cds = $.require("xsjs").xsjs(conn);
```

The connection to the database is managed by the connection provided in the argument to the XSJS call; the connection object can be modified through the `cds.Transaction` object, for example, you can close the connection with the method `cds.Transaction.$close()`.

Similarly to the way it is done in standard (not compatibility-mode) node-cds, you can import entities using the syntax and the synchronous style of XSDS, as illustrated in the following example:

## Sample Code

### The `cds.$importEntity` Method

```
var Post = cds.$importEntity("sap.hana.xs2.cds.examples.data", "bboard.post",
  { Comments:
    { $association:
      { $entity:
        "sap.hana.xs2.cds.examples.data::bboard.user",
        $viaBacklink: "Post"
      }
    }
  });
});
```

You can retrieve entities either in **managed** or in **unmanaged** mode, using the operations already known from standard (not compatibility-mode) node-cds, but in the synchronous approach of XSJS. For example, the `$get` method of an entity will retrieve an entity instance by its key:

## Sample Code

```
var post = Post.$get({ pid: 101 });
```

To create a new instance, invoke the “new” operator and persist the resulting object:

## Sample Code

```
// create new post
var newpost = new Post({ pid: 102,
  Title: "New Post",
  Text: { Text: "Hello BBoard!", Text: "EN" },
  Author: post.Author,
  Rating: 1,
  Created: new Date() });
newpost.$save();
```

For complex ad hoc queries, you can use the same query builder as the one available in standard (not compatibility-mode) node-cds. The `$query` method of the entity constructor returns a query object that enables you to build a complex query step by step:

### Sample Code

#### The `$query` Method

```
var results = Post.$query().$project({
  Author: {
    Name: true,
    Email: true
  },
  Title: true
}).$execute();
```

## Related Information

[Download and Install JavaScript Data Services \[page 562\]](#)

[Getting Started with Node.js Data Services \[page 564\]](#)

## 7.1.4 Tutorial: Setting up your JavaScript Application in XS Advanced

Tutorials that show you how to set up JavaScript applications for XS advanced.

This section includes a selection of tutorials that show you how to set up various basic but important aspects of the application that you plan to deploy on the XS advanced JavaScript run time.

### Note

The tutorials in this section focus on tools provided with the XS advanced command-line client.

## Prerequisites

Before you start the tutorials in this section, bear in mind the following prerequisites:

- SAP HANA XS advanced model must be installed on your SAP HANA system and you can log in to it using the XS advanced command-line client. For more information, see *Related Links* below.
- Node.js is installed locally. If you use a proxy to connect to the Internet, you might need to configure it. For more information about proxy configuration settings for NPM, see *npm Documentation* in the *Related Links* below.
- SAP Node.js packages

The archive (`xs_javascript-<version>-bundle.tar.gz`) is available on the SAP HANA media and the SAP Marketplace (SMP); it includes a selection of Node.js packages developed by SAP for use with the Node.js applications running in the XS advanced run time.

➔ **Tip**

Alternatively, you can configure NPM to resolve SAP-related NPM dependencies using the SAP public NPM Registry (at [npm.sap.com](https://npm.sap.com)).

## Related Information

[Standard Node.js Packages for XS Advanced \[page 519\]](#)

[The SAP NPM Registry \[page 560\]](#)

[Deploy a Node.js Hello-World Application using the XS CLI \[page 84\]](#)

[Working with the XS Advanced Command-Line Client \[page 78\]](#)

### 7.1.4.1 Create a Node.js Application

Create a sample Node.js application.

#### Procedure

1. Create a new directory named `node-tutorial`.
2. Create a `manifest.yml` file in the `node-tutorial` directory with the following content:

```
---  
applications:  
- name: myapp  
  path: myapp
```

This descriptor is used to describe applications and where their sources are located.

3. Create a new directory inside `node-tutorial` named `myapp`.
4. Inside the `myapp` directory, create a new file called `package.json` with the following content:

```
{  
  "name": "myapp",  
  "description": "My App",  
  "version": "0.0.1",  
  "dependencies": {  
    "express": "^4.13.0"  
  },  
  "scripts": {  
    "start": "node start.js"  
  }  
}
```

- Inside the `myapp` directory, create another file called `start.js` with the following content:

```
var express = require('express');
var app = express();
app.get('/', function (req, res) {
  res.send('Hello World!');
});
var port = process.env.PORT || 3000;
app.listen(port, function () {
  console.log('myapp listening on port ' + port);
});
```

- To install dependencies execute the `npm install` command in the `myapp` directory.
- Deploy the application on XS advanced.

Execute the following command in the `node-tutorial` directory.

```
xs push
```

You can check the state and URL of your application by using the `xs apps` command.

#### Output Code

```
> xs apps
Getting apps in org <orgname> / space <spacename> as XSA_ADMIN...
Found apps:
name      requested state    instances   memory      disk       urls
-----
myapp    STARTED           1/1        1.00 GB  <unlimited>
<URL>
```

- Open a browser window with your application URL.

You should see the `Hello World!` message.

### 7.1.4.2 Consume XS Advanced Services

XS advanced provides services, which can be used by applications.

#### Prerequisites

You have configured your command-line interface to use the SAP public NPM registry. For more information, see *Related Links* below.

## Context

XS advanced provides a convenient package (`@sap/xsenv`) for Node.js applications, which can be used to read bound services. To use the `@sap/xsenv` package, you have to add it to your application's dependencies, which are specified in a corresponding `package.json` file.

## Procedure

1. Navigate to the `myapp` directory.
2. Execute `npm install @sap/xsenv@^1.2.7 --save` to install the `@sap/xsenv` module and add it as a dependency in the `package.json` file.

After the installation of the package is complete, the dependencies section in the `package.json` file should have the following content:

```
"dependencies": {  
    "express": "^4.13.0",  
    "@sap/xsenv": "^1.2.7"  
},
```

3. Update the `start.js` file with the content shown in the following example:

```
var express = require('express');  
var app = express();  
var xsenv = require('@sap/xsenv');  
var services = xsenv.getServices({ hana:'myhana' });  
app.get('/', function (req, res) {  
    res.send('Using HANA ' + services.hana.host + ':' + services.hana.port);  
});  
var port = process.env.PORT || 3000;  
app.listen(port, function () {  
    console.log('myapp listening on port ' + port);  
});
```

4. Create a SAP HANA service instance named `myhana`.

The new service instance must have the service type “hana” and the service plan “hdi-shared”, as illustrated in the following command:

```
xs create-service hana hdi-shared myhana
```

5. Replace the content of the `manifest.yml` file in the `node-tutorial` directory with the following:

```
---  
applications:  
- name: myapp  
  path: myapp  
  services:  
    - myhana
```

The XS advanced run time will bind the `myhana` service instance to the `myapp` application during deployment.

6. Update the application on XS advanced.

Install the modified version of the application to XS advanced run-time environment by running the `xs push` command in the `node-tutorial` directory, as follows:

```
xs push
```

7. Display the URL for accessing the updated application after deployment.

```
xs app myapp --urls
```

8. Open a browser window and paste the displayed URL in to start the application.

You should see a message similar to: Using SAP HANA <hana host>:<hana port>.

## Related Information

[Standard Node.js Packages for XS Advanced \[page 519\]](#)

[The SAP NPM Registry \[page 560\]](#)

### 7.1.4.3 Connect to SAP HANA

#### Context

XS advanced provides a package (`@sap/hdbext`) for Node.js applications, which can be used to connect to SAP HANA. To use it, you have to add it to your application.

#### Procedure

1. Navigate to the `<myapp>` directory.
2. Execute `npm install @sap/hdbext@^4.3.4 --save` to install the `@sap/hdbext` module and add it as a dependency in the `package.json` file.

After the installation of the package is complete, the dependencies section in the `package.json` file should have the following content:

```
"dependencies": {  
  "express": "^4.13.0",  
  "@sap/xsenv": "^1.2.7"  
  "@sap/hdbext": "^4.3.4"  
,
```

3. Update the `start.js` file and replace this content:

```
app.get('/', function (req, res) {
```

```
        res.send('Using HANA ' + services.hana.host + ':' + services.hana.port);
    });
}
```

with the following:

```
var hdbext = require('@sap/hdbext');
app.use('/', hdbext.middleware(services.hana));
app.get('/', function (req, res, next) {
    req.db.exec('SELECT CURRENT_UTCTIMESTAMP FROM DUMMY', function (err,
rows) {
        if (err) { return next(err); }
        res.send('Current HANA time (UTC): ' + rows[0].CURRENT_UTCTIMESTAMP);
    });
});
}
```

`hdbext.middleware` will connect to SAP HANA automatically on each access to the specified path ([/](#)) in this case. Afterwards the connection is available in `req.db`. This is the client object of the `hdb`  driver. The connection is closed automatically at the end of the request.

4. Update the application on XS advanced.

Execute the following command in the `node-tutorial` directory.

```
xs push
```

5. Open a browser window with the `<myapp>` application URL.

You should see the current SAP HANA time.

## Related Information

[Standard Node.js Packages for XS Advanced \[page 519\]](#)

### 7.1.4.4 Set up Authentication for your XS Advanced JavaScript Application

Instructions to help you set up user authentication for your JavaScript application in XS advanced.

## Context

Authentication and authorization in the XS advanced run time are implemented via OAuth2 with the help of the User Account and Authentication (UAA) service. For more information, see *Maintaining Application Security* in the *Related Links* below.

## Procedure

1. Create a UAA service instance named `myuaa`.

```
xs create-service xsuaa default myuaa
```

2. Add the `myuaa` service to the application's `manifest.yml` file.

The updated `manifest.yml` should look like the following example:

```
---
applications:
- name: myapp
  path: myapp
  services:
    - myhana
    - myuaa
```

The XS advanced JavaScript run time will bind the `myuaa` service instance (as well as `myhana`) to the `myapp` application during deployment.

### Note

Authentication and authorization are topics that are related to more components than just one application.

The XS advanced run time supports the microservices concept in which a single business application may contain several different applications (microservices). A single entry point for these microservices is provided by the application router (`approuter`). The application router receives all the requests, performs authentication, and forwards the authenticated requests to the appropriate microservice along with a JSON Web Token (JWT). The application router can also perform first-level authorization.

To complete the authorization part of the tutorial you have to add one more application - the application router.

3. Create a directory named `web` in the `node-tutorial` directory.
4. In the `web` directory, create a subdirectory named `resources`.
5. In the `resources` directory, create file `index.html` and add the following content.

```
<html>
<head>
  <title>JavaScript Tutorial</title>
</head>
<body>
  <h1>JavaScript Tutorial</h1>
  <a href="/myapp/">myapp</a>
</body>
</html>
```

This will be the application's entry page.

6. Create a `package.json` file in the `web` directory with the following content:

```
{
  "name": "myapp-entry-point",
  "scripts": {
    "start": "node node_modules/@sap/approuter/approuter.js"
  }
}
```

- In the `web` directory execute `npm install @sap/approuter --save` to install the `@sap/approuter` module and add it as a dependency in the `package.json` file.

After the installation of the package is complete, the dependencies section in the `package.json` file should have the following content:

```
"dependencies": {  
    "approuter": "^2.7.0"  
},
```

- Add the following content to the end of the `manifest.yml` file in the `node-tutorial` directory:

```
- name: web  
  path: web  
  env:  
    destinations: >  
      [  
        {  
          "name": "myapp",  
          "url": "<myapp url>",  
          "forwardAuthToken": true  
        }  
      ]  
    services:  
      - myuaa
```

The `destinations` environment variable defines the destinations to which the application router forwards requests.

Set the `url` property to the URL of the `myapp` application, which is displayed in the output generated by the command: `xs app myapp --urls`.

### i Note

The `myuaa` service is bound to the `web` application during deployment.

- Create the application descriptor (`xs-app.json`) in the `web` directory and add the following content:

```
{  
  "routes": [  
    {  
      "source": "^/myapp/(.*)$",  
      "target": "$1",  
      "destination": "myapp"  
    }  
  ]  
}
```

- In the `myapp` execute `npm install @sap/xssec --save` to install the `@sap/xssec` module and add it as a dependency in the `package.json` file.
- Execute `npm install passport --save` to install the `passport` module and add it as a dependency in the `package.json` file.

The list of dependencies should look like the following example:

```
"dependencies": {  
  "express": "^4.13.0",  
  "@sap/xsenv": "1.2.7",  
  "@sap/hdbext": "^4.3.4",  
  "@sap/xssec": "^1.1.0",  
  "passport": "^0.3.2"
```

```
}
```

12. Verify that the request is authenticated by checking the JWT token included in the request.

Replace the content of the application startup file (`myapp/start.js`) with the following.

```
var express = require('express');
var xsenv = require('@sap/xsenv');
var passport = require('passport');
var JWTStrategy = require('@sap/xssec').JWTStrategy;
var app = express();
var services = xsenv.getServices({ hana:'myhana', uaa:'myuaa' });
passport.use(new JWTStrategy(services.uaa));
app.use(passport.initialize());
app.use(passport.authenticate('JWT', { session: false }));
app.get('/', function (req, res, next) {
    res.send('Application user: ' + req.user.id + '<br>' + 'HANA user: ' +
services.hana.user);
});
var port = process.env.PORT || 3000;
app.listen(port, function () {
    console.log('myapp listening on port ' + port);
});
```

13. Go to the `node-tutorial` directory and deploy the application by executing the following command.

```
xs push
```

This command updates the `myapp` application and also deploys the `web` application.

#### Note

From this point in the tutorial we make requests using the URL of the `web` application, which forwards the requests to the `myapp` application.

14. Find the URL of the `web` application using the `xs apps` command.
15. Request the `web` application with path `/myapp/`.
16. Enter the credentials of a valid user. If no custom identity provider (IDP) is configured, you should use the credential of a known SAP HANA user.

You should see a message similar to the following in the browser:

- ***Application user: <login user>***  
The name of the user who logs on to the application.
- ***HANA user: <db user>***  
The technical user used to connect to SAP HANA.

#### Note

An application user is not the same as a database user.

#### Note

Both `myapp` and `web` applications are bound to the same UAA service instance - `myapp`. In this scenario authentication is handled by the `web` application.

## Related Information

[Maintaining Application Security in XS Advanced \[page 758\]](#)

### 7.1.4.5 Set up Authorization for your XS Advanced JavaScript Application

Instructions to help you set up user authorization for your JavaScript application in XS advanced.

#### Context

The web application can also handle user authorization, for example, with defined scopes and role templates. Scopes and role templates are defined in the application security descriptor (`xs-security.json`), and the application router's descriptor (`xs-app.json`) can contain configurations with authorization restrictions, for example, the specific scopes a user requires to access a resource. For information about setting up authorization scopes, role assignment, and role collections, see *Related Links* below.

It is a general recommendation to implement authorization in the different microservices as well.

#### Procedure

1. Add an authorization check to your application.

To add an authorization check in the `myapp` application, you need to modify the application's startup file `start.js` file; replace the express handler for `GET` requests to path “`/`” with the following code:

```
app.get('/', function (req, res, next) {
  var isAuthorized = req.authInfo.checkScope('uaa.user');
  if (isAuthorized) {
    res.send('Application user: ' + req.user.id);
  } else {
    res.status(403).send('Forbidden');
  }
});
```

2. Deploy the modified application.

You can update the `myapp` application by redeploying, for example, by executing the command `xs push myapp` in the `node-tutorial` directory.

In this example the `myapp` application checks whether the current user has the authorization scope `uaa.user`. Since this is a default scope, the authorization check is successful.

➔ Tip

It is possible to create a UAA service instance with scopes that are specific to your application.

## Related Information

[Standard Node.js Packages for XS Advanced \[page 519\]](#)

[Setting Up Security Artifacts \[page 757\]](#)

[Maintaining XS Advanced Application Routes and Destinations \[page 716\]](#)

### 7.1.4.6 Set up Logging and Tracing for your XS Advanced JavaScript Application

Step-by-step instructions to help you write log and trace files for your JavaScript application in XS advanced.

#### Context

SAP provides a Node.js package for logging and tracing (@sap/logging).

**i** Note

Logs are for the administrator of an application; trace files are for developers or support.

Events that need to be logged are related to how an application operates. For example, entries are created in logs and trace files if a user has been refused access to an application after too many failed log-in attempts, or the application cannot display results returned by a remote HTTP service because the remote server is down. The administrator of an application does not need to know how it is implemented; they should just be able to determine the state of the application itself.

Traces are mainly used when a problem has occurred and further investigation is necessary at the JavaScript code level.

To consume the @sap/logging library, execute the following steps:

#### Procedure

1. Go to the myapp directory.
2. Execute `npm install @sap/logging --save` to install the @sap/logging module and add it as a dependency in the package.json file.

```
"dependencies": {
  "express": "^4.13.0",
  "@sap/xsenv": "1.2.7",
  "@sap/hdbext": "^4.3.4",
  "@sap/xssec": "^1.1.0",
  "passport": "^0.3.2",
  "@sap/logging": "^3.0.0"
}
```

3. Replace the content of the application startup file (`start.js`) with the following code:

```
var express = require('express');
var xsenv = require('@sap/xsenv');
var passport = require('passport');
var JWTStrategy = require('@sap/xssec').JWTStrategy;
var logging = require('@sap/logging');
var appContext = logging.createAppContext();
var app = express();
app.use(logging.expressMiddleware(appContext));
passport.use(new JWTStrategy(xsenv.getServices({uaa:{tag:'xsuaa'}}).uaa));
app.use(passport.initialize());
app.use(passport.authenticate('JWT', { session: false }));
app.get('/', function (req, res, next) {
  var logger = req.loggingContext.getLogger('/Application');
  var tracer = req.loggingContext.getTracer(__filename);
  var isAuthorized = req.authInfo.checkScope('example.scope');
  if (isAuthorized) {
    tracer.debug("Authorization success. User: " + req.user.id + ", Path: './.');
    res.send('Application user: ' + req.user.id);
  } else {
    logger.info("Authorization failed. User: " + req.user.id + ", Path: './.');
    res.status(403).send('Forbidden');
  }
});
var port = process.env.PORT || 3000;
app.listen(port, function () {
  console.log('myapp listening on port ' + port);
});
```

### Note

To demonstrate what happens with a failed authorization, the used scope (`example.scope`) is not among the scopes the test user should possess.

4. Update the `myapp` application by executing the command `xs push myapp` in the `node-tutorial` directory.
5. Call the web application using the path `/myapp/`.

There will be a message in the Web browser saying you are not authorized.

6. Check the logs of the `myapp` application.

In a command shell, execute the command `xs logs myapp --recent`. The displayed output should contain an entry in the log format used by the `@sap/logging` library with the severity level “info” and a message similar to the following example:

```
Authorization failed. User: <HANA-user>, Path: '/'.  
INFO [2018-01-11T11:45:11.123Z] [myapp] [id: 1] [path: /] [method: GET] [status: 403] [error: Authorization failed. User: <HANA-user>, Path: '/'.]
```

If you change the required user scope in the JavaScript code to one already assigned to the requesting user (for example, “`uaa.user`”) and update the application in the XS advanced run time, for example, using the `xs push myapp` command, then a trace entry with severity status `debug` is generated.

### Tip

The `debug` level is not enabled by default; this entry is only generated if the `debug` level is enabled for the application.

7. Enable debug tracing for the application.

```
xs set-logging-level myapp /start.js debug
```

**i Note**

If you are using SAP Web IDE for SAP HANA, you can use the <XS\_APP\_LOG\_LEVEL> environment variable to set the logging level. In SAP Web IDE for SAP HANA, the default application logging level is ERROR.

8. Call the web application using the path `/myapp/`.

The result should contain the name of the requesting user.

9. View the application trace files.

In a command shell, execute the command `xs logs myapp --recent` to view the trace entries. The trace entries with severity `debug` should contain a message similar to the following example:

```
Authorization success. User: <HANA-user>, Path: '/'
```

## Related Information

[Standard Node.js Packages for XS Advanced \[page 519\]](#)

### 7.1.4.7 Use the XSJS Compatibility Layer in XS Advanced

The XSJS compatibility layer is a framework that allows you to run XS classic JavaScript applications in XS advanced.

## Procedure

1. Create a new directory inside `node-tutorial` named `xsjs`.
2. Create `package.json` file in the `xsjs` directory containing the following code.

```
{  
  "name": "xsjs",  
  "version": "0.0.1",  
  "scripts": {  
    "start": "node start.js"  
  }  
}
```

3. Execute `npm install @sap/xsenv@^1.2.7 --save` to install the `@sap/xsenv` module and add it as a dependency in the `package.json` file.
4. Execute `npm install @sap/xsjs@^1.16.1 --save` to install the `@sap/xsjs` module and add it as a dependency in the `package.json` file.

5. Create `start.js` file in the `xsjs` directory containing the following.

```
var xsjs = require('@sap/xsjs');
var xsenv = require('@sap/xsenv');
var port = process.env.PORT || 3000;
var options = xsenv.getServices({ hana:{tag:'hana'}, uaa:{tag:'xsuaa'} });
xsjs(options).listen(port);
console.log('XSJS application listening on port %d', port);
```

The XSJS compatibility layer takes an object containing configurations. In this example SAP HANA and UAA configurations are provided.

6. Create a new directory named `lib` inside the directory `xsjs`.
7. Create a new file named `get-time.xsjs` inside the `lib` directory and add the following content to it.

```
var conn = $.hdb.getConnection();
var resultSet = conn.executeQuery('SELECT CURRENT_UTCTIMESTAMP FROM DUMMY');
$.response.setBody('Current HANA time (UTC) : ' +
resultSet[0].CURRENT_UTCTIMESTAMP);
conn.close();
```

8. Navigate to the `node-tutorial` directory and add the following content at the end of the `manifest.yml` file.

```
- name: xsjs
  path: xsjs
  services:
    - myhana
    - myuaa
```

9. Deploy the XS JavaScript application with the following command executed from the `node-tutorial` directory:

```
xs push xsjs
```

10. Get the URL of the `xsjs` application by running the `xs app xsjs --urls` command.

### Note

Authentication is enabled by default in the compatibility layer; you are not able to directly access the `xsjs` application.

11. Update the configuration for the Web application in the `manifest.yml` file in the `node-tutorial` directory:

```
- name: web
  path: web
  env:
    destinations: >
      [
        {
          "name": "myapp",
          "url": "<myapp url>",
          "forwardAuthToken": true
        },
        {
          "name": "xsjs",
          "url": "<xsjs url>",
          "forwardAuthToken": true
        }
      ]
  services:
```

- myuaa

Set the `url` property of the `xsjs` destination to the URL of the `xsjs` application.

#### Note

The application-router based application can forward requests to two microservices: the `myapp` application and the `xsjs` application. This should be configured in the XS advanced application descriptor (`xs-app.json`) located in the `web` directory.

12. Navigate to the `web` directory and replace the content of the application descriptor (`xs-app.json`) with the following content:

```
{  
  "routes": [  
    {  
      "source": "^/myapp/(.*)$"  
      "target": "$1"  
      "destination": "myapp"  
    },  
    {  
      "source": "^/xsjs/(.*)$"  
      "target": "$1"  
      "destination": "xsjs"  
    }  
  ]  
}
```

13. Add a link to the `xsjs` application in `web/resources/index.html`, which should then look like the following example:

```
<html>  
<head>  
  <title>JavaScript Tutorial</title>  
</head>  
<body>  
  <h1>JavaScript Tutorial</h1>  
  <a href="/myapp/">myapp</a>  
  <p>  
    <a href="/xsjs/get-time.xsjs">xsjs</a>  
  </body>  
</html>
```

14. Navigate to the `node-tutorial` directory and execute the command `xs push web` to update the `web` application.
15. Display the URL required for access to the XS advanced application `web` by executing the command `xs apps` command and use the URL to start the application in a Web browser.

#### Note

You may need to log in again, for example, if your session has expired.

16. Choose the link `xsjs` to see a page displaying the current SAP HANA time.

## Related Information

[Tutorial: Setting up your JavaScript Application in XS Advanced \[page 572\]](#)

## 7.1.4.8 Trouble Shoot XS Advanced JavaScript Applications

Investigate problems that occur during deployment and execution of an XS advanced application.

### Enhancing the logs

There are several possibilities to enhance the logs for Node.js applications in XS advanced. SAP-specific packages usually use the `@sap/logging` module. With the `@sap/logging` module, you can set the logging and tracing severity level to the most verbose level in the following ways:

- Set the environment variable `XS_APP_LOG_LEVEL` to `debug`.

#### i Note

You must restart the application to reset the logging level.

- Use the `xs set-logging-level` command, as illustrated in the following example:

```
xs set-logging-level <application-name> '*' debug
```

#### i Note

You do not need to restart the application to reset the logging level.

Other Node.js packages (or dependencies of SAP-specific packages) might use different logging mechanisms. For example, many applications use the `debug` package. Additional traces can also be enabled by setting the `<DEBUG>` environment variable; setting `<DEBUG>` to `*` will write a large amount of information to the logs and traces very quickly, which could cause problems with disk space.

Internal Node.js traces can be enabled with the `NODE_DEBUG` environment variable.

#### ⚠ Caution

Enabling some of these options may write sensitive data to logs and traces in an insecure form.

### Related Information

[The SAP HANA XS Advanced JavaScript Run Time \[page 510\]](#)

[Download and Consume SAP Node.js Packages \[page 516\]](#)

[Tutorial: Setting up your JavaScript Application in XS Advanced \[page 572\]](#)

## 7.1.4.9 Debug an XS Advanced JavaScript Application Using Command-Line Tools

You can use this procedure to debug your Node.js applications.

### Prerequisites

The XS command-line client is installed on the machine where you will debug a Node.js application deployed on the XS advanced run time.

#### ➔ Tip

For more information about downloading and installing the XS CLI client, see *Related Links* below.

### Context

To debug an XS advanced JavaScript application using the XS CLI, perform the following steps:

### Procedure

1. Enable debug **mode** for the Node.js application you want to debug, for example, using the following command:

```
xs enable-debugging <application-name> --startup
```

This will open a secure debugging connection from the machine on which the XS command line tool is running to the application instance. The port the debugger should attach to is part of the output.

#### ➔ Tip

You can configure which port to be used with the `--port` option.

2. Configure a debugger of choice to attach to the host where the XS command-line tool is running and the port displayed in the output of the `xs enable-debugging` command.

#### i Note

If you are using a custom debugging tool, bear in mind that tools that try to read sources directly from the file system are not suitable for debugging applications deployed on the XS advanced run time.

The standard command-line debugger client for Node.js is a suitable debugging tool; it can be started with the following command:

```
node debug localhost:<debug-port>
```

`<debug-port>` is the one displayed in the output returned by the `xs enable-debugging` command. For more information about how to use the standard XS advanced debugging tool, see *Related Links* below.

## Related Information

[The XS Command-Line Interface Reference \[page 848\]](#)

[Node.js Debugger Documentation](#)

## 7.2 The SAP HANA XS Advanced Java Run Time

SAP HANA XS advanced provides a Java run time to which you can deploy your Java applications.

The Java run time for SAP HANA XS advanced provides a Tomcat or TomEE run time to deploy your Java code. During application deployment, the build pack ensures that the correct SAP Java Virtual Machine (JVM) is provided and that the appropriate data sources for the SAP HANA HDI container are bound to the corresponding application container.

### Note

XS advanced makes no assumptions about which frameworks and libraries to use to implement the Java micro service.

XS advanced does, however, provide the following components to help build a Java micro service:

- TomEE and Tomcat run times
- Setup of SAP HANA data sources
- Java library to support authentication with JSON Web Tokens (JWT)  
Used for communication between the application router, micro services, and the database
- Java library to better support SAP HANA Core Data Services (CDS)

To use Tomcat on SAP JVM 8, specify the `java-buildpack` in your XS advanced application's deployment manifest (`manifest.yml`), as illustrated in the following example:

### Sample Code

```
applications:  
  - name: my-java-service  
    path: path/to/<app>.war  
    services:  
      - my-hdi-container  
      - my-uaa-instance
```

## 7.2.1 Download and Consume Java Libraries

A selection of SAP-specific and ready-to-use Java client libraries is available for download from the SAP Service Marketplace.

### Prerequisites

You have installed Maven on your development system.

### Context

To download the SAP Java Client libraries, perform the following steps:

### Procedure

1. Download the ZIP archive xs\_JAVA\_1 from SAP Service Marketplace.

The ZIP archive is available in the following location:  [Support Portal](#)  [Download Software](#).

#### Tip

To find the ZIP file for download, type “XS JAVA 1” in the [Search](#) box.

2. Extract the contents of the XS\_JAVA archive into a folder in your local file system.
3. Run a Maven build.

Navigate to the folder in your local file system where you extracted the Java client libraries and run a Maven build with the following command:

```
mvn clean install
```

The Java client libraries are installed in the local Maven repository from where they can be consumed by your Maven project.

### Related Information

[XS\\_JAVA: Standard Java Client Libraries for XS Advanced \[page 591\]](#)

[Tutorials: Setting Up the Java Run-Time Environment in XS Advanced \[page 602\]](#)

## 7.2.1.1 XS\_JAVA: Standard Java Client Libraries for XS Advanced

A collection of Java client libraries developed by SAP is provided to help you develop Java applications for SAP HANA XS advanced.

SAP HANA includes a selection of standard Java libraries, which are available for download from the SAP Service Marketplace (SMP) to customers and partners who have the appropriate access authorization. On SAP Service Marketplace, search for the software component named `XS_JAVA`, which is an archive that contains the packages listed in the following table:

➔ Tip

For more details of the package contents, see the `README` file in the corresponding package or, where available, the JavaDoc reference guide.

Table 85: XS\_JAVA Package Contents

Library	Description
<code>xs-env</code>	A utility for setting up and accessing XS advanced environment variables and services
<code>audit-java-client-api</code>	A client that enables applications to use the XS advanced audit-log service
<code>java-container-security</code>	A convenient Java wrapper for the security context of an authentication
<code>java-js-client</code>	A convenient Java wrapper for the Job Scheduler REST API end points
<code>jds</code>	A set of tools that facilitate the consumption of Core Data Services (CDS) artifacts in Java applications
<code>gateway.v4</code>	A client library that enables you to set up an OData (v4) service for use by a Java application in an XS advanced
<code>sap-java-hdi</code>	The SAP HANA Deployment Infrastructure (HDI) client library for Java applications

### xs-env

A utility that enables easy setup of and access to XS advanced environment variables and services.

Objects of type `VcapApplication` store information about applications and objects of type `VcapServices` store information about service instances.

The following example shows how to create `VcapApplication` from the `<VCAP_APPLICATION>` system property:

Sample Code

```
public static VcapApplication fromSystemProperty() throws  
IllegalArgumentException {
```

```
        return VcapApplication.from(System.getProperty(VCAP_APPLICATION));
    }
```

If the system property is not set, an empty instance is returned. To check if the system property exists, use `getProperty()`, as illustrated in the following example:

```
System.getProperty(VCAP_APPLICATION) != null;
```

The following example shows how to create `VcapApplication` from the `<VCAP_APPLICATION>` environment variable.

### Sample Code

```
public static VcapApplication fromEnvironment() throws
IllegalArgumentException {
    return VcapApplication.from(System.getenv(VCAP_APPLICATION));
}
```

If the environment variable is not set, an empty instance is returned. To check if the environment variable exists, use `getenv()`, as illustrated in the following example:

```
System.getenv().containsKey(VCAP_APPLICATION);
```

The following example shows how to create `VcapServices` from the `<VCAP_SERVICES>` system property:

### Sample Code

```
public static VcapServices fromSystemProperty() throws
IllegalArgumentException {
    return VcapServices.from(System.getProperty(VCAP_SERVICES));
}
```

If the system property is not set, an empty instance is returned. To check if the system property exists, use `getProperty()`, as illustrated in the following example:

```
System.getProperty(VCAP_SERVICES) != null;
```

The following example shows how to create `VcapServices` from the `<VCAP_SERVICES>` environment variable:

### Sample Code

```
public static VcapServices fromEnvironment() throws IllegalArgumentException {
    return VcapServices.from(System.getenv(VCAP_SERVICES));
}
```

If the environment variable is not set, an empty instance is returned. To check if the environment variable exists, use `getenv()`, as illustrated in the following example:

```
System.getenv().containsKey(VCAP_APPLICATION);
```

## audit-java-client-api

SAP HANA includes a client for the SAP HANA XS advanced audit-log service that is used by Web applications to send audit-log messages to a remote persistent storage.

You can configure an application to use the audit-log service to write entries detailing important events in the audit log. For that purpose, you configure an audit log API and then bind the application to the service. To use the client, the following high-level steps are necessary:

1. Include the audit-log API in your Maven project.

### Sample Code

Define the dependency in `pom.xml`

```
<dependency>
    <groupId>com.sap.xs.auditlog</groupId>
    <artifactId>audit-java-client-api</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <scope>provided</scope>
</dependency>
```

2. Declare the resource.

- Integration in Tomcat:

### Sample Code

Add a new resource in `META-INF/context.xml`.

```
<?xml version='1.0' encoding='utf-8'?>
<Context>
    <Resource name="audit" auth="Container"
        type="com.sap.xs.audit.api.AuditLogMessageFactory"
        factory="com.sap.xs.XSObjectFactory" singleton="true" />
</Context>
```

- Integration in TomEE:

### Sample Code

Add a new resource in `WEB-INF/resources.xml`

```
<?xml version='1.0' encoding='utf-8'?>
<resources>
    <Resource id="audit" type="com.sap.xs.audit.api.AuditLogMessageFactory"
        provider="xs.openejb:XS Audit Log Message Factory Provider"/>
</resources>
```

3. Enable access to the audit-log message factory.

- Java Naming and Directory Interface (JNDI) look-up:

```
Context ctx = new InitialContext();
AuditLogMessageFactory auditlogMesageFactory = (AuditLogMessageFactory)
ctx.lookup("java:comp/env/audit");
```

- Resource injection:

```
@Resource(name="audit")
```

```
private AuditLogMessageFactory messageFactoryInj;
```

## ➔ Tip

For more information about setting up the client to use the audit-log API, see *Related Information*.

## java-container-security

A client library that provides a convenient Java wrapper for the security context of an authentication.

- Add the security library to the `pom.xml`.
- Edit the `spring-security.xml` file to secure the resource end-points and requests.

To maintain the details specified in the Spring security configuration (`spring-security.xml`), for example:

### Sample Code

`spring-security.xml`

```
<?xml version="1.0" encoding="UTF-8" ?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:oauth="http://www.springframework.org/schema/security/oauth2"
       xmlns:sec="http://www.springframework.org/schema/security"
       ...>
    <!-- protect secure resource endpoints ===== -->
    <sec:http pattern="/**" create-session="never"
              entry-point-ref="oauthAuthenticationEntryPoint"
              access-decision-manager-ref="accessDecisionManager"
              authentication-manager-ref="authenticationManager"
              use-expressions="true">
        <sec:anonymous enabled="false" />
        <sec:intercept-url pattern="/rest/addressbook/deletedata"
                            access="#oauth2.hasScope('${xs.appname}.Delete')"
                            method="GET" />
        <sec:intercept-url pattern="/**" access="isAuthenticated()"
                            method="GET" />
        <sec:custom-filter ref="resourceServerFilter"
                           before="PRE_AUTH_FILTER" />
        <sec:access-denied-handler ref="oauthAccessDeniedHandler" />
    </sec:http>
    <!-- secure the requests ===== -->
    <oauth:resource-server id="resourceServerFilter"
                          resource-id="springsec" token-services-
                          ref="offlineTokenServices" />
    <bean id="offlineTokenServices"
          class="com.sap.xs2.security.commons.SAPOfflineTokenServices">
        <property name="verificationKey" value="${xs.uaa.verificationkey}" />
        <property name="trustedClientId" value="${xs.uaa.clientid}" />
        <property name="trustedIdentityZone" value="${xs.uaa.identityzone}" />
    </bean>
</beans>
```

For example, to enable a Java application to obtain user information related to a security context in XS advanced:

### Sample Code

```
try {
    UserInfo userInfo = SecurityContext.getUserInfo();
    logonName = userInfo.getLogonName();
    email = userInfo.getEmail();
    givenName = userInfo.getValue("givenName");
    familyName = userInfo.getValue("familyName");
} catch (UserInfoException e) {
    // handle exception
}
```

## java-js-client

This client library is a Java wrapper that enables convenient access to the XS advanced Job Scheduler's REST API end points. The Job Scheduler service enables consumers to create, schedule, manage, and monitor application tasks within the SAP HANA XS advanced run-time, both on-premise and in Cloud Foundry.

### Usage

The following code shows how to obtain an instance of the `SchedulerClient` from the factory:

```
Client client = SchedulerFactory.getSchedulerClient();
```

You must set the `jobscheduler` credentials to the client, as illustrated in the following code example:

```
client.setApi("https://host.acme.com:4242");
client.setUserName("a9a32ac0-5522-11e5-adbd-13adbfaf2ac7");
client.setPassword("6081473c-6563-43ca-a8dd-61abb987330f");
```

To enable the client to consume the scheduler API, you must create an instance of the `SchedulerManager`, as illustrated in the following example:

```
SchedulerManager schedulerManager = client.getSchedulerManager();
```

The following example shows how to **update** a scheduled job:

```
newJob.setActive(false);
newJob.setUser("test");
newJob.setPassword("Toor1234");
try {
    newJob = (Job) schedulerManager.update(newJob);
    slf4jLogger.info("Job is updated. Now, the job is " + (newJob.getActive() ? "active" : "inactive"));
} catch (SchedulerException e) {
    slf4jLogger.error("Failed to update the job");
}
```

The following example shows how to **add** a new schedule to an existing job and **update** a schedule:

```
// Create new schedule entity
Schedule newSchedule = new Schedule();
```

```

newSchedule.setDescription("Ping Google every 20 minutes using xs cron");
newSchedule.setCron("* * * * */20 *");
newSchedule.setJob(newJob);
try {
    Schedule createdSchedule = (Schedule) schedulerManager.create(newSchedule);
    slf4jLogger.info("New schedule is created with id " +
createdSchedule.getId());
    // -----Update schedule-----
    createdSchedule.setStartTime(startTime);
    createdSchedule.setEndTime(endTime);
    Schedule schedule2 = (Schedule) schedulerManager.update(createdSchedule);
    slf4jLogger.info("schedule updated with start time "
        + schedule2.getStartTime().getDate() + " end time "
        + schedule2.getEndTime().getDate());
    newJob = schedulerManager.searchJobById(newJob.getId());
    slf4jLogger.info("Now, the number of schedules is " +
newJob.getSchedules().size());
} catch (SchedulerException e) {
    slf4jLogger.error("Failed to update the schedule");
}

```

The following example shows how to **activate** or **deactivate** all schedules configured for a job:

```

try {
    schedulerManager.activateAllSchedules(newJob);
    Collection<Schedule> schedules = newJob.getSchedules();
    boolean scheduleActive = false;
    for (Iterator<Schedule> scheduleIterator = schedules.iterator();
scheduleIterator.hasNext();)
        Schedule schedule2 = (Schedule) scheduleIterator.next();
        scheduleActive = scheduleActive & schedule2.getActive();
    }
    slf4jLogger.info("All the schedules are now " + (scheduleActive == true ?
"active" : "inactive"));
} catch (SchedulerException e1) {
    slf4jLogger.error("Failed to activate the schedules");
}
try {
    schedulerManager.deactivateAllSchedules(newJob);
    newJob = schedulerManager.searchJobById("90009");
    slf4jLogger.info("All schedules are deleted. The number of schedules is " +
newJob.getSchedules().size());
} catch (SchedulerException e1) {
    slf4jLogger.error("Failed to deactivate the schedules");
}

```

The following example shows how to **delete** a scheduled job:

```

try {
    schedulerManager.delete(newJob);
    slf4jLogger.info("The job is deleted");
} catch (SchedulerException e) {
    slf4jLogger.error("Failed to delete the job");
}

```

The following example shows how to **update** a log:

```

Log log = new Log();
log.setJobId("1");
log.setScheduleId("da722278-0a22-440c-a4d6-c599b77617f4");
log.setId("11765A573A618A79E10000000A61A17B");
log.setMessage("New log updated");
log.setSuccess(true);
Log newLog = new Log();
try {

```

```
    newLog = (Log) schedulerManager.update(log);
    slf4jLogger.info("Log is Updated ");
} catch (SchedulerException e2) {
    slf4jLogger.error("Failed to update the log");
}
..
```

## jds

The Java Data Services (JDS) are a set of tools that facilitate the consumption of Core Data Services artifacts in Java applications. More specifically, JDS maps existing CDS data models to corresponding Java Persistence API (JPA) 2.1 data models that may be consumed by Java applications using EclipseLink or Hibernate.

### Usage

Run the tool from the command-line in a command shell:

```
jds> java -jar jds-import-1.0.1-full.jar help
usage: java -jar importer-x.y.z-full.jar <command> <parameters>
Commands:
    help - displays this help text
    userguide - display the JSON user guide
    display - display available entities in namespace
    create - create Java classes for entities
    <config file>.json - process instructions in document (advanced)
```

#### ➔ Tip

For more information about using Java Data Services, see *Related Information*.

### Support and Limitations

The current release of JDS supports basic CDS entities, CDS types, CDS views, and CDS one-to-one associations. All imported CDS entities can be used with any available JPA technology, including queries using JPQL and the Criteria API.

Note the following limitations:

- nokey entity

Entities without keys are not supported by JPA:

- CDS artifact is created.
- `JpaRepresentationEntity` is not created
- Java file is not created

- associations to non-primary key fields

Foreign key associations must not contain foreign keys which are not primary keys:

- CDS artifact is created.
- `JpaRepresentationEntity` is created **without** the association
- Java file is created **without** the association.

## gateway.v4

This is client library that enables you to set up an OData (v4) service for use by a Java application in an XS advanced.

An OData v4 service for use with an XS advanced Java application is described in a design-time OData artifact and referenced in the corresponding CDS document (by means of an @OData.publish annotation) that exposes the specified tables or entities in the annotated context. The definition of your OData service should be placed in the java/odata/v4/ folder of your Java application project module. The following example shows the project structure for an XS advanced Java application which consumes an OData version 4.0 service.

➔ Tip

For more information about using Java Data Services, see *Related Information*.

## sap-java-hdi

The sap-java-hdi client library enables Java applications to access the SAP HANA Deployment Infrastructure (HDI). Currently, the library is available in the following versions:

- sap-java-hdi version 1.0  
Released with SAP HANA 2.0 SPS 01 and still available in SAP HANA 2.0 SPS 02.

i Note

sap-java-hdi version 1.0 is considered feature complete and will not receive any further enhancements in future releases.

- sap-java-hdi version 2.0  
Released with SAP HANA 2.0 SPS 02

➔ Tip

It is recommended to use version 2.0 of sap-java-hdi.

### sap-java-hdi (version 1.0)

Version 1.0 of the Java client library for HDI provides the following **methods** to enable access to HDI functionality:

➔ Tip

For reasons of support and compatibility, it is recommended to use version 2.0 of sap-java-hdi.

- connect
- disconnect
- configureDI
- configureDIParameters

- createContainer
- dropContainer
- configureContainer
- configureContainerParameters
- listLibraries
- listDeployed
- configureLibraries
- listConfiguredLibraries
- status
- read
- readDeployed
- write
- delete
- make
- makeAsync
- grantContainerApiPrivileges
- grantContainerApiPrivilegesWithGrantOption
- revokeContainerApiPrivileges
- grantContainerSchemaPrivileges
- revokeContainerSchemaPrivileges
- grantContainerSchemaRoles
- revokeContainerSchemaRoles

The HDI class provides two constructors to create a new HDI instance. The first can be used to execute non-container-specific tasks:

#### Sample Code

```
HDI(String host, String port, String dbName, String user, String password)
```

The second can be used to enable operations on a container, as illustrated in the following example:

#### Sample Code

```
HDI(String host, String port, String dbName, String user, String password,  
String container)
```

An HDI instance created with the first constructor can be upgraded by using the `setContainer()` command:

#### Sample Code

```
setContainer(String container)
```

The connection to the database is opened and closed with the commands “`connect()`” and “`disconnect()`”.

## sap-java-hdi (version 2.0)

Version 2.0 of the SAP HANA DI (HDI) client library for Java applications wraps the HDI SQL APIs in Java classes and methods. The following main classes are available:

Table 86: HDI JAVA Classes

Class	Description
HDI	Provides access to the _SYS_DI API
Container	Provides access to the API of a specific HDI container
ContainerGroup	Provides access to the API of a specific HDI container group

### ➔ Tip

For reasons of support and compatibility, it is recommended to use version 2.0 of sap-java-hdi.

To use all APIs, the connected user must have the following privileges:

- EXECUTE  
The EXECUTE privilege is required for the corresponding SQL procedures.
- SELECT  
The SELECT privilege is required for all table types in SYS\_DI (TT\*).

### i Note

For all the examples used in this section to illustrate some of the most common usage scenarios for the HDI Java client library `sap-java-hdi`, it is assumed that the `hdiUser` has the privileges required to access `_SYS_DI`. In addition, some APIs require a minimum SAP HANA version. For more information about which SAP HANA version is required by a specific class or method, see the `sap-java-hdi` JavaDoc reference.

The following example shows how to use `sap-java-hdi` to instantiate an HDI API wrapper object for the database “DB1” in the SAP HANA instance running on a host named “someHost” on SQL port 30015 with the user “hdiUser”, the password “Abcd123” and, in addition, using the user’s default schema for temporary tables.

```
HDI hdi = new HDI("someHost", "30015", "DB1", "hdiUser", "Abcd123", "hdiUser");
```

To create a new container group named “someGroup”, use the following code:

### ➔ Tip

The last argument (“null” in this case) is an optional list of key-value parameters for the call.

```
ResultTuple rt = hdi.createContainerGroup("someGroup", null);
System.out.println("Return code: " + rt.getRC());
System.out.println("Messages: " + rt.getMessages());
```

The following example shows how to grant the default container-group administrator privileges for "someGroup" to the user "hdiUser":

```
List<APIPrivilege> groupAdminPrivileges =  
    hdi.getDefaultContainerGroupAdminPrivileges("", "hdiUser");  
    rt = hdi.grantContainerGroupApiPrivileges("someGroup", groupAdminPrivileges,  
    null);
```

The following code shows how to use the sap-java-hdi to instantiate a container-group-API wrapper object for "someGroup" in the database "DB1" in the SAP HANA instance running on host "someHost" with SQL port 30015 with user hdiUser, password Abcd1234 and using the user's default schema for temporary tables:

```
ContainerGroup group = new ContainerGroup("someGroup", "someHost", "30015",  
    "DB1", "hdiUser", "Abcd123", "hdiUser");
```

The following example shows how to create a new container named "someContainer" with the Java HDI API and place it in the container group "someGroup" created in the previous example:

#### ➔ Tip

The API object "group" used in this example was created for the container group "someGroup" (= new ContainerGroup("someGroup", [...]))

```
rt = group.createContainer("someContainer", null);
```

To grant the default container-administrator privileges for the HDI container "someContainer" to the HDI user "hdiUser", consider the following functions:

```
List<APIPrivilege> containerAdminPrivileges =  
    group.getDefaultContainerAdminPrivileges("", "hdiUser");  
    rt = group.grantContainerApiPrivileges("someContainer",  
    containerAdminPrivileges, null);
```

The following functions show how to use the HDI container API to instantiate a container-API wrapper object for someContainer in database DB1 in the SAP HANA instance running on host someHost with SQL port 30015 with user hdiUser, password Abcd1234, and using the user's default schema to store temporary tables:

```
Container container = new Container("someContainer", "someHost", "30015", "DB1",  
    "hdiUser", "Abcd123", "hdiUser");
```

To list the plug-in libraries configured in someContainer, consider the following functions:

```
LibraryInformationResultTuple lrt = container.listConfiguredLibraries(null);  
System.out.println("Configured libraries: " + lrt.getResults());
```

## Related Information

[Download and Consume Java Libraries \[page 590\]](#)

[The SAP HANA XS Advanced Java Run Time \[page 589\]](#)

[Configure an Application to Use the Audit Log Service \[page 611\]](#)

[Enable and Use Java Data Services \[page 629\]](#)

[Defining OData v4 Services for XS Advanced Java Applications \[page 463\]](#)

## 7.2.2 Tutorials: Setting Up the Java Run-Time Environment in XS Advanced

A selection of tutorials that show you how to set up an application to run in the XS advanced Java run-time environment.

### Prerequisites

To perform the tasks described in the following tutorials, the following tools and components must be available:

- SAP HANA XS advanced run time must be installed on your SAP HANA system
- The SAP HANA XS CLI client must be installed on your development machine.

#### ➔ Tip

The package containing the XS CLI client tools (`xs.onpremise.runtime.client_<platform>-<version>.zip`) can be downloaded from the SAP HANA server, from the installation DVD, or from the SAP support portal.

- You must have valid user logon credentials for the target system where XS advanced is running

[Change Application Settings for the XS Advanced Java Run Time \[page 603\]](#)

You can change the run-time settings in your XS advanced Java application's manifest file.

[Configure a Java Application for Logs and Traces \[page 608\]](#)

Configure the collection of log and trace messages generated by a Java application in XS advanced.

[Configure an Application to Use the Audit Log Service \[page 611\]](#)

You can configure an application to use the audit-log service to write entries detailing important events in the audit log. For that purpose, you configure an audit log API and then bind the application to the service.

[Configure Authentication and Authorization \[page 617\]](#)

You configure authentication and authorization by using the integrated container authentication provided with the Java build pack or the Spring security library.

[Configure a Database Connection \[page 621\]](#)

You can configure your application to use a database connection so that the application can persist its data.

[Debug an XS Advanced Java Application \[page 628\]](#)

You can use this procedure only if you can restart your Java application or the application has not yet been pushed.

[Enable and Use Java Data Services \[page 629\]](#)

Enable Java Data Services, a native Java client for using Core Data Services functionality in the XS advanced Java run time.

#### [Run a Java Main Application \[page 633\]](#)

You can create a Java application that starts its own run time.

## Related Information

[Working with the XS Advanced Command-Line Client \[page 78\]](#)

[Getting Started with Application Development in XS Advanced \[page 25\]](#)

[SAP Note 2242468: Setting up the XS Advanced CLI](#) 

### 7.2.2.1 Change Application Settings for the XS Advanced Java Run Time

You can change the run-time settings in your XS advanced Java application's manifest file.

## Set SAP JDK

### Context

You can specify that your application needs SAP JDK by setting the `<JBP_CONFIG_COMPONENTS>` environment variable in its `manifest.yml` file before pushing, as illustrated in the following example:

```
env
  JBP_CONFIG_COMPONENTS: [jres: ['com.sap.xs.java.buildpack.jdk.SAPJVMJDK']]
```

If this environment variable is not set, by default your application will use JRE.

# Set the TomEE Run Time

## Context

The default Java run-time environment in XS advanced is “Tomcat”. However, you can change your desired Java run-time environment in XS advanced to “TomEE” by setting the `<TARGET_RUNTIME>` environment variable in the Java application’s `manifest.yml`, as illustrated in the following example:

### Code Syntax

```
env:  
  TARGET_RUNTIME: tomee
```

# Configure Memory Sizes

## Context

The HEAP, METASPACE and STACK memory types are configured with a corresponding default size. However, the default values can also be customized, as illustrated in the following example.

## Procedure

- Configure the memory sizes during the staging phase.

If this is the first staging of the application (the application is not pushed to SAP HANA XS advanced), provide `<JBP_CONFIG_SAPJVM>` environment variable in the `manifest.yml` file of the application and then stage the application.

### Example

Sample `manifest.yml`

```
---  
applications:  
- name: <app-name>  
  memory: 512M  
...  
env:  
  JBP_CONFIG_SAPJVM: "[memory_calculator: {memory_heuristics: {heap: 85,  
stack: 10}}]"
```

For more information about memory types for the XS advanced Java run-time environment, see *Memory Size Options in Related Information*.

- Configure the memory sizes during the application run time.

This configuration can be changed multiple times after the application is staged and does not require re-staging.

- a. Open the command prompt.
- b. Set custom weights, sizes, and initials.

For more information, see *Memory Size Options* in *Related Links*.

- c. Restart the application.

## Configure Other Memory Options

### Context

You can add custom JVM properties as this configuration is done once and can be changed only when the application is re-staged.

For more information about the Java run-time options in XS advanced , see *Related Links*.

### Procedure

- Use an environment variable in the manifest file.

If this is the first staging of the application (the application has not yet been “pushed” to SAP HANA XS advanced), provide `<JBP_CONFIG_JAVA_OPTS>` environment variable in the `manifest.yml` file of the application and then stage the application.

#### Sample Code

Excerpt from a Java `manifest.yml` file

```
---
applications:
- name: <app-name>
  memory: 512M
...
env:
  JBP_CONFIG_JAVA_OPTS: '[from_environment: false, java_opts: ''-
    DtestJBConfig=%PATH% -DtestJBConfig1="test test" -DtestJBConfig2="%PATH
    %''' ]'
```

#### Note

A single quote is used to enclose the `<JBP_CONFIG_JAVA_OPTS>` string in the YML file because the string can contain any one (or combination) of the following characters: `: {, }, [ , ], ., &, *, #, ?, |, -, <, >, =, !, %, @, \.`.

- Set the environment variable with a the `xs set-env` command.

If the application is already available on SAP HANA XS advanced and the application developer wants to fine-tune the JVM additionally with an additional or modified property, a new value for the

`<JBP_CONFIG_JAVA_OPTS>` environment variable can be specified with the `xs set-env` command. The new values will overwrite the default (or set) values in the corresponding buildpack the next time the application is staged.

### Example

```
xs set-env myapp JBP_CONFIG_JAVA_OPTS "[from_environment: false, java_opts: '-DtestJBPCfg=^%PATH^% -DtestJBPCfg1=\"test test\" -DtestJBPCfg2=^%PATH^%' ]"
```

### Note

When the Java options are specified on the command line with the `xs set-env` command, the string defining the new values must be enclosed in double quotes, for example, "`<New-config_values>`".

## Related Information

[Memory Size Options \[page 606\]](#)

[Java Memory Options \[page 608\]](#)

### 7.2.2.1.1 Memory Size Options

#### Memory Types

There are three memory types that can be sized and for each memory type there is a command-line option, which is passed to the JVM.

- **HEAP**  
The initial and maximum sizes of the HEAP memory are controlled by the `-Xms` and `-Xmx` options respectively.
- **METASPACE**  
The initial and maximum sizes of the METASPACE memory are controlled by the `-XX:MetaspaceSize` and `-XX:MaxMetaspaceSize` options.
- **STACK**  
The size of the STACK is controlled by the `-Xss` option.

#### Changing Memory Settings with the Memory Calculator

The memory calculator uses calculation techniques that allocate the available memory to the memory types. The total available memory is first allocated to each memory type in proportion to its weighting (this is called 'balancing'). If the resultant size of any memory type lies outside its range, the size is constrained to the range, the constrained size is excluded from the remaining memory, and no further calculation is required for that

memory type. After that, the remaining memory is balanced against the memory types that are left, and the check is repeated until no calculated memory sizes lie outside their ranges. These iterations terminate when none of the sizes of the remaining memory types are constrained by their corresponding ranges.

The memory calculator uses the following calculation techniques:

- memory calculations using absolute memory sizes
- memory calculations using relative memory weights

## Setting Custom Weights, Sizes, and Initials

You can set custom values in the manifest file before the application is staged by using the following parameters:

- `memory_sizes`  
The data for this parameter specifies the absolute values for the memory properties of the SAP JVM: -Xms, -Xmx, -Xss, -XX:MaxMetaspaceSize, or -XX:MetaspaceSize.
- `memory_heuristics`  
The data for this parameter specifies the relative weights for the different memory types.
- `memory_initials`  
This parameter is used to adjust the initial values for HEAP and METASPACE types.

### Sample Code

Setting HEAP memory to 8.5 times larger than the STACK memory.

```
env:  
  JBP_CONFIG_SAPJVM:  "[memory_calculator: {memory_heuristics: {heap: 85,  
stack: 10}}]"
```

Furthermore, you can change the sizes during the application run time with the `xs set-env` command, as follows:

- To set custom weights, use the `JBP_CONFIG_SAPJVM_MEMORY_WEIGHTS` environment variable.

### Sample Code

```
xs set-env myapp JBP_CONFIG_SAPJVM_MEMORY_WEIGHTS "heap:5,stack:1,metaspace:  
3,native:1"
```

- To set custom sizes, use the `JBP_CONFIG_SAPJVM_MEMORY_SIZES` environment variable.

### Sample Code

```
xs set-env myapp JBP_CONFIG_SAPJVM_MEMORY_SIZES "heap: 30m..400m, stack:  
2m..., metaspace: 10m..12m"
```

- To set custom initials, use the `JBP_CONFIG_SAPJVM_MEMORY_INITIALS` environment variable.

### Sample Code

```
xs set-env myapp JBP_CONFIG_SAPJVM_MEMORY_INITIALS "heap: 50%, metaspace: 50%"
```

### Note

If you specify values for weight and size at the same time, the size values take precedence over the weight values.

## Java Options

You can set the Java options by using the following parameters:

- `from_environment`  
This parameter expects a Boolean value which specifies if the value of the `<JAVA_OPTS>` environment variable must be taken into account. By default, it is set to "false".
- `java_opts`  
This section is used for additional Java options such as `-Xloggc:$PWD/beacon_gc.log -verbose:gc -XX:NewRatio=2`. By default, no additional options are provided.

### 7.2.2.2 Configure a Java Application for Logs and Traces

Configure the collection of log and trace messages generated by a Java application in XS advanced.

## Prerequisites

- You have installed and configured Maven.
- You use SAP JVM 8 and your `<JAVA_HOME>` environment variable points to this location.
- You do not have any SLF4J and logback JAR files in the application.

### Note

The log JARs for the SLF4J and logback are already included in the XS advanced Java run-time environment; importing them in the application could cause problems during class loading.

## Context

The recommended framework for logging is Simple Logging Facade for Java (SLF4J). To use this framework, you can create an instance of the `org.slf4j.Logger` class. To configure your XS advanced Java application to generate logs and traces, and if appropriate set the logging or tracing level, perform the following steps:

## Procedure

1. Instruct Maven that the application should not package the SLF4J dependency; this dependency is already provided by the run time.

To do this, set the dependency scope to `provided` in the file `pom.xml`:

```
<dependency>
<groupId>org.slf4j</groupId>
<artifactId>slf4j-api</artifactId>
<version>1.7.12</version>
<scope>provided</scope>
</dependency>
```

2. Import the SLF4J classes into the application code (`org.slf4j.Logger` and `org.slf4j.LoggerFactory` libraries) and develop the log handling code.

### Example

Ensure that `debug` and `info` log messages from the Java application appear in the log files.

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
public void logging () {
    final Logger logger =
LoggerFactory.getLogger(Log4JServlet.class);
    logger.debug("this is record logged through SLF4J
param1={}, param2={}.", "value1", 1);
    logger.info("This is slf4j info");
}
```

3. If necessary, set a logging level.

By default, the XS advanced Java application uses the logging level `ERROR`, but you can change this setting with the `xs set-logging-level` command. For more information, see *Logging Levels* in *Related Information* below.

### Note

To enable logs to be written in the log files, the logging level of the corresponding logger needs to be set to the value that matches the severity of the log entries in the application code. For example, for the sample in the previous step, the severity of the `Log4Servlet` needs to be set to `DEBUG` or `INFO` in order to write the appropriate entry in the corresponding log or trace file.

## Related Information

[Installing Apache Maven](#) ↗

[Logging Levels \[page 610\]](#)

### 7.2.2.2.1 Logging Levels

The Log4J application supports the following logging levels:

Table 87: Logging Levels

Logging Level	Description
ALL	All logging levels are displayed in the logs and traces.
DEBUG	This level is used for debugging purposes. When you set this level, DEBUG, INFO, WARN, ERROR and FATAL levels are displayed in the application logs.
INFO	This level is used for information purposes. When you set this level, the INFO, WARN, ERROR and FATAL levels are displayed in the application logs.
WARN	This level is used for warning purposes. When you set this level, the WARN, ERROR and FATAL levels are displayed in the application logs.
ERROR	This level is used to display errors. When you set this level, ERROR and FATAL levels are displayed in the application logs.
FATAL	This level is used to display critical errors. When you set this level, only the FATAL level is displayed in the application logs.
OFF	Use this setting to turn off logging.

Before you use the command to set a logging level, make sure that your application is configured to support the logger location. To set logging levels for a specific location, you need to use the `xs set-logging-level` command, as illustrated in the following example:

```
xs set-logging-level <applications name> <logger path> <log level>
```

For more information about the `xs set-logging-level` command, see *XS CLI: Application Management* in *Related Information*.

#### i Note

You can change your logger location and messages by editing the `Log4JServlet` class in the Log4J application.

The specified logging level is applied during application startup.

## Related Information

[XS CLI: Application Management \[page 852\]](#)

### 7.2.2.3 Configure an Application to Use the Audit Log Service

You can configure an application to use the audit-log service to write entries detailing important events in the audit log. For that purpose, you configure an audit log API and then bind the application to the service.

## Obtain the Audit Log API

### Context

You use an audit log API with Web applications by means of standard Java EE mechanisms.

### Procedure

1. Include the audit log API in your Maven project.

#### Example

Sample dependency in `pom.xml`

```
<dependency>
    <groupId>com.sap_xs.auditlog</groupId>
    <artifactId>audit-java-client-api</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <scope>provided</scope>
</dependency>
```

2. Declare the resource.

- o Integration in Tomcat

Add a new resource in `META-INF/context.xml`.

#### Sample Code

```
<?xml version='1.0' encoding='utf-8'?>
<Context>
```

```
<Resource name="audit" auth="Container"  
type="com.sap.xs.audit.api.AuditLogMessageFactory"  
factory="com.sap.xs.XSObjectFactory" singleton="true" />  
</Context>
```

- Integration in TomEE  
Add a new resource in WEB-INF/resources.xml.

#### Sample Code

```
<?xml version='1.0' encoding='utf-8'?>  
<resources>  
  <Resource id="audit" type="com.sap.xs.audit.api.AuditLogMessageFactory"  
  provider="xs.openejb:XS Audit Log Message Factory Provider"/>  
</resources>
```

3. Access the audit log message factory.

- Access through Java Naming and Directory Interface (JNDI) look-up

```
Context ctx = new InitialContext();  
AuditLogMessageFactory auditlogMesageFactory = (AuditLogMessageFactory)  
ctx.lookup("java:comp/env/audit");
```

- Access through resource injection

```
@Resource(name="audit")  
private AuditLogMessageFactory mesageFactoryInj;
```

## Write an Audit Log Message

### Prerequisites

You have installed the XS Java client libraries.

For more information about the libraries installation, see SAP Note [2283600](#).

### Context

You use the methods of the audit log interfaces to specify what message to be displayed in the Audit Log UI.

### Procedure

- Use the general interface to specify the user that triggers the audit log event.

## Sample Code

```
AuditLogMessage message = new AuditLogMessage();
message.setUser("John Smith")
```

This user is applicable for all audit log categories.

- Use the audit log message factory to create a message for a specific category.

## Sample Code

Sample message for the data read access category.

```
AuditLogMessageFactory messageFactory = new AuditLogMessageFactory();
DataAccessAuditMessage message =
messageFactory.createDataAccessAuditMessage();
```

- Log an event with a transactional logic

### Note

The transactional logic is applicable only for the configuration change events.

## Sample Code

A message for a configuration change event that is about to happen.

```
ConfigurationChangeAuditMessage message =
messageFactory.createConfigurationChangeAuditMessage();
...
message.logPrepare();
```

- Log a configuration change event

Use the `AuditLogMessage`, `AuditLogMessageFactory`, `ConfigurationChangeAuditMessage`, and `TransactionalAuditLogMessage` interfaces to log a configuration change event.

## Sample Code

Logging for event "User John Smith changed the configuration for the address of object Library."

```
ConfigurationChangeAuditMessage message =
messageFactory.createConfigurationChangeAuditMessage();
message.setUser("John Smith");
message.setObjectId("class=com.sap.example.Library"); //a string that could
uniquely identify the object
message.addValue("address", "Old address", "New address");
message.log();
```

- Log a data read access event

Use the `AuditLogMessage`, `AuditLogMessageFactory`, and `DataAccessAuditMessage` interfaces to log a data access event.

## Sample Code

Logging for event "User John Smith read the address attribute of object Library"

```
DataAccessAuditMessage message =  
messageFactory.createDataAccessAuditMessage();  
message.setUser("John Smith");  
message.setObjectId("class=com.sap.example.Library(id=10)"); //a string  
that could uniquely identify the object  
message.addAttribute("address", true);  
message.log();
```

- Log a data modification event.

Use the `AuditLogMessage`, `AuditLogMessageFactory`, and `DataModificationAuditMessage` interfaces to log a data modification event.

## Sample Code

Logging for event "User John Smith modified the firstname attribute of object Person."

```
DataModificationAuditMessage message =  
messageFactory.createDataModificationAuditMessage();  
message.setUser("John Smith");  
message.setObjectId("class=com.sap.example.Person(id=5)"); //a string that  
could uniquely identify the object  
message.addAttribute("firstname", true);
```

- Log a security event.

Use the `AuditLogMessage`, `AuditLogMessageFactory`, and `SecurityEventAuditMessage` interfaces to log a security event.

## Sample Code

Logging for the event "User John Smith logged on from IP [10.20.30.40](http://10.20.30.40)."

```
SecurityEventAuditMessage message =  
messageFactory.createSecurityEventAuditMessage();  
message.setUser("John Smith");  
message.setIp("10.20.30.40");  
message.setData("Successful logon");  
message.log();
```

# Bind the Application to the Audit Log Service

## Procedure

1. Create a service instance.

```
xs create-service auditlog free <my-service-instance>
```

- Bind the service instance to the application.

- With the command line interface

```
xs bind-service <my-application> <my-service-instance>
```

- In manifest.yml

#### Sample Code

```
---
applications:
- name: java-auditlog-sample
  memory: 512M
  instances: 1
  path: target/java-auditlog-sample.war
  env:
    TARGET_RUNTIME: tomcat
  services:
    - <my-service-instance>
```

## View Audit Logs

### Context

To view the audit logs, you use an Audit Log UI available as part of the XS Advanced Admin tools.

### Related Information

[Audit Logs in Java Run Time \[page 615\]](#)

### 7.2.2.3.1 Audit Logs in Java Run Time

SAP HANA XS advanced Java run time provides an audit log API that enables applications and the run time to record audit logs.

### Audit Log Categories

- Configuration change log
- Data read access log
- Data modification log

- Security event log

## Utility Interfaces

- `AuditLogMessage`  
A general interface that applies to all categories and used to persists the created audit log message.
- `ConfigurationChangeAuditMessage`  
A child interface specific for the configuration change category and used to log changes in the configuration data.
- `DataAccessAuditMessage`  
A child interface specific for the data read access category and used to log successful and unsuccessful access to any sensitive personal data.
- `DataModificationAuditMessage`  
A child interface specific for the data modification category and used to log successful and unsuccessful modifications to any sensitive personal data.
- `SecurityEventAuditMessage`  
A child interface specific for the security event category and used to log all security-relevant events. These are events that may impact the confidentiality, integrity, and availability of the system such as server start or stop, failed logins, failed authorization checks, start of critical transactions, and changes of critical system parameters.
- `TransactionalAuditLogMessage`  
The transactional interface is applicable for the *Configuration Change* category and indicates whether the audit log event initiated in a previous step has been successful or not. You can use one of the following methods for such logging:
  - `logPrepare()`  
Indicate that the audit event is about to happen.
  - `logSuccess()`  
Indicate that the audit event has been successful.
  - `logFailure()`  
Indicate that the audit event has failed.

## Audit Log Message Format

The audit log message contains the following information:

Table 88:

Parameter	Description
<code>user</code>	The user that triggered the audit log event.
<code>objectId</code>	The unique ID of the object, which the action was performed on.

Parameter	Description
key	<ul style="list-style-type: none"> <li>The key of the data set.</li> <li>The type of the data access channel.</li> </ul>
name	<ul style="list-style-type: none"> <li>The heading name for the attribute that has been read or modified, either successfully or unsuccessfully</li> <li>The name of an attachment when the event is triggered by the download or display of some attachments.</li> </ul>
id	The ID of an attachment when the event is triggered by the download or display of some attachments.
ip	The source IP address that triggered the event during an external access.
data	The detailed data describing the security event
timestamp	This is the time at which the action was performed, not the time of the audit log creation.

## Related Information

[Configure an Application to Use the Audit Log Service \[page 611\]](#)

### 7.2.2.4 Configure Authentication and Authorization

You configure authentication and authorization by using the integrated container authentication provided with the Java build pack or the Spring security library.

#### Prerequisites

- You have secured your SAP HANA XS advanced run time.  
To configure authorizations for Java run time, you just need to set them for SAP HANA XS advanced. For more information, see *Setting up Security Artifacts* in *Related Information* below.

#### i Note

A role collection must have already been assigned to the user in SAP HANA. This is done through SAP HANA studio. For more information, see *Role Collections for XS Advanced Administrators* in the *SAP HANA Administrators Guide*.

- You have secured the application router.

For more information, see *Configure the XS Advanced Application Router* in *Related Information* below.

- You have created an instance of the XS User Account and Authentication (UAA) service. You must have created the `xsuaa` service instance and have bound it to the application for which you want to configure authentication. For more information about creating a service instance, see *Create an Instance of the xsuaa Service* and *Bind the xsuaa Service Instance to the XS Advanced Application* in *Related Information*.

## Context

XS advanced enables offline validation of the JSON Web Token (JWT) used for authentication purposes; it does not require an additional call to the User Account and Authorization (UAA) service. The trust for this offline validation is created by binding the `xsuaa` service instance to your application.

# Configure Integrated Container Authentication of the SAP Java Buildpack

## Context

The Java build pack includes an authentication method called `XSUAA`. This makes an offline validation of the received JWT token possible. The signature is validated using the verification key received from the service binding to the `xsuaa` service.

To enforce a check for the `$XSAPPNAME.Display` scope, proceed as follows:

## Procedure

1. Configure the use of the `XSUAA` authentication method.

This is done in the `login-config` section of the `web.xml` file.

Sample `web.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
    xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd
    version="3.0">
    <display-name>sample</display-name>
    <login-config>
        <auth-method>XSUAA</auth-method>
    </login-config>
</web-app>
```

2. Add a security constraint to a servlet.

```
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.*;
import javax.servlet.http.*;
/**
 * Servlet implementation class HomeServlet
 */
@WebServlet("/*")
@ServletSecurity(@HttpConstraint(rolesAllowed = { "Display" }))
public class HomeServlet extends HttpServlet {
    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
     response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
        response.getWriter().println("principal" +
request.getUserPrincipal());
    }
}
```

## Configure Spring Security

### Context

Applications using the Spring libraries can use the corresponding Spring security libraries. SAP HANA XS advanced model provides a module for offline validation of the received JSON Web Token (JWT). The signature is validated using the verification key received from the application's binding to the XS UAA service. To configure Spring security for your Java application, perform the following steps:

### Procedure

1. Specify the relevant libraries in your build manifest.

Sample pom.xml

```
<dependency>
    <groupId>com.sap.xs2.security</groupId>
    <artifactId>java-container-security</artifactId>
    <version>0.14.5</version>
</dependency>
```

2. Specify the required listeners in your web.xml.

Sample web.xml

```
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</
    listener-class>
</listener>
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring-security.xml</param-value>

```

```

</context-param>
<filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-
class>
</filter>
<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>

```

3. Configure the Spring beans by adding `spring-security.xml`.

Sample `spring-security.xml`

```

<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:oauth="http://www.springframework.org/schema/security/oauth2"
    xmlns:sec="http://www.springframework.org/schema/security" xmlns:xsi="http://
    www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://
    www.springframework.org/schema/security/oauth2      http://
    www.springframework.org/schema/security/spring-security-
    oauth2-1.0.xsd          http://www.springframework.org/schema/security
    http://www.springframework.org/schema/security/spring-
    security-3.2.xsd        http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-3.1.xsd">
    <!-- protect secure resource endpoints
    ===== -->
    <sec:http pattern="/**" create-session="never" entry-point-
ref="oauthAuthenticationEntryPoint" access-decision-manager-
ref="accessDecisionManager" authentication-manager-
ref="authenticationManager" use-expressions="true">
        <sec:anonymous enabled="false" />
        <sec:intercept-url pattern="/rest/addressbook/deletedata"
access="#oauth2.hasScope('${xs.appname}.Delete)" method="POST" />
        <sec:intercept-url pattern="/**" access="isAuthenticated()" 
method="GET" />
        <sec:custom-filter ref="resourceServerFilter"
before="PRE_AUTH_FILTER" />
        <sec:access-denied-handler ref="oauthAccessDeniedHandler" />
    </sec:http>
    <bean id="oauthAuthenticationEntryPoint"
class="org.springframework.security.oauth2.provider.error.OAuth2Authentication
EntryPoint" />
    <bean id="oauthWebExpressionHandler"
class="org.springframework.security.oauth2.provider.expression.OAuth2WebSecuri
tyExpressionHandler" />
    <bean id="accessDecisionManager"
class="org.springframework.security.access.vote.UnanimousBased">
        <constructor-arg>
            <list>
                <bean
class="org.springframework.security.web.access.expression.WebExpressionVoter">
                    <property name="expressionHandler"
ref="oauthWebExpressionHandler" />
                </bean>
                <bean
class="org.springframework.security.access.vote.AuthenticatedVoter" />
            </list>
        </constructor-arg>
    </bean>
    <sec:authentication-manager alias="authenticationManager" />
    <oauth:resource-server id="resourceServerFilter" resource-id="springsec"
token-services-ref="offlineTokenServices" />
    <bean id="offlineTokenServices"
class="com.sap_xs2.security.common.SAPOfflineTokenServices">
        <property name="verificationKey" value="${xs.uaa.verificationkey}" />
    
```

```

<property name="trustedClientId" value="${xs.uaa.clientid}" />
<property name="trustedIdentityZone" value="${xs.uaa.identityzone}" />
</bean>
<bean id="oauthAccessDeniedHandler"
class="org.springframework.security.oauth2.provider.error.OAuth2AccessDeniedHandler" />
<!-- define properties file
=====
<bean
class="com.sap.xs2.security.commons.SAPPropertyPlaceholderConfigurer">
    <property name="location" value="classpath:/application.properties" />
</bean>
</beans>
```

In this example, the most important line in the service context is highlighted in **bold**. The contents of this file specify which parts of the micro service are to be made secure. For example, an `HTTP POST` request to `/rest/addressbook/deletedata` requires the `Delete` scope.

- Configure the security properties of the service instance in `application.properties`.

Sample `application.properties` file of a Java application

```
# parameters of hello-world-java
xs.appname=java-hello-world
```

- Update your Java code.

The following code snippet shows that not only the requests are authenticated but also that the user principal is available.

```
UserInfo userInfo = SecurityContext.getUserInfo();
String name = userInfo.getLogonName();
String email = userInfo.getEmail();
String[] attribute = userInfo.getAttribute("my attribute");
boolean hasDeleteScope = userInfo.hasLocalScope("Delete");
```

## Related Information

[Setting Up Security Artifacts \[page 757\]](#)

[Configure the XS Advanced Application Router \[page 717\]](#)

[Create an Instance of the XS UAA Service \[page 773\]](#)

[Bind the XS UAA Service Instance to the XS Advanced Application \[page 776\]](#)

### 7.2.2.5 Configure a Database Connection

You can configure your application to use a database connection so that the application can persist its data. This configuration is applicable for Tomcat or TomEE run times.

## Related Information

[Configure a Database Connection for the Tomcat Run Time \[page 622\]](#)

[Configure a Database Connection for the TomEE Run Time \[page 623\]](#)

[Database Connection Configuration Details \[page 625\]](#)

[SAP HANA HDI Data Source \[page 627\]](#)

### 7.2.2.5.1 Configure a Database Connection for the Tomcat Run Time

#### Procedure

1. Create a `context.xml` file.

The `context.xml` file has to be inside the `META-INF/` directory of the application's WAR file and has to contain information about the data source to be used.

#### Example

Sample `context.xml`

```
<?xml version='1.0' encoding='utf-8'?>
<Context>
  <Resource name="jdbc/DefaultDB"
    auth="Container"
    type="javax.sql.DataSource"
    factory="com.sap.xs.jdbc.datasource.tomcat.TomcatDataSourceFactory"
    service="${service_name_for_DefaultDB}"/>
</Context>
```

2. Add the default keys and their values.

You need to include the data source information in `META-INF/sap_java_buildpack/config/resource_configuration.yml` of the WAR file.

#### Example

Defining a default service in `resource_configuration.yml`

```
---
tomcat/webapps/ROOT/META-INF/context.xml:
  service_name_for_DefaultDB: di-core-hdi
```

3. Add the key values to be updated.

You include the data source information to be updated in `manifest.yml`.

## Example

Defining a new service for the look-up of the data source

```
env:  
    JBP_CONFIG_RESOURCE_CONFIGURATION: "[ 'tomcat/webapps/ROOT/META-INF/  
context.xml' : { 'service_name_for_DefaultDB' : 'my-local-special-di-core-  
hdi' } ]"
```

As a result of this configuration, when the application starts, the factory named `com.sap.xs.jdbc.datasource.TomcatDataSourceFactory` takes the parameters bound for service `my-local-special-di-core-hdi` from the environment, creates a data source, and binds it under `jdbc/DefaultDB`. The application then uses the Java Naming and Directory Interface (JNDI) to look up how to connect with the database.

## Related Information

[Configure a Database Connection \[page 621\]](#)

[Database Connection Configuration Details \[page 625\]](#)

[SAP HANA HDI Data Source \[page 627\]](#)

## 7.2.2.5.2 Configure a Database Connection for the TomEE Run Time

### Procedure

1. Create a `resources.xml` file.
  - If the data source is to be used from a Web application, you have to create the file inside the `WEB-INF` directory.
  - If the data source is to be used from Enterprise JavaBeans (EJBs), you have to create the file inside the `META-INF` directory.

The `resources.xml` file has to be inside the application's WAR file and has to contain information about the data source to be used.

## Example

Sample `resources.xml`

```
<?xml version='1.0' encoding='utf-8'?>  
<resources>  
    <Resource id="jdbc/DefaultDB" provider="xs.openejb:XS Default JDBC  
Database" type="javax.sql.DataSource" >  
        service=${service_name_for_DefaultDB}  
    </Resource>  
</resources>
```

## 2. Add the default keys and their values.

You need to include the data source information in `META-INF/sap_java_buildpack/config/resource_configuration.yml` of the WAR file.

-  **Example**

Defining a default service in `resource_configuration.yml` for a Web application

```
---  
tomee/webapps/ROOT/WEB-INF/resources.xml:  
  service_name_for_DefaultDB: di-core-hdi
```

-  **Example**

Defining a default service in `resource_configuration.yml` for an EJB

```
---  
tomee/webapps/ROOT/META-INF/resources.xml:  
  service_name_for_DefaultDB: di-core-hdi
```

## 3. Add the key values to be updated.

You include the data source information to be updated in `manifest.yml`.

-  **Example**

Defining a new service for the look-up of the data source in a Web application

```
env:  
  TARGET_RUNTIME: tomee  
  JBP_CONFIG_RESOURCE_CONFIGURATION: "[  
    'tomee/webapps/ROOT/WEB-INF/resources.xml':  
      {'service_name_for_DefaultDB': 'my-local-special-di-core-hdi'}  
  ]"
```

-  **Example**

Defining a new service for the look-up of the data source in an EJB

```
env:  
  TARGET_RUNTIME: tomee  
  JBP_CONFIG_RESOURCE_CONFIGURATION: "[  
    'tomee/webapps/ROOT/META-INF/resources.xml':  
      {'service_name_for_DefaultDB': 'my-local-special-di-core-hdi'}  
  ]"
```

As a result of this configuration, when the application starts, the XS factory takes the parameters bound for service `my-local-special-di-core-hdi` from the environment, creates a data source, and binds it under `jdbc/DefaultDB`. The application then uses the Java Naming and Directory Interface (JNDI) to look up how to connect with the database.

## Related Information

[Configure a Database Connection \[page 621\]](#)

[Database Connection Configuration Details \[page 625\]](#)

[SAP HANA HDI Data Source \[page 627\]](#)

### 7.2.2.5.3 Database Connection Configuration Details

Define details of the database connection used by your Java application in XS advanced.

#### Configuration Files

To configure your XS advanced application to establish a connection to the SAP HANA database, you must specify details of the connection in a configuration file. For example, you must define the data source that the application will use to discover and look up data. The application then uses a Java Naming and Directory Interface (JNDI) to look up the specified data in the file.

The easiest way to define the required data source is to declare the keys for the data source in a resource file.

#### Example

For the Tomcat run time, you can create a `context.xml` file in the `META-INF` / directory with the following content:

```
<?xml version='1.0' encoding='utf-8'?>
<Context>
  <Resource name="jdbc/DefaultDB"
    auth="Container"
    type="javax.sql.DataSource"
    factory="com.sap.xs.jdbc.datasource.tomcat.TomcatDataSourceFactory"
    service="di-core-hdi"/>
</Context>
```

For the TomEE run time, you can create a `resources.xml` file in the `WEB-INF` / directory with the following content:

```
<?xml version='1.0' encoding='utf-8'?>
<resources>
  <Resource id="jdbc/DefaultDB" provider="xs.openejb:XS Default JDBC Database"
    type="javax.sql.DataSource" >
    service=di-core-hdi
  </Resource>
</resources>
```

The problem with this simple approach is that your `WAR` file cannot be signed, and any modifications can only be made in the `WAR` file. For this reason, it is recommended that you do not use the method in a production

environment; use modification settings in `resource_configuration.yml` and `manifest.yml` instead, as illustrated in the following examples:

The `resource_configuration.yml` is used for the default “key” settings.

### Example

Defining a default service in `resource_configuration.yml`

```
---  
tomcat/webapps/ROOT/META-INF/context.xml:  
  service_name_for_DefaultDB: di-core-hdi
```

Specifying a default name for a service is useful (for example, for automation purposes) only if you are sure there can be no conflict with other names. For this reason, it is recommended that you include a helpful and descriptive error message instead of a default value. That way the error message will be part of an exception triggered when the data source is initialized, which helps troubleshooting.

### Example

Sample `resource_configuration.yml`

```
---  
tomcat/webapps/ROOT/META-INF/context.xml:  
  service_name_for_DefaultDB: Specify the service name for Default DB in  
  manifest.yml via "JBP_CONFIG_RESOURCE_CONFIGURATION"..
```

## Placeholders

The generic mechanism `JBP_CONFIG_RESOURCE_CONFIGURATION` basically replaces the key values in the specified files. For this reason, if you use placeholders in the configuration files, it is important to ensure that you use unique names for the placeholders.

### Example

Sample `context.xml`

```
<?xml version='1.0' encoding='utf-8'?>  
<Context>  
  <Resource name="jdbc/DefaultDB"  
    auth="Container"  
    type="javax.sql.DataSource"  
    description="Datasource for general functionality"  
    factory="com.sap.xs.jdbc.datasource.tomcat.TomcatDataSourceFactory"  
    service="${service_name_for_DefaultDB}"  
    maxActive="\${max_Active_Connections_For_DefaultDB}" />  
  <Resource name="jdbc/DefaultXADB"  
    auth="Container"  
    type="javax.sql.XADatasource"  
    description="Datasource for functionality requiring more than one  
    transactional resource"  
    factory="com.sap.xs.jdbc.datasource.tomcat.TomcatDataSourceFactory"  
    service="\${service_name_for_DefaultXADB}"  
    maxActive="\${max_Active_Connections_For_DefaultXADB}" />
```

```
</Context>
```

The names of the defined placeholders are also used in the other resource files.

### Example

Sample `resource_configuration.yml`

```
---
```

```
tomcat/webapps/ROOT/META-INF/context.xml:
  service_name_for_DefaultDB: di-core-hdi
  max_Active_Connections_For_DefaultDB: 100
  service_name_for_DefaultDB: di-core-hdi-xa
  max_Active_Connections_For_DefaultXADB: 100
```

Sample `manifest.yml`

```
---
```

```
env:
  JBP_CONFIG_RESOURCE_CONFIGURATION: "['tomcat/webapps/ROOT/META-INF/
context.xml': {'service_name_for_DefaultDB': 'my-local-special-di-core-hdi' ,
'max_Active_Connections_For_DefaultDB' : '30',
'service_name_for_DefaultXADB' : 'my-local-special-xa-di-core-hdi' ,
'max_Active_Connections_For_DefaultXADB' : '20' }]"
```

## Related Information

[Configure a Database Connection \[page 621\]](#)

[SAP HANA HDI Data Source \[page 627\]](#)

### 7.2.2.5.4 SAP HANA HDI Data Source

To use a SAP HANA HDI container from your Java run time, you must create a service instance for the HDI container and bind the service instance to the Java application. To create a service instance, use the `xs create-service` command, as illustrated in the following example:

### Sample Code

```
xs create-service hana hdi-shared my-hdi-container
```

To bind the service instance to a Java application, specify the service instance in the Java application's deployment manifest file (`manifest.yml`).

### Sample Code

```
services:
  - my-hdi-container
```

Next, add the resource reference to the `web.xml` file, which must have the name of the service instance:

#### Sample Code

```
<resource-ref>
  <res-ref-name>jdbc/my-hdi-container</res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
</resource-ref>
```

Look up the data source in your code; you can find the reference to the data source in the following ways:

- Annotations

```
@Resource(name = "jdbc/my-hdi-container")
private DataSource ds;
```

- Java Naming and Directory Interface (JNDI) look-up

```
Context ctx = new InitialContext();
return (DataSource) ctx.lookup("java:comp/env/jdbc/my-hdi-container");
```

## 7.2.2.6 Debug an XS Advanced Java Application

You can use this procedure only if you can restart your Java application or the application has not yet been pushed.

### Procedure

- Add an environment variable to the Java application's manifest file (`manifest.yml`).

Open the `manifest.yml` file and add the `<JBP_CONFIG_JAVA_OPTS>` variable, as illustrated in the following example:

#### Example

```
---
applications:
- name: test-java
  memory: 256M
  instances: 1
  path: target/test-java.war
  env:
    TARGET_RUNTIME: tomcat
    JBP_CONFIG_JAVA_OPTS: [java_opts: '-
agentlib:jdwp=transport=dt_socket,address=8000,server=y,suspend=y']
```

### Note

With `<JBP_CONFIG_JAVA_OPTS>`, you set options to the JVM, which will activate the Java Debug Wire Protocol (JDWP) agent. You could change the desired options to your liking. For more information, about the available options, see `-agentlib:jdwp` and `-Xrunjdwp` sub-options in *Related Information* below.

2. Push the application.

To push the application to the XS advanced Java run-time, use the following command:

```
xs push -f manifest.yml
```

Before the application starts, it waits for a debugger to connect to port 8000.

3. Attach the debugger to the specified port.

### Caution

Your application will fail to start if you do not attach a debugger in time.

## Related Information

[-agentlib:jdwp and -Xrunjdwp sub-options](#) 

## 7.2.2.7 Enable and Use Java Data Services

Enable Java Data Services, a native Java client for using Core Data Services functionality in the XS advanced Java run time.

## Context

The Java Data Services (JDS) for XS advanced provides a set of tools that facilitate the consumption of Core Data Services (CDS) artifacts in Java applications. JDS enables you to map existing CDS data models to the corresponding Java Persistence API (JPA) 2.1 data models so that they can be consumed by Java applications. To use Java Data Services in XS advanced, perform the following actions:

## Procedure

1. Start the JDS tool using the following command in a command shell.

```
$:/jds> java -jar jds-import-1.0.1-full.jar help
```

2. Have a look at the tools provided in JDS.

The `help` option lists the top-level functions available.

#### Output Code

```
usage: java -jar importer-x.y.z.jar <command> <parameters>
Commands:
    help - displays this help text
    userguide - display the json userguide
    display - display available entities in namespace
    create - create Java classes for entities
    <config file>.json - process instructions in document (advanced)
```

#### ➔ Tip

Further information is displayed concerning how to set and use parameters.

## Related Information

[Java Data Services in XS Advanced \[page 630\]](#)

### 7.2.2.7.1 Java Data Services in XS Advanced

Java Data Services provide a native Java client for using Core Data Services functionality in the XS advanced Java run time container.

The Java Data Services (JDS) are a collection of tools which help to enable the consumption of Core Data Services (CDS) artifacts in Java applications. JDS maps existing CDS data models to corresponding Java Persistence API (JPA) 2.1 data models that can then be consumed by Java applications, for example, using EclipseLink or Hibernate. The JDS tools are available by default from the command line, as illustrated in the following example:

#### Sample Code

```
java -jar jds-import-1.0.1-full.jar help
```

In addition to providing all the parameters of `jds-tool` on the command line or inside the `jds.ini` file it is also possible to use a JavaScript Object Notation (JSON) document instead. For more advanced features such as providing information about back links (one-to-many associations) or “n:m” associations, a JSON document is required. The meaning of all possible flags or parameters is equal to the command-line options.

#### Note

When using a JSON document, the contents of any `jds.ini` properties file are ignored.

The following example shows the basic structure of a JSON document that describes the parameters available for the JDS tools:

#### Code Syntax

```
{  
  "connection" : {  
    /* connection parameter */  
  },  
  "<command>" : {  
    /* either create or display - as with the commandline tool */  
  },  
  "contexts" : {  
    /* optional additional metadata for the "create" command.  
     * used to provide information for  
     * - backlinks  
     * - m:n connections  
     * and list all entities if you don't want to get JPA classes  
     * for all CDS entities in the namespace */  
  }  
}
```

The generated Java source files can be copied into your Java application. Remember to regenerate the files if your underlying CDS data model changes.

#### Caution

Note that JDS is for use in consuming existing CDS data models; it is strongly recommended **not** to use drop-create-tables with JDS.

## Connection Parameters

For the “connection” object, all properties except `proxy` are mandatory; if the mandatory properties are omitted, they and their assigned value are queried by the JDS tool.

#### Code Syntax

```
"connection" : {  
  "password": "<yourPassword>",  
  "schema": "<yourSchema>",  
  "host": "<your host>:<your port>",  
  "user": "<yourUser>",  
  "proxy": "<yourproxy>"  
}
```

## Commands

For the "command" object, some parameters (for example, `namespace`) are mandatory and some optional, as illustrated in the following example:

### Code Syntax

```
"create": {  
    "namespace": "<namespace of CDS entities>", //mandatory  
    "package": "<packageName>", // optional  
    "targetDirectory": "<targetDirectory>", //optional  
    "flags": ["continueOnError", "simulate", "overwrite",  
              "includeSchema", "listedEntitiesOnly"], //optional  
    "jpaFormat": { //optional  
        "embeddedKeySuffix": "<MyPK>",  
        "indentationDepth": 8,  
        "quotationCharacter": "\\\\""  
    }  
}  
"display": {  
    "namespace": "<your namespace>",  
    "flags": ["withoutDependencies"] // currently only one flag  
}
```

You can use "contexts" to add additional metadata to `create` objects. This might be necessary if there is no support for one-to-many (back link) and many-to-many associations. The following example shows a simple way to use contexts to supply additional metadata:

### Code Syntax

```
"contexts" : {  
    "post": {  
        "Comments": {  
            "target": "comment",  
            "viaBacklink": "Post"  
        },  
        "Tags": {  
            "target": "tag",  
            "viaEntity": "tags_by_post",  
            "sourceField": "Tag.tid",  
            "targetField": "Post.pid"  
        }  
    },  
    "comment": {},  
    "tag": {} // no need for reference to 'post'  
}
```

### ⚠ Restriction

The "contexts" object in combination with the flag "listedEntitiesOnly" can also be used to create JPA classes only for a limited selection of entities instead of the full namespace.

## Command line Properties

The most commonly used command-line arguments, for example, connection properties, can be placed in a `jds.properties` file, as illustrated in the following example:

### Code Syntax

```
#---JDS connection properties---
password=<secretPassword>
schema=<mySchema>
host=<myHost>:<myPort>
user=<myUserName>
```

## CDS Support

This release of Java Data Services supports the following CDS elements:

- CDS entities
- CDS types
- CDS views
- CDS associations (one-to-one, only)

## Related Information

[Enable and Use Java Data Services \[page 629\]](#)

[Download and Install JavaScript Data Services \[page 562\]](#)

### 7.2.2.8 Run a Java Main Application

You can create a Java application that starts its own run time.

## Prerequisites

You are not using any resource configurations.

The resource configurations needed for the database connection are not applicable for the Java Main application. For more information about database connections, see *Related Information*.

## Context

The SAP Java buildpack provides an Apache Tomcat or an Apache TomEE run time. However, you can create a Java Main application to use other frameworks and Java run times such as *Spring Boot*, *Jetty*, *Undertow*, or *Netty*.

### ⚠ Caution

The Tomcat and TomEE run times provided by the SAP Java buildpack have the following functionality:

- Solution manager end-to-end tracing
- Java EE web container security  
Java Main applications can only use Spring Security.
- Management of exit codes  
The exit codes are used by XS advanced to monitor started processes. If a process exits with a *O* exit code, it will not be started automatically.

When you create an application to use its own run time, you have to take into consideration that the above functionality might not be available.

## Procedure

1. Make sure your built JAR archive is configured properly.

Regardless of the tool you use to build your Java application, you have to make sure that the following tasks are performed:

- a. You have built a JAR archive.
- b. You have specified a main class in the `META-INF/MANIFEST.MF` file of the JAR archive.

### Sample Code

#### Sample `MANIFEST.MF`

```
Manifest-Version: 1.0
Archiver-Version: Plexus Archiver
Built-By: p123456
Created-By: Apache Maven 3.3.3
Build-Jdk: 1.8.0_45
Main-Class: com.companya.xs.java.main.Controller
```

- c. You have packaged all your dependent libraries in the JAR file, also known as creating an uber JAR or a fat JAR.

If you are using Maven as your build tool, you can use the [\*maven-shade-plugin\*](#) to perform the above tasks.

### Sample Code

An example configuration for the [\*maven-shade-plugin\*](#)

```
<build>
  <finalName>java-main-sample</finalName>
```

```

<plugins>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-shade-plugin</artifactId>
    <version>2.4.3</version>
    <configuration>
      <createDependencyReducedPom>false</createDependencyReducedPom>
      <filters>
        <filter>
          <artifact>*:*</artifact>
          <excludes>
            <exclude>META-INF/*.SF</exclude>
            <exclude>META-INF/*.DSA</exclude>
            <exclude>META-INF/*.RSA</exclude>
          </excludes>
        </filter>
      </filters>
    </configuration>
    <executions>
      <execution>
        <phase>package</phase>
        <goals>
          <goal>shade</goal>
        </goals>
        <configuration>
          <transformers>
            <transformer
implementation="org.apache.maven.plugins.shade.resource.ServicesResourceTran
sformer" />
            <transformer
implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTran
sformer">
              <manifestEntries>
                <Main-Class>com.sap.xs.java.main.Controller</Main-Class>
              </manifestEntries>
            </transformer>
          </transformers>
        </configuration>
      </execution>
    </executions>
  </plugin>
</plugins>
</build>

```

## 2. Configure manifest.yml.

To be able to push the Java Main application, you need to specify the path to the .jar archive in the manifest.yml file.

### Sample Code

#### Sample manifest.yml

```

---
applications:
- name: java-main
  memory: 128M
  path: ./target/java-main-sample.jar
  instances: 1
  services:
    - java_main_hana

```

## 3. Push the application to XS advanced run time.

For that purpose, use the `xs push` command.

### Example

Donna Moore would like to create a Java Main application to use its own run time. For that purpose she performs the following steps:

1. She downloads her `sample_main` application from the company's server and extracts it into a local folder.
2. She navigates in the command line tool to the `sample_main` directory of the application and builds it. Because Maven is installed on her computer, she can perform this operation with the command line `mvn clean install`.
3. After the build is successful, she checks that her folder contains a `sample_main.jar` file, which is located in the `target` directory.
4. She opens the `sample_main.jar` file and checks that the META-INF/MANIFEST.MF file contains the `Main-Class` header with a value the name of the main class.
5. She adds the path to the JAR file in the `manifest.yml` file.  
She needs to do that to make sure that the application can be pushed to the XS advanced run time. For that purpose, she adds the following line in the manifest file:

```
path: ./target/sample_main.jar
```

6. Finally, she pushes the Java Main application with the following command line:

```
xs push sample_main
```

## Related Information

[Configure a Database Connection \[page 621\]](#)

## 7.3 Custom Run-Time Environments in SAP HANA XS Advanced

Create a custom run-time environment for an application written in a language that is not supported by default by XS advanced.

SAP HANA XS advanced includes a list of standard predefined run-time environments (for example Java, Node.js, Tomcat, TomEE) that can be downloaded from a run-time “store” included with XS advanced during the compilation phase of the corresponding buildpack. These standard default run-time environments are primarily intended for use with the buildpacks included by default with XS advanced.

However, you can also create and run a custom run-time environment of your own in XS advanced, so that you can deploy applications written using languages such as Python or PHP, which are not supported by any of the default run-time environments included in the XS advanced run-time store.

## Caution

SAP does not provide support for custom language, buildpack, or run-time scenarios.

Applications deployed to a custom run-time environment in XS advanced do not have automatic access to (or use of) some important features that are built into and supported by the XS advanced framework, including (but not limited to): authentication and security, logging and auditing, and connections to the database. If you deploy an application to a custom run-time environment in XS advanced, you must configure these components manually for the custom application.

If you want to deploy an application written in a language for which XS advanced does not include a corresponding run-time environment (for example, Python or PHP), you must provide a suitable run-time “buildpack”. Buildpacks are a convenient way of packaging framework, dependencies, and run-time support for an application at deployment time. The buildpack defines what happens to your application when it is deployed and, in addition, how it is executed after deployment.

Buildpacks are stored in the central blob store for XS advanced along with run-time environments and applications. To upload a buildpack to the central XS advanced blob store, you can use the XS command-line interface (CLI), for example, the `xs create-buildpack` command.

## Restriction

Access to the blob store is restricted to users with administrator rights.

## Related Information

[Create a Python Buildpack for XS Advanced \[page 640\]](#)

[Custom Buildpacks in XS Advanced \[page 642\]](#)

[XS CLI: Buildpacks \[page 904\]](#)

### 7.3.1 Best Practices for Deploying Applications in a Custom Run-Time Environment

A list of things to bear in mind when coding an application that you plan to deploy in a custom XS advanced run-time environment.

## Overview

If you plan to develop an application that will be deployed to a non-standard run-time environment in SAP HANA XS advanced, for example, Python or PHP, it is important to develop a strategy that facilitates the creation (or migration) and deployment of the application to the custom run-time environment. For example, application logic that cannot be easily ported to one of the supported languages (Node.js, XS JavaScript, Java) should be contained in its own module.

You also need to understand the nature of the custom language you want to use and support:

- Compiled language (for example, C++, Swift)
- Interpreted language (for example, JavaScript, PHP, Perl)
- Runs in a Virtual Machine (for example, Java)
- Requires run-time support (for example, ASP.NET)

The primary function of your custom application module is to service HTTP requests that come exclusively from the application router module. The work that these requests require should be a natural fit for the programming language that the buildpack is supporting. If your module needs to access other services, for example, a data store, it can find details required for connecting to the service by inspecting the environment that the deployment process provides.

#### ➔ Tip

Keep in mind that your application module is executing in isolation; the primary interface to the outside world is through the HTTP server interface it provides.

## Deployment Environment

It is important to understand how the custom application will be deployed, for example, which parts of the application need to scale or be able to respond to a fail over. Since details of the deployment environment are often only available at run time, consider the following questions and points:

- External dependencies  
Which, if any, external services are required?
- Host names and/or ports  
Each module must be identified uniquely in the system. Typically, this is done by either assigning a port or a host name to the module. Since it is not always possible to obtain details of the deployed environment before deployment (or if details of the deployed environment change), it is essential to avoid hard-coding any host names or ports in your application; all ports and host names must be read from the environment.

#### i Note

In a multi-tenant PaaS environment (for example, SAP Cloud Platform) where many customers have multiple accounts and many deployed MTAs, host names are the only way to insure uniqueness.

- Configuration details  
Since configuration information is provided as run-time environment variables, the application-programming language must be able to read the run-time environment and parse JSON in order to obtain this configuration information.
- Security and encryption  
The application language must also include the libraries and code required to provide an HTTPS server, which can be used to respond to requests from the application router. In addition, the logic provided must be able to handle security certificates, encryption, and uncompressing data payloads.  
The application language must also be able to decrypt, parse, and verify JSON Web Tokens (JWT), for example, so that an application can refuse to respond to any request that does not provide a JWT in the request header (or provides an invalid one).  
If the custom application needs to communicate with other modules, it might be necessary to provide an HTTPS client, as well.

- Environment variables  
Your application module must be able to read and parse execution environment variables. The application-deployment process initializes the environment in such a way as to make available any required information so that the module can make connections to other services and interact with them. For example, JDBC is primarily used to connect to the data store, but other services are also available, for example, the job scheduler. An application module can also use HTTP to communicate with other modules, however, it will need to pass on the JWT in the request header, if necessary, so that the receiving module can also perform the same verification checks. If an application module needs to interact with an external service, it can perform outbound HTTP connections as well.
- Logging and tracing  
An application module does not usually have access to a writeable file system, so it is not recommended to assume that one will be available. This is especially important in a multi-tenant environment where your applications are running inside your own account. Often logging and (or) tracing service is provided by the system, so an application module should at least be able to log its activity, for example, to record when it encounters conditions that produce a warning or an error.

## Best Practices

To understand which components and dependencies are required to ensure the successful deployment of applications to a custom run-time environment in XS advanced, for example, Python, or PHP, bear in mind the following important points:

- Task focus  
Try and organize your custom application's modules into single (or related) tasks that require little or no communication with other modules. Choose a programming language that is best suited for the selected tasks.
- Portability  
Keep deployment and portability in mind as you develop. Since it is often not possible to know if the target system for deployment will be a dedicated system configured for port assignment or a multi-tenant system configured to use host names to distinguish module identity, it is essential to ensure that any configuration details and settings data is held in deployment descriptors so that it can be read dynamically during module start up.
- Database access  
Try to confine access to the persistence store (database) to one module and ensure this dedicated module fields all database requests from other modules, if needed. This keeps the data access unified in one place. Since these modules are all running in the same process space, the overhead is minimal anyway.
- Target deployment environment  
For compiled languages (for example, C++ or Swift) or languages that have dependencies to specific static or shared libraries, try to develop your custom application in an environment that, in configuration terms, is as close as possible to the target deploy environment.

### ➔ Tip

This is less important for interpreted languages (for example, PHP or Perl) or languages such as Java that run in a Virtual Machine (VM).

- Testing and debugging

If you have full control of the deployed environment (for example, in an on-premise scenario), consider setting up an HTTP server and a language run-time environment so that you can test logic and debug locally.

- Build/Deployment environment  
If possible, re-create as closely as possible a local environment for the language you are working in.
- Mock data/services  
Code your module in a way that it can be tested on the command line with mock data representing the services the module will eventually connect to. In this way, you can iterate quickly and focus on the logic of your module and avoid the time-consuming edit-deploy-test cycle.
- Routing and authentication  
Test each module **without** routing or any requirement for User Account and Authentication (UAA) services, for example, by using mock data that is provided directly in the target environment. If possible perform the tests locally and work out all the logical bugs (parsing JSON, string manipulation, iterating over data, and so on). Only after successful tests for the individual modules, should you consider connecting them to other modules, at which point you can set up the MTA's deployment descriptor (`mtad.yaml`) to configure and coordinate inter-module dependencies.
- Buildpacks  
When developing buildpacks, use `chroot` to help resolve any dependencies between environment and libraries. The Unix/Linux `ldd` command can be used to display a list of dependent libraries for a particular binary.
- Deployment target  
Test deployment of the custom application in multiple spaces and know where to find deployment logs and how to read and interpret them.

## Related Information

[Custom Run-Time Environments in SAP HANA XS Advanced \[page 636\]](#)

[Create a Python Buildpack for XS Advanced \[page 640\]](#)

### 7.3.2 Create a Python Buildpack for XS Advanced

Create a simple buildpack to compile a run-time environment for your Python application.

## Prerequisites

- You have installed the XS command-line client on your development machine.  
The XS CLI client tools are installed by default on the SAP HANA server. However, you must download and install the client tools locally, if you want to connect to SAP HANA from your local machine. The XS CLI client tools (`xs.onpremise.runtime.client_<platform>-<version>.zip`) can be downloaded from the SAP HANA server, from the installation DVD, or from the SAP support portal. For more information about the SAP support portal, see the link to SAP note 2242468 in the related links below.

- SAP Node.js packages  
The archive (`xs_javascript-<version>-bundle.tar.gz`) is available on the SAP HANA media and the SAP Marketplace (SMP); it includes a selection of Node.js packages developed by SAP for use with the Node.js applications running in the XS advanced run time.

➔ Tip

Alternatively, you can configure NPM to resolve SAP-related NPM dependencies using the SAP public NPM Registry (at [npm.sap.com](https://npm.sap.com)).

- `XS_ADMIN` privileges are required to install buildpacks in the SAP HANA XS advanced run-time environment.

## Context

The example described here shows the basic steps required to put together a buildpack that enables you to deploy a Python application to the XS advanced run-time environment.

⚠ Restriction

The information included in this example is provided for illustration purposes only; it is intended to help you put together your own buildpack for use in your own custom environment.

## Procedure

1. Create a package for your Python run-time.

In this example, we use a “portable” Python run-time environment that is compatible with the operating system where XS advanced is running. This Python run time can be used immediately after being unpacked.

ℹ Note

The packed run time is copied to a directory inside the buildpack (`runtime/PYTHON.TGZ`).

2. Detect the run-time required for the pushed application.

Implement `bin/detect` to search the run-time archive for a file with the name “`server.py`” and returns code “0” if successful. If `server.py`, the content in the run-time archive can be deployed to the specified (packaged) run-time.

3. Compile the run-time required for the pushed application.

Implement `bin/compile` to extract the Python run time (located in `runtime/PYTHON.TGZ`) included in the custom buildpack.

ℹ Note

The extraction process copies all content to the `work/` directory.

- Start the run-time required for the pushed application.

Implement `bin/release` to invoke the `server.py` file in the Python run time that was unpacked in the previous (compile) step. The release step uses the following command:

```
python server.py/$<VCAP_APP_PORT>
```

Where `<VCAP_APP_PORT>` is the port number where XS advanced expects the custom Python application to run. This port will be mapped to the URL assigned to the application in XS advanced.

- Upload the custom buildpack to the XS advanced run time environment.

#### **Restriction**

`XS_ADMIN` privileges are required to install buildpacks in the SAP HANA XS advanced run-time environment.

In a command shell, use the XS CLI to create the buildpack, as illustrated in the following example:

```
xs create-buildpack <myBuildPack> -p <path>/python-buildpack 20
```

#### **Tip**

The `<path>` specifies the location to store the built buildpack, and the integer “20” determines the build-pack sorting priority; buildpacks are sorted from lowest to highest. This is the order in which the detect script asks the available build-packs if they can deploy the specified application.

- Deploy the application for which you created the custom run-time.

In a command shell, use the XS CLI's push command to deploy the Python application to the custom run time in XS advanced, as illustrated in the following example:

```
xs push <myPythonApp> -p <path>/myPythonApp
```

- Test the application, for example, in a Web browser.

## Related Information

[Custom Buildpacks in XS Advanced \[page 642\]](#)

[XS CLI: Run-Time Environments \[page 908\]](#)

### 7.3.2.1 Custom Buildpacks in XS Advanced

Custom buildpacks define the conditions for the deployment of an application to a custom run-time environment in XS advanced.

buildpacks are a convenient way of configuring the packaging framework and run-time support for an application. The buildpack defines what happens to your application when deployed in XS advanced and, in addition, how it is executed after deployment.

SAP HANA XS advanced includes a list of standard predefined run-time environments (for example Java, Node.js, Tomcat) that can be downloaded from a run-time “store” during the compilation phase of the

corresponding buildpack. These standard run-time environments are primarily intended for use with the buildpacks included by default with XS advanced. However, you can also create and run a custom run-time environment in XS advanced, so that you can deploy applications written using languages such as Python or PHP, which are not supported by any of the default run-time environments included with XS advanced.

### Note

Applications deployed to a custom run-time environment in XS advanced cannot by default use some of the integrated features built into and supported by the XS advanced framework, including (but not limited to): authentication and security, logging and auditing, and connections to the database, all of which would need to be configured manually for the custom application. This would all need to be implemented in the specific programming language and (or) supplied by the run-time environment provided by the buildpack.

If you want to deploy an application written in a language for which XS advanced does not include a corresponding run-time environment (for example, Python or PHP), you must provide a suitable run-time buildpack. A simple buildpack contains the following scripts, which are typically located in the directory `/bin/`.

- Build-pack detection  
The `bin/detect` script is used to find a suitable buildpack to use to deploy an application.
- Compilation  
The `bin/compile` script assembles all the components that are required to deploy and run the application in the custom run-time environment.
- Deployment and execution  
The `bin/release` script is used to provide metadata for XS advanced; the metadata describes how and where the deployed application should be executed.

## Buildpack Detection

The `bin/detect` script is used to find a suitable buildpack to use to deploy an application. If more than one buildpack is available, XS advanced uses a detection priority to find the appropriate buildpack, where priority 1 (one) is the highest priority and indicates the first buildpack to check for suitability. The `detect` script requires one argument that specifies the location of the build directory for the custom application. The build directory is the location to which the pushed (application) content is copied by the `xs push` command during deployment. If the `detect` script finds a suitable buildpack for the specified application, it returns the exit code “0” (success). If multiple buildpacks are installed for a particular language and a buildpack with a lower priority is desired, the required buildpack can be explicitly identified by name during application deployment, as illustrated in the following command:

### Sample Code

```
xs push myAPP -b <BUILDPACK_NAME>
```

### Note

If the returned exit code is not “0”, XS advanced attempts to find and use other buildpacks by calling their corresponding `bin/detect` scripts.

You can find out what buildpacks are installed in your deployment system and what priority is assigned to each buildpack during the detection phase by running the `xs buildpacks` command, as illustrated in the following example:

### Output Code

```
xs buildpacks
Getting buildpacks...
buildpacks      version      position      enabled      locked
-----
sap_java_buildpack    <unknown>    1            true
nodejs_buildpack_op   3.0.0        2            true
python_buildpack      1.1.0        9            true
phpbp_buildpack       1.0.0        10           true
```

## Buildpack Compilation

The `bin/compile` script builds the droplet run by the execution agent; the droplet contains all the components that are required to deploy and run the application in the custom run-time environment. The `compile` script requires a value for the following arguments:

- The build location (a directory path/name) for the application
- The location of the cache directory that the buildpack uses for temporary storage during the build process

### i Note

The platform provides the flexibility required to handle the execution of architecture-dependent applications.

If the architecture where the buildpack compiles the droplet is the same as the architecture where the instances run, you can configure the custom buildpack to detect the architecture during compile phase and do the compilation accordingly. For example, the buildpack could take an architecture-specific run time that is included a “local” archive (the archive you are putting together in this process). The buildpack could also fetch the required run time from an external repository. It is also possible for the buildpack to assemble different application binaries for different architectures. If necessary, you can configure the buildpack to compile the application sources, too.

## Buildpack Deployment and Execution

The `bin/release` script is used to provide metadata for XS advanced; the metadata describes how the deployed application should be executed. The `release` script requires a value for the build location of the application (which was prepared in the compile step). The content of the file generated by the `release` script

be formatted according to YAML rules and, in addition, contain at least the information illustrated in the following example:

### Sample Code

#### Buildpack Application Run-time Metadata

```
config_vars:  
  name: <value>  
default_process_types:  
  web: <commandLine>
```

- `config_vars`  
A list of environment variables that will be set in the run-time environment where the application is deployed
- `<command>`  
The command used to start the deployed application; the command is run in the `work` directory prepared in the buildpack's compile phase.

## Related Information

[Custom Run-Time Environments in SAP HANA XS Advanced \[page 636\]](#)

[Create a Python Buildpack for XS Advanced \[page 640\]](#)

### 7.3.3 Create a PHP Buildpack for XS Advanced

A high-level overview of the steps required to build a custom buildpack for use in deploying a custom application (for example PHP) to the XS advanced run-time environment.

## Prerequisites

- This example assumes that the target deployment environment is SUSE 12.3 or similar.
- The project directory in this example is called “`buildpacks`”. The `buildpack` contains a subdirectory called `rootdir`. The `rootdir` directory contains the files associated with the use of the `chroot` command (for example, to determine build and library dependencies).

## Context

The run-time environment is like an isolated file system and an execution environment that hosts the components of the buildpack. The application files deployed are added to the environment that the buildpack

provides, and the instructions in the buildpack's compile script are run. In this example, we use PHP as the programming language.

### **Restriction**

The information included in this example is provided for illustration purposes only; it is intended to help you put together your own buildpack for use in your own custom environment.

## **Procedure**

1. Set up the new buildpack components.

Have a look at an existing (working) buildpack to get an example of the detect/compile/release scripts that you'll need to modify to accommodate your custom language. The existing buildpack will also provide much of the file-system files that the deployment run-time environment supplies.

#### **Note**

A buildpack typically includes a minimal set of system files. For this reason, it is highly likely that you will need to add any specific shared libraries that are required to support the execution of your programming language.

2. Build the executables.

If your language is a compiled language, you will either need to build the executables on your local development machine or provide a script to build them on the target machine during application deployment (for example, using the buildpack's "compile" script). Since PHP is an interpreted language, the module's logic is expressed in various PHP scripts that are included in the main "execution" script (`index.php`) that is executed by the buildpack during application deployment. The most important component that our buildpack needs to provide is a functional PHP interpreter.

3. Build the PHP interpreter.

It is possible to try to use a binary version of the PHP interpreter, but you will need to ensure that all the required shared libraries are available to it.

Depending on the language, you may have to implement its packager as well. For example, both Perl and Node.js have a dedicated packager you can use. PHP, however, uses the concept of modules that provide various services or enable various libraries. When building the PHP interpreter for the sample buildpack, it makes sense to start by trying to build the interpreter as statically as possible and only include those features that are needed.

#### **Note**

This involves getting the source code from PHP.net and compiling it with the desired features enabled.

4. Test the creation of the custom buildpack.

If possible, try to create the custom buildpack on the target server environment, where the build environment is as close as possible to the deployed run-time environment.

## Note

If you log in to the run-time system remotely, your login shell has access to the remote system's entire environment including the system <path> that is used for locating files and libraries. Bear in mind that files and libraries found in the build environment (because they are in locations configured in the system <path>) might not necessarily be found in the "real" but more isolated context of the buildpack running in the XS advanced run time. Dependencies to shared libraries that work in the target server environment might not work when you execute the buildpack.

In Unix and Linux environments, you can use the `chroot` command to simulate the isolated execution environment of the buildpack. To ensure you have all the components required to create your PHP buildpack, perform the following steps:

- Check the version of the PHP binary available in the `chroot` environment.

```
sudo chroot rootdir  
cd bin/php/bin  
.php -v  
  
PHP 5.6.22 (cli) (built: Jun 7 2016 21:43:46)  
Copyright (c) 1997-2016 The PHP Group  
Zend Engine v2.6.0, Copyright (c) 1998-2016 Zend Technologies
```

The output of the `php -v` command should provide version information only; if any error messages are displayed indicating dependencies to missing shared libraries, you will need to investigate and resolve the errors as described in the following steps.

- Exit the `chroot` shell.
- Run the `ldd` command on the PHP binary to list any shared dependencies, as illustrated in the following example:

```
cd rootdir/bin/php/bin/php  
ldd ./php  
  
linux-vdso.so.1 (0x00007ffe3b3ca000)  
 libcrypt.so.1 => /lib64/libcrypt.so.1 (0x00007f5c23302000)  
 libresolv.so.2 => /lib64/libresolv.so.2 (0x00007f5c230eb000)  
 libpng12.so.0 => /usr/lib64/libpng12.so.0 (0x00007f5c22ec0000)  
 libz.so.1 => /lib64/libz.so.1 (0x00007f5c22caa000)  
 libcurl.so.4 => /usr/lib64/libcurl.so.4 (0x00007f5c22a44000)  
 [...]
```

- Copy the missing shared libraries into the correct location in the `chroot` buildpack environment, for example, `/usr/lib64`.

```
cp /usr/lib64/libpng12.so.0 rootdir/usr/lib64
```

- Check the PHP binary again
- Package the directory structure containing the components required for your buildpack.

Package your directory in such a way that, on deployment, the directory contents are extracted into the target run-time environment based on the instructions in the buildpack's `compile` script. You can then install the buildpack, for example, using the `xs create-buildpack` command.

### Sample Code

```
cd buildpacks/rootdir  
tar czvf ../php-buildpack/runtime/PHP.TGZ .  
cd buildpacks
```

6. Install the custom buildpack in XS advanced.

You can install the buildpack using the following command:

```
xs create-buildpack <myBuildpackName> <Path-to-BuildPack> <position> --enable
```

For example, to create a buildpack called `phpbp` using a package called `php-buildpack.zip` with position “10” and make the buildpack available, use the following command:

```
xs create-buildpack phpbp php-buildpack.zip 10 --enable
```

### Tip

If multiple buildpacks are available, they are sorted for detection purposes according to `<position>` (priority), where position 1 (one) is the highest priority. A buildpack with position 1 (one) is inspected first during any detection process.

### Output Code

```
xs buildpacks  
Getting buildpacks...  


| buildpacks          | version   | position | enabled | locked |
|---------------------|-----------|----------|---------|--------|
| sap_java_buildpack  | <unknown> | 1        | true    |        |
| nodejs_buildpack_op | 3.0.0     | 2        | true    |        |
| phpbp               | 1.0.0     | 10       | true    |        |


```

### Tip

If a buildpack with the same name already exists, you can use the `xs update-buildpack` command to modify and overwrite it, for example:

### Sample Code

```
xs update-buildpack phpbp -p php-buildpack
```

7. Create and push PHP application modules to the new PHP run-time environment you have created in XS advanced.

When the PHP buildpack is installed in XS advanced, you can start creating PHP modules and pushing them to the deployed environment. During the deployment operation, your `index.php` file is detected and the PHP buildpack is used for your run-time environment.

### Sample Code

```
xs push phpapp -p php-test
```

### Note

phpapp is the name of the application to deploy and -p php-test specifies the location of the (ZIP) archive or directory containing the application to deploy.

## Related Information

[Custom Run-Time Environments in SAP HANA XS Advanced \[page 636\]](#)

[Best Practices for Deploying Applications in a Custom Run-Time Environment \[page 637\]](#)

[Create a Python Buildpack for XS Advanced \[page 640\]](#)

[XS CLI: Application Management \[page 852\]](#)

### 7.3.3.1 Buildpack Scripts for PHP Run-Time Environment in XS Advanced

Example scripts for a custom PHP buildpack.

This topic provides some simple examples of the three basic scripts required by a buildpack (in this case PHP) for use when deploying PHP application modules to the XS advanced run-time environment:

- [bin/compile \[page 649\]](#)
- [bin/detect \[page 650\]](#)
- [bin/release \[page 650\]](#)

### Note

The scripts here are provided for illustration purposes only; you might need to adjust them to suit your specific requirements.

## bin/compile

The following code snippets illustrates a simple `compile` script for use in a PHP buildpack:

### Sample Code

Compile Script for a PHP Buildpack

```
#!/bin/bash
```

```
# bin/compile <build-dir> <cache-dir>
BUILD_DIR=$1
BIN_DIR=`dirname $0` 
BUILD_PACK_DIR=`dirname ${BIN_DIR}` 
echo "Extracting PHP into..." 
echo ${BUILD_DIR} 
cd ${BUILD_DIR} 
tar xzvf ${BIN_DIR}/../runtime/PHP.TGZ 
echo "Extracting PHP Done."
```

## bin/detect

The following code snippets illustrates a simple `detect` script for use in a PHP buildpack:

### Sample Code

Detect Script for a PHP Buildpack

```
bin/detect
#!/bin/bash
# bin/detect <build-dir>
if [ -f $1/index.php ];
    then exit 0
fi
exit 1
```

## bin/release

The following code snippets illustrates a simple `release` script for use in a PHP buildpack:

### Sample Code

Release Script for a PHP Buildpack

```
#!/bin/bash
# bin/release
cat <<EOF
---
config_vars:
addons:
default_process_types:
    web: ./bin/php/bin/php -S 127.0.0.1:$VCAP_APP_PORT
EOF
```

## Related Information

[Create a PHP Buildpack for XS Advanced \[page 645\]](#)

## 7.3.4 Available Build Packs

Display a list of all available build packs in your XS advanced system.

The following table shows which build packs are available for use in XS advanced, both on-premise and in the SAP Cloud Platform:

➔ Tip

You can create your own build packs for use on-premise, for example, for Python or PHP.

To display the build packs available in your installed system, use the `[xs | cf] buildpacks` command in an XS CLI shell, as illustrated in the following example. The command prefix you use depends on the installed platform, for example, on-premise (`xs`) or Cloud Foundry (`cf`).

```
xs buildpacks
Getting buildpacks...
buildpacks      version   position  enabled  locked
-----
sap_java_buildpack  1.4.1     1        true     false
sap_nodejs_buildpack 3.2.3     2        true     false
```

The following table lists the standard build packs available with XS advanced on-premise and in the Cloud:

Table 89: Build Pack Availability in XS Advanced

Build Pack	On-Premise	Cloud Foundry
sap_java_buildpack	✓	✓
sap_nodejs_buildpack	✓	-
nodejs_buildpack*	-	✓
staticfile_buildpack*	-	✓
ruby_buildpack*	-	✓
go_buildpack*	-	✓
python_buildpack*	-	✓
php_buildpack*	-	✓
binary_buildpack*	-	✓
dotnet_core_buildpack*	-	✓

**i** Note

\* Community-supported build packs.

## Related Information

[Create a Python Buildpack for XS Advanced \[page 640\]](#)

[Create a PHP Buildpack for XS Advanced \[page 645\]](#)

[Custom Run-Time Environments in SAP HANA XS Advanced \[page 636\]](#)

[XS CLI: Buildpacks \[page 904\]](#)

## 7.4 Create the XS Advanced Application Package Descriptor

The package descriptor defines a JavaScript application's design-time and run-time dependencies in SAP HANA XS Advanced.

### Prerequisites

- File name:  
`package.json`
- Location:  
Application source-file folder, for example, `package.json`
- Format  
JavaScript Object Notation (JSON)

### Context

The build, deployment, and run-time dependencies of a JavaScript application are described in the `package.json` file. The `package.json` file is mandatory for JavaScript applications and it must be located in the JavaScript application's source-file folder, for example, `js/`.

To create an application package description for your new application project, perform the following steps:

## Procedure

1. Create the application resource-file structure.

For example, /appname1/

2. Create a sub-folder for the JavaScript source-files in the application root folder.

For example, /appname1/js

3. Create the application package description file.

The application descriptor file package.json must be located in the JavaScript source-files folder, for example, /appname1/js.

4. Define details of the application packages and any dependencies:

The contents of the package.json file must follow the required syntax.

## Related Information

[The XS Advanced Application Package Descriptor \[page 653\]](#)

[XS Advanced Application Package Descriptor Configuration Syntax \[page 656\]](#)

### 7.4.1 The XS Advanced Application Package Descriptor

A file whose contents describe the prerequisites and dependencies that apply to a JavaScript application in SAP HANA XS Advanced.

The dependencies of a JavaScript application are described in the package.json file. The package.json file is mandatory for JavaScript applications and must be located in the JavaScript application's source-file folder, for example, /path/appname/js.

Typically, the package.json file is created with the node.js command `npm init`, where you provide general information about the JavaScript application. Note that for XS JavaScript applications, you must also provide the following additional information:

- Dependency to XS JavaScript module (the XS compatibility layer)
- Dependency to the application router (see example below)
- Start command to specify node options

#### Note

For illustration purposes, the following example includes dependencies to a fictitious github account (on "acme.com"); you must adapt any URLs to suit the local development environment and dependencies.

## Sample Code

Application's Package Description (package.json)

```
{  
  "name": "<application_name>",  
  "description": "<application_description>",  
  "version": "<application_version>",  
  "repository": {  
    "url": "git@github.acme.com:xs-samples/nodejs-hello-world.git"  
  },  
  "dependencies": {  
    "@sap/xsjs": "^1.1.9"  
    "approuter": ">=1.0.1 <2.0.1"  
    "@sap/hdbext": "^1.0.2"  
    "@sap/xssec": "^1.0.5"  
  },  
  "engines": {  
    "node": ">=0.10.x <0.12"  
  },  
  "scripts": {  
    "start": "node --max-old-space-size=400 --expose-gc main.js"  
  }  
}
```

A package description is required for both the `web/` and `db/` modules. In the `web/` module, the information in the `package.json` file is enable the startup of the application router; the contents of the `package.json` file located in the `db/` module are used to start the SAP HANA HDI deployment application.

## SAP HANA Database Connections

To establish a connection to the database, an application must first look up the connection properties from the bound services and create an “hdb” client. The easiest way to do this is to add a dependency to `@sap/hdbext` to the application’s `package.json` file, as illustrated in the following example:

## Sample Code

```
"dependencies": {  
  "@sap/hdbext": "1.0.2"  
},
```

Next, the application has to create a connection, as illustrated in the following example:

## Sample Code

```
var xsConnection = require("@sap/hdbext");  
var hdbclient = xsConnection.createConnection(function(error) { ... });
```

## Connection Pools

The `@sap/hdbext` module includes a simple method for pooling connections. To use the connection-pool feature, you must first create the connection pool, as illustrated in the following example:

```
var pool = hdbext.getPool(hanaConfig, poolConfig);
```

Then you can acquire a connection client from the pool; the client is delivered in a callback, as illustrated in the following example:

```
pool.acquire(function(err, client) {});
```

If the connection client is no longer needed, it should be released back to the pool:

```
pool.release(client);
```

## Security

For Node.js, the client security library is an `npm` module called `@sap/xssec`. If you want your application to use the security library `@sap/xssec`, you must add the dependency to `@sap/xssec` in the `dependencies` section of your application's package description file, `package.json`, as illustrated in the following example:

### Sample Code

```
"dependencies": {  
    "@sap/xssec": "^1.0.5"  
},
```

## Outbound Connectivity

If a Node.js application needs to call external applications or services, it must do so by performing HTTP requests to the required services. The service can either be located in the same system (for example, an SAP HANA XS advanced system) or on an external system on the Internet. For connections to remote systems, it is recommended for Node.js applications to make use of the “`request`” module. For outbound connections, you need to consider the following:

- Proxy connections  
If your Node.js applications runs behind an HTTP proxy, the details of the proxy connection must be provided to the `request` module. Request can discover this information either from environment variables set in the application's manifest file (`manifest.yml`) or directly from the options object of the HTTP request.
- Authentication  
If the application needs to call a service that performs authentication by means of Security Assertion Markup Language (SAML) and JWT (authentication), and the calling service does so as well, then the SAML token can be taken from the incoming request and attached to the outgoing request.

### Sample Code

```
var options = {
  url: remoteURL,
  headers: {
    'authorization': req.headers.authorization
  }
};
```

#### Note

For this so called “principal propagation” to work, both the calling and the called application must be bound to the same User Account and Authentication (UAA) service. Otherwise the SAML token attached to the request will not be recognized and the request cannot be authenticated.

## Related Information

[Create the XS Advanced Application Package Descriptor \[page 652\]](#)

[XS Advanced Application Package Descriptor Configuration Syntax \[page 656\]](#)

[Maintaining Application Security in XS Advanced \[page 758\]](#)

## 7.4.2 XS Advanced Application Package Descriptor Configuration Syntax

The pacakge descriptor defines the prerequisites and dependencies that apply to packages in a JavaScript application deployed in the SAP HANA XS advanced run time.

The contents of the XS advanced application package descriptor, `package.json`, must be formatted according to the JSON rules. The required syntax for the properties and their corresponding keys is described below:

### Sample Code

```
{
  "name": "<application name>",
  "description": "<application description>",
  "private": true,
  "version": "1.3.1",
  "repository": {
    "url": "git@github.acme.com:xs-samples/nodejs-hello-world.git"
  },
  "dependencies": {
    "@sap/xsjs": "^1.0.5",
    "approuter": "^2.3.1"
  },
  "engines": {
    "node": ">=0.10.x <0.12"
  },
  "scripts": {
```

```
        "start": "node --max-old-space-size=400 --expose-gc main.js"
    }
}
```

## name

The name of the JavaScript application whose package prerequisites and dependencies are defined in this package description.

### Sample Code

```
"name": "<application name>",


```

## description

A short description the JavaScript application whose package prerequisites and dependencies are defined in this package description.

### Sample Code

```
"description": "<application description>",


```

## private

Use the `private` property to indicate if access to the package specified in `name` is restricted. Private packages are not published by `npm`.

### Sample Code

```
"private": true,
```

## version

The version of the JavaScript application whose package prerequisites and dependencies are defined in this package description.

The version value must follow the standard format for semantic versioning: <major>.<minor>.<patch>. The restriction ensures a reliable contract between a development environment and deployed system components.

### Sample Code

```
"version": "<Major_Version>.<Minor_Version>.<Patch_Version>"
```

### ➔ Tip

When defining `dependencies` and run-time `engines` in the package description, you can specify a range of versions, for example: >1.0.3, <=1.2.5, ^1.0.5 (compatible with version 1.0.5), or 1.2.x (any 1.2 version), or 1.1.0 – 1.3.12 (any version including or between the specified versions).

## repository

The absolute path to the repository used by the JavaScript application whose package prerequisites and dependencies are defined in this package description.

### Sample Code

```
"repository": {  
    "url": "<repository>@<host.name>:</path/to/your/repository>"  
},
```

### Sample Code

```
"repository": {  
    "url": "git@github.acme.com:samples/nodejs-hello-world.git"  
},
```

## dependencies

A list of dependencies that apply to the JavaScript application whose package prerequisites and dependencies are defined in this package description.

### Sample Code

```
"dependencies": {  
    "@sap/xsjs": "^1.2.5",  
    "approuter": "^2.3.1",  
    "@sap/cds": "^1.1.7",  
    "@sap/xssec": "^1.3.6",  
    "@sap/xsenv": "^1.2.2"
```

```
},
```

### Note

In the package description, you can define a range of versions, for example: `>1.0.3,<=1.2.5`, or `1.2.x` (any 1.2 version), and `1.1.0 - 1.3.12` (any version including or between the specified versions).

## engines

The run-time engines used by the application specified in the `name` property.

### Sample Code

```
"engines": {  
    "node": ">=0.10.x <0.12"  
},
```

### Note

In the package description, you can define one or a range of versions, for example: `>1.0.3,<=1.2.5`, or `1.2.x` (any 1.2 version), and `1.1.0 - 1.3.12` (any version including or between the specified versions).

## scripts

The script containing the command used to start the JavaScript application along with any additional (required) options and parameters.

### Sample Code

```
"scripts": {  
    "start": "node --max-old-space-size=400 --expose-gc main.js"  
},
```

## Related Information

[Create the XS Advanced Application Package Descriptor \[page 652\]](#)

# 8 Maintaining Application Services in XS Advanced

In XS advanced, applications can make use of services managed by a service broker.

In SAP HANA XS Advanced, application developers can make use of a catalog of services managed by a service broker, for example, for job schedules or user accounts and OAuth clients. To make use of a service, an instance of the service must be created and an the XS advanced application must be bound to the specified service instance. Services are either installed with the XS Advanced run-time platform, or, if required, deployed manually on the platform. The following services are available by default:

- File-system storage service
- SAP HANA Deployment Infrastructure (HDI) services
- User Account and Authentication (UAA) service
- Job Scheduler service
- Audit-log service
- Portal services

Services brokers are advertised in the marketplace. To check which services are available, you can use the `[xs|cf] marketplace` command, as illustrated in the following example:

## Sample Code

```
$ xs marketplace
Getting services from marketplace in org acme / space devel as admin...
OK
service      plans          description
-----
fs-storage   free           Environment variable denotes the
                           root of client app's file system
hana         hdi-shared, sbss, schema
                           securestore
xsuaa        default, devuser, space
auditlog     free           XS UAA Service Broker for user
                           authentication & authorization...
portal-services site-host, site-content,
                  admin-cockpit
jobscheduler default        Audit log broker on XS advanced
                           Service broker for creating and
                           accessing portal DB
                           Job Scheduler on XS advanced
```

Services that are manually deployed are pushed as applications (for example, using the `xs/cf` command) and exposed as services. Services that provide their own persistence, for example, HDI services, must be installed with the platform. This is because deployed services (pushed as applications) run in containers that do not provide persistent access to the corresponding file-system. In addition, services that are required by the SAP HANA platform itself, for example, the UAA service for user account management and authentication, are installed with the platform to avoid issues at boot time. Each instance of a service is assigned a “service plan”.

## Tip

The service plan is mapped to a corresponding application “resource type” that is typically specified in the `resources` section of the application’s MTA deployment descriptor (`mtad.yaml`). For example, the

resource type “com.sap.xs.hana-sbss” is mapped to the “hana” service plan “sbss”; the resource type “com.sap.xs.uaa-devuser” is mapped to the “xsuaa” service plan “devuser”.

Services that bind to other services for their persistence and are only consumed by applications (for example, the job-scheduler service) can be deployed on the platform by pushing them (and the corresponding service broker) as applications. Note that the job-scheduler service is deployed and available by default.

➔ Tip

The `<VCAP_SERVICES>` variable is typically used to provide applications with configuration details of a service, for example: the name of the service, any logon credentials required, the host name and port to use for connections, and the URL for the service end-point.

## Related Information

[Create a Service Instance \[page 681\]](#)

[Default Services in XS Advanced \[page 661\]](#)

[Service Plans and Resources in XS Advanced \[page 678\]](#)

## 8.1 Default Services in XS Advanced

In XS advanced, services are either installed with the run-time platform or deployed manually on the platform.

Service brokers are used to manage service instances. The following services are available by default:

Table 90: Default XS Advanced Service Brokers

Service Broker Name	Service Plans	Description
auditlog	free	Audit log service broker on the XS advanced platform
fs-storage	free	Provides storage Volume Services, which define a special type of service broker that can be used to make persistent file systems available to any bound applications
hana	hdi-shared sbss schema securestore	The service broker for the SAP HANA database and deployment infrastructure (HDI) service component
jobscheduler	default	Job Scheduler service broker on the XS advanced platform

Service Broker Name	Service Plans	Description
portal-services	site-host site-content admin-cockpit	Service broker for creating and accessing portal DB
xsuaa	default devuser space	The XS UAA Service Broker for user authentication and authorization services in XS advanced

## Related Information

[Service Plans and Resources in XS Advanced \[page 678\]](#)

[Service Types in XS Advanced \[page 679\]](#)

[Maintaining Application Services in XS Advanced \[page 660\]](#)

### 8.1.1 File-System Storage Services in XS Advanced

Configure file-system services for XS advanced applications.

Since an XS advanced application runs inside a container that might be stopped and discarded at any time, the file system of a running XS advanced application is ephemeral. To provide support for applications that require a persistent POSIX file system to function, XS advanced provides Volume Services, which enable you to define a special type of service broker that makes persistent file systems available to any bound applications.

## Service Configuration

The File System Services available on premise and on Cloud Foundry are implementations of a volume service. With both XS advanced on-premise and on Cloud Foundry you can create a service instance by using the service `fs-storage` with the service plan “`free`”, as illustrated in the following examples:

### Sample Code

```
[xs | cf] create-service fs-storage free <service-instance-name>
```

You can display a list of the available service plans using the `xs marketplace` command, as illustrated in the following example:

#### Sample Code

```
[xs | cf] marketplace -s fs-storage
service plan      description
-----
free              A NFSv3 share with a 1 GB quota; the plan ...
nfsv3-10          A NFSv3 share with a 10 GB quota; the plan ...
nfsv3-100         A NFSv3 share with a 100 GB quota; the plan ...
```

## Service Maintenance

You can bind the service instance to your XS advanced application and use it by inspecting the bound service instance entry in the `<VCAP_SERVICES>` environment variable of your application. This entry will contain a `volume_mounts` field that contains an array of length 1 of `volume_mounts` that have the following properties:

- `mode`  
Defines the access mode to the named file system; currently only “`rw`” is allowed
- `device_type`  
Defines the type of file-system access; currently only “`shared`” is allowed
- `container_dir`  
Defines the path to the location where the persistent file system is mounted and made available, and where it can be consumed using regular file-IO operations.

#### Sample Code

`VCAP_SERVICES` for `fs-storage`

```
"VCAP_SERVICES" : {
  "fs-storage" : [ {
    "name" : "volume1",
    "label" : "fs-storage",
    "tags" : [ ],
    "plan" : "free",
    "credentials" : {
      "storage-path" : "/path/to/resource/18b48853-888f-4109-9730-
dbf5041c6a24"
    },
    "volume_mounts" : [ {
      "mode" : "rw",
      "device_type" : "shared",
      "container_dir" : "/path/to/directory/"
    } ]
  } ]}
```

## Related Information

[Default Services in XS Advanced \[page 661\]](#)

[Maintaining Application Services in XS Advanced \[page 660\]](#)

### 8.1.2 Deployment-Infrastructure Services in XS Advanced

The SAP HANA XS Deployment Infrastructure service (HDI) is the central infrastructure component of XS advanced model for application-container management.

The SAP HANA Deployment Infrastructure (HDI) is a service layer of the SAP HANA database that simplifies the deployment of SAP HANA database objects by providing a declarative approach for defining database objects (as design-time artifacts) and ensuring a consistent deployment into the database. The HDI service enables you to create and manage not only HDI containers but also plain schemata, and secure stores on shared SAP HANA XS advanced systems.

The deployment of database objects with HDI is based on a container model where each container corresponds to a database schema. Containers can be used for multiple deployments of the same database artifacts, and for development sandboxes, too. Containers are isolated from each other; each database schema with its deployed database objects is owned by a per-schema technical database user. By default, a cross-container access at the database level is not possible. However, such cross-container access could be granted by means of database privileges.

In the XS advanced, database service instances are created by a database-specific service broker, for example, the SAP HANA database service broker. In the context of the XS advanced run time, a database service instance corresponds to a single database schema and an assigned technical database user. The HANA database service broker delegates schema creation to HDI in such a way that HDI can create its required metadata, database users, and database roles inside the database.

#### i Note

When the service is bound to an application, the bind operation “injects” into the application an application user (for use by the application code) and a deployment user (for use by the database artifact deployer). The container (schema) is on a shared SAP HANA database and, for this reason, is not suited for a productive deployment.

## HDI Service Plans

In XS advanced, SAP HANA is exposed as a managed service with the service name “hana”. If an application is bound to an instance of the HANA service, the credentials required to access SAP HANA are defined in the application's `<VCAP_SERVICES>` environment variable. A tenant database of an SAP HANA instance can be mapped to an Organization or Space in XS advanced, and all SAP HANA service instances created within such an Organization or Space relate to a schema in the mapped tenant database.

## Sample Code

### SAP HANA Managed Service

```
[xs | cf] marketplace -s hana

Getting services from marketplace...
Getting plans for service "hana"...

service plan      description          free/paid
-----
hdi-shared        An HDI container on an SAP HANA database    free
sbss              User with permissions to use the SBSS    free
schema            A schema on an SAP HANA database           free
securestore       User with permissions to use the secure store  free
```

Table 91: HANA Service Plans and Availability

HANA Service Plan	Description	On-Premise	CF
<a href="#">hdi-shared</a>	An HDI container on an SAP HANA database	✓	✓
<a href="#">sbss</a>	User with permissions to use the SBSS	✓	✓
<a href="#">schema</a>	A schema on an SAP HANA database	✓	✓
<a href="#">securestore</a>	User with permissions to use the secure store	✓	✓

## The “`hdi-shared`” Service Plan

When you create and bind a service instance with the service plan `hdi-shared`, an application receives the credentials required for access to an HDI container, which is basically a database schema that is equipped with additional metadata.

HDI containers ensure isolation, and within an SAP HANA database you can define an arbitrary number of HDI containers. The same objects can be deployed multiple times into different HDI containers in the same SAP HANA database, for example, to install several instances of the same software product in the same SAP HANA database. HDI containers are isolated from each other by means of schema-level access privileges. Cross-container access at the database level is prevented by default, but can be enabled by explicitly granting the necessary privileges, for example, using synonyms.

Database objects (tables, views, procedures, and so on) have an owner; the user who created the object. An interesting aspect of object ownership is that all objects are deleted from the database, when their owner is deleted. If application objects would be created by end-users, they would be deleted, when the end-user is deleted, for example because an employee leaves the organization. HDI ensures that during deployment all database objects are created by a container-specific technical user, which is never deleted as long as the container exists.

In HDI, database schema content (for example, tables, views, procedures, etc.) is defined in corresponding design-time files as part of a development project. These definition artifacts are pushed to the platform as part of a special Node.js application, the HDI Deployer application, which is part of the XS advanced. This deployer

application binds to a HANA service instance and, upon startup, creates the set of database objects that correspond to the pushed definition files, for example: myTable.hdbtable, myView.hdbview, or myProcedure.hdbprocedure.

### Sample Code

Service-Creation Parameters:

```
{"schema": "<schema-name>", "database_id": "<database_id>"}
```

#### Note

If no “schema” parameter is used, a random schema name is created and accessible via `<VCAP_SERVICES>` after the service instance is bound to an application. The parameter “database\_id” must be used if multiple tenant databases are mapped to the same XS advanced organization or space.

Random schema names is the recommended way to create a service instance; setting a **fixed** schema name creates a global singleton in the database and means that it is not possible to create a service instance with the same schema name.

### Sample Code

Service-Binding Parameters:

```
{"roles": "<list of user roles>"}
```

Every time the service instance is bound to an application, the service broker creates a technical application user who is named “user” in the service instance’s credentials. The bound application must provide the technical user’s credentials when accessing the HDI container’s run-time schema. This user is equipped with the service instance’s global access role (`<schema-name>::access_role`), where `<schema-name>` is the name of the HDI container schema.

In order to assign roles from the HDI content to the technical application binding user, the HDI deployer implements an automatic assignment of the “default\_access\_role” if the role is included in the deployed content. By using the service-binding parameter roles when binding an HDI container to an application, it is possible to declare any role which will be assigned to the technical binding “user”, as illustrated in the following example:

```
$> [xs | cf] bind-service my-app my-hdi-container -c {"roles": "sap.myapp.roles::read_access_role" }
```

In the application deployment descriptor (`mtad.yaml`), this would look like the following code:

### Sample Code

Application Deployment Descriptor

```
ID: com.sap.xs2.samples.my-app
version: 0.1.0
modules:
  - name: my-app1
    type: javascript.nodejs
    requires:
      - name: my-hdi-container
```

```
parameters:  
  config:  
    roles: sap.myapp.roles::read_access_role  
...
```

## The “sbss” Service Plan

Access to a service instance needs to be protected. When an application uses a service, the application needs at least the credentials (for example, user name and password) of a technical user that the service must authenticate before allowing the application to access the service functionality.

### Note

This does not apply to the "business (end) user" who accesses the application.

Since each service broker needs the ability to create, validate and delete credentials, we offer these three distinctive functions within this Service Broker Security Support (SBSS) reuse component:

- Credential generation  
Credentials can be generated by the service broker either when a service instance is created or when the created service instance is bound to an application. It is recommended to use application-specific credentials in order to allow for the revoking of credentials of one application without influencing any other bound applications. Credentials are generated and made available to the bound application during the application-bind process.
- Credential validation  
A service instance needs to validate the credentials supplied by the application requesting access. Access to the service is only be granted if the provided credentials are valid and match the credentials required by the particular service instance.
- Credential de-provisioning  
When a service instance is unbound from an application, the credentials need to be revoked in order to prevent unauthorized access attempts (for example, if the application stored the service access credentials).

## The “schema” Service Plan

The schema service plan creates a plain schema, which you need to manage by hand. You should consider using this service plan if your application uses an OR Mapper concept and a framework is available that creates the necessary database resources on demand.

### Sample Code

Service Creation Parameters:

```
{"schema": "<schema-name>", "database_id": "<database_id>"}
```

### **i** Note

If no schema parameter is used, a random schema name is created and accessible via `<VCAP_SERVICES>` after the service instance is bound to an application. Random schema names is the recommended way to create a service instance; setting a **fixed** schema name creates a global singleton on the database and means that it is not possible to create a service instance with the same schema name.

## The “**securestore**” Service Plan

The **securestore** service plan enables a bound application to make use of stored procedures that provide access to the SAP HANA secure store. This functionality is available when using the XS JavaScript `$.security.store` API and allows the bound application not only to insert encrypted values **into** the secure store but also to retrieve (or delete) encrypted data **from** the secure store.

## Related Information

[Create a Service Instance \[page 681\]](#)

[Default Services in XS Advanced \[page 661\]](#)

[Maintaining Application Services in XS Advanced \[page 660\]](#)

## 8.1.3 User Account and Authentication Services in XS Advanced

The SAP HANA XS User Account and Authentication service (UAA) is the central infrastructure component of XS advanced model for user authentication.

The UAA uses OAuth 2.0 for authentication to the XS advanced application and database containers. In the context of the OAuth flow, the UAA provides scopes, role templates, and attributes to applications deployed in the XS advanced run time. If a business user has a certain scope, an access token with this scope is issued, which enables a user to run the application: the scopes are used to define the model that describes who has the authorization to start an application running in the XS advanced run time.

### **i** Note

The UAA service supports either an SAML 2.0 identity provider (IDP) or an SAP HANA database user store as external user stores.

The XS advanced model's UAA service provides a programming model for developers; it enables developers to define templates that can be used to build role and authorization models for business users of applications deployed to the XS advanced run time. In the OAuth 2.0 work flow, the UAA acts as an OAuth server, which includes both token and revocation end points.

## Security Services in the Application Router

In the XS advanced run time, the application router is the single point-of-entry for an application running in SAP HANA XS advanced; the application router is part of the XS advanced application instance and triggers the user-authentication process in the UAA. In the OAuth work flow, the XS application instance (including the application router and any bound containers) is the OAuth 2.0 client, which handles user authentication and authorization.

## Container Security APIs

In the context of OAuth 2.0, the security APIs for the XS advanced run time container act as resource servers; the APIs provide the security context for user authentication in XS advanced. When the user authentication process is triggered, the XS advanced container security APIs receive an HTTP header with "Authorization: Bearer" and a JSON Web token (JWT) from the application router.

The token contains information describing the user and any authorization scopes, which must be validated by the XS advanced container security APIs using the security libraries provided by the UAA. Applications deployed in the XS advanced run time can use the security API to check if scope values have been assigned to the user or application. The XS container security API provides the security **context** for XS advanced applications (for example, scopes, attributes, token information); the JWT token initializes the XS advanced security context. A security API is available for the following application run time containers:

- Node.js API
- XS JavaScript (compatibility layer) API
- Java API

Every XS advanced application's run time container must use the XS advanced container security API to provide the security context. The XS advanced security context is initialized with the JWT token and enables you to perform the following functions:

- Get user information
- Check an authorization scope
- List authentication attributes
- Get a token

## XS UAA Service Plans

The XSUAA service plan you use determines the impact on the `xsappname` that is written into the credentials section of the environment from the application bound to the service (`[xs | cf] env <appname>`). Each of the XSUAA service plans is designed for use with a particular kind of user-security scenario, and service availability depends on platform, for example, on-premise (`xs`) and SAP Cloud Platform (SCP) @ Cloud Foundry (`cf`), as illustrated in the following examples:

## Sample Code

SAP HANA Managed Service

```
xs marketplace -s xsuaa

Getting services from marketplace...
Getting plans for service "xsuaa"...

service plan      description          free/paid
-----
default           Production            free
devuser           Development           free
space             Space-specific name adoption   free
```

## Sample Code

SAP HANA Managed Service (SCP/Cloud Foundry)

```
cf marketplace -s xsuaa

Getting services from marketplace...
Getting plans for service "xsuaa"...

service plan      description          free/paid
-----
application      PAAS enabled application plan   free
broker           PAAS enabled application plan   free
```

The XS User Account and Authorization service provides the following plans for XS advanced:

Table 92: XSUAA Service Plans and Availability

XSUAA Service Plan	Description	On-Premise	CF
default	Production use	✓	-
devuser	Development user; required if you want to run applications for different developers in the same XS advanced "space"	✓	-
space	Space-specific name adoption; required if you want to run the same XS advanced applications in different XS advanced "spaces".	✓	-
application	PAAS-enabled application plan	-	✓
broker	PAAS-enabled broker plan	-	✓

## XSUAA Service Plan: “default”

If you use the XSUAA service plan "default", you must ensure that the name of the XS advanced application specified in `xsappname` in the application's security descriptor `xs-security.json` is unique in the XS

advanced landscape in which the application is deployed. This means that it is not possible to deploy an application more than once with service plan "default" unless you modify the ""xsappname"" property in the corresponding `xs-security.json` file. You would get an error message during the second deployment.

➔ Tip

The `xsappname` that is written into the credentials section of the environment is identical to the application name specified in the application's `xs-security.json` file.

The obvious disadvantage of this plan is that it is not possible to find out which application names are already in use in the XS advanced landscape (defined in "`xsappname`"). If a duplicate application name is used, the deployment of the application with the duplicate name will fail. For this reason, it is recommended to use the `default` service plan only in smaller XS advanced landscapes, for example, your private test landscape.

## XSUAA Service Plan: "devuser"

The service plan "devuser" requires that the value of the "`xsappname`" property in the the application descriptor `xs-security.json` is unique in the XS advanced landscape for the development user that deploys the application. This means that an application can be deployed by several development users at the same time but not twice by the **same** development user (unless the developer modifies the "`xsappname`" property in the corresponding application's `xs-security.json`file). A duplicate application name would result in an error message during the second deployment.

➔ Tip

The "`xsappname`" that is written into the credentials section of the environment is enhanced with a suffix and has the following format: "`<xsappname>!u<user index>`", where `<user index>` is an incrementing number assigned per user that the XS UAA maintains internally and differs from landscape to landscape, for example, `myapp!u2`.

The advantage of "devuser" plan is that you know which `xsappname` your development user has already deployed in the XS advanced landscape where you want to deploy your application. It is recommended to use the "devuser" plan for XS advanced on-premise landscapes where multiple developers are working independently on the same application.

## XSUAA Service Plan: "space"

The service plan "space" requires that the value of the "`xsappname`" property specified in the `xs-security.json` file is unique in the XS advanced landscape within the space in which the application is deployed. This means that an application may be deployed in several spaces at the same time but not twice in the same space (unless you modify the value of "`xsappname`" in the second application's `xs-security.json` security descriptor). Failure to adjust the XS advanced application name would result in an error message during the second deployment.

### ➔ Tip

The "xsappname" that is written into the credentials section of the environment is enhanced with a suffix and has the following format: "<xsappname>!i<space index>", where the <space index> is a number that the UAA maintains internally and differs from landscape to landscape, for example, myapp!i3.

The advantage of the "space" service plan compared to the "default" service plan is that it is possible to see which xsappnames are already in use in the XS advanced space where you want to deploy your application. It is recommended to use the "space" plan for XS advanced on-premise landscapes.

## XSUAA Service Plan: "application"

The service plan "application" requires that the value of the "xsappname" property specified in the `xs-security.json` file is unique in the Cloud Foundry landscape within the tenant in which the application is deployed. This means that an application may be deployed in several tenants at the same time but not twice in the same tenant (unless you modify the value of "xsappname" in the second application's `xs-security.json` security descriptor). Failure to adjust the application name would result in an error message during the second deployment.

### ➔ Tip

The "xsappname" that is written into the credentials section of the environment is enhanced with a suffix and has the following format: "<xsappname>!t<tenant index>", where the <tenant index> is a number that the UAA maintains internally and differs from landscape to landscape, for example, myapp!i5.

This service plan is only available in Cloud Foundry landscapes. It is highly recommended to use the "application" service plan for all Cloud Foundry deployments.

## XSUAA Service Plan: "broker"

The service plan "broker" requires that the value of the "xsappname" property specified in the `xs-security.json` file is unique in the Cloud Foundry landscape within the tenant in which the application is deployed. This means that an application may be deployed in several tenants at the same time but not twice in the same tenant (unless you modify the value of "xsappname" in the second application's `xs-security.json` security descriptor). Failure to adjust the application name would result in an error message during the second deployment.

### ➔ Tip

The "xsappname" that is written into the credentials section of the environment is enhanced with a suffix and has the following format: "<xsappname>!b<tenant index>", where the <tenant index> is a number that the UAA maintains internally and differs from landscape to landscape, for example, myapp!i8.

This service plan is only available in Cloud Foundry landscapes. It is highly recommended to use the "broker" service plan for Cloud Foundry deployments where you want to provide a reuse service to other applications.

## Related Information

[Create a Service Instance \[page 681\]](#)

[Default Services in XS Advanced \[page 661\]](#)

[Maintaining Application Services in XS Advanced \[page 660\]](#)

[Maintaining Application Security in XS Advanced \[page 758\]](#)

### 8.1.4 Audit Log Services in XS Advanced

The auditlog service allows you to perform audit-log tasks within the SAP HANA XS advanced run-time environment.

The SAP HANA XS advanced audit-log service is intended for use by Web applications that want to send audit log messages to a remote persistent storage. SAP HANA includes a Java client that can be used by these Web applications.

The SAP HANA XS advanced Java runtime provides an audit-log application programming interface (API) that enables applications and run-time environments to comply with the regulations on audit logging of the PIL Security standard. There are three main categories of audit logs:

- Security events
- Data access events
- Configuration change events

The audit-log API provides different interfaces that can be used for these different log types.

## Service Configuration

You can use the XS command-line interface (CLI) to check that the audit-log service is displayed in the list of services registered in the Service Marketplace:

```
$ xs marketplace
Getting services from marketplace in org acme / space devel as admin...
OK
service      plans          description
-----
fs-storage   free           Environment variable denotes the
                           root of client app's file system
hana         hdi-shared, sbss, schema
                           securestore
xsuaa        default, devuser, space
auditlog     free           XS UAA Service Broker for user
                           authentication & authorization...
portal-services site-host, site-content,
                           admin-cockpit
jobscheduler default
                           Job Scheduler on the XSA platform
```

To deploy a client application, you can use the XS CLI `xs push` command, as illustrated in the following example, which assumes the client is called `auditlog-client`

```
xs push auditlog-client
```

To create an instance of the audit-log service, use the XS CLI `xs create-service` command, as illustrated in the following example:

```
xs create-service auditlog free auditlog-instance
```

### ➔ Tip

The service plan `free` enables unlimited access to the functionality provided by the audit-log service.

To bind the audit-log service instance to the client application that you deployed, use the `xs bind-service` command:

```
xs bind-service auditlog-client auditlog-instance
```

Check that the deployed application (`auditlog-client`) has been successfully bound to the instance by checking its `<VCAP_SERVICES>` environment variable, for example, with the `xs env` command:

```
xs env auditlog-client
```

The audit-log service broker provisions these credentials for the client application. The environment variable configuration should look like the following example:

### Sample Code

#### VCAP\_SERVICES Environment Variable

```
"VCAP_SERVICES" : {  
  "auditlog" : [ {  
    "name" : "auditlog-instance",  
    "label" : "auditlog",  
    "tags" : [ "auditlog", "audit log", "xsa" ],  
    "plan" : "free",  
    "credentials" : {  
      "password" : "Aa_93080244...221922102809335986",  
      "user" : "SBSS_6592728326...602494781306527846",  
      "url" : "http://localhost:4243"  
    }  
  } ]  
}
```

## Service Maintenance

To unbind your application from the audit-log service instance, use the `xs unbind-service` command in an XS CLI command shell, as illustrated in the following example, which assumes that the application name is `auditlog-client` and the name of the service instance is `auditlog-instance`:

```
xs unbind-service auditlog-client auditlog-instance
```

To delete the service instance, use the `xs delete service` command, as follows:

```
xs delete-service auditlog-instance
```

## Related Information

[Configure an Application to Use the Audit Log Service \[page 611\]](#)

[Service Plans and Resources in XS Advanced \[page 678\]](#)

## 8.1.5 Job Scheduler Services in XS Advanced

The Job Scheduler service enables you to create and schedule long-running operations or jobs.

In SAP HANA XS advanced model, the Job Scheduler is an application service. The Job Scheduler service enables you to create and schedule long-running operations or jobs. This service is deployed during the installation of the SAP HANA XS advanced model.

## Service Configuration

You can use the XS command-line interface (CLI) to check that the `jobscheduler` service is displayed in the list of services registered in the Service Marketplace:

```
$ xs marketplace
Getting services from marketplace in org acme / space devel as admin...
OK
service      plans                      description
-----
fs-storage   free                       Environment variable denotes the
                                         root of client app's file system
hana         hdi-shared, sbss, schema    SAP HANA database
             securestore
xsuaa        default, devuser, space   XS UAA Service Broker for user
                                         authentication & authorization...
auditlog     free                       Audit log broker on the XSA Platform
portal-services site-host, site-content,
                  admin-cockpit
jobscheduler default                    Service broker for creating and
                                         accessing portal DB
                                         Job Scheduler on the XSA platform
```

To deploy a client application, you can use the XS CLI `xs push` command, as illustrated in the following example, which assumes the client is called `job-scheduler-client`

```
xs push job-scheduler-client
```

To create an instance of the `jobscheduler` service, use the XS CLI `xs create-service` command, as illustrated in the following example:

```
xs create-service jobscheduler default scheduler-instance
```

## ➔ Tip

The service plan `default` ensures standard access to the functionality provided by the `jobscheduler` service. For more information about access configuration, see [Job Scheduler Access \[page 677\]](#) below.

To bind the `jobscheduler` service instance to the client application that you deployed, use the `xs bind-service` command:

```
xs bind-service job-scheduler-client scheduler-instance
```

Check that the deployed application (`job-scheduler-client`) has been successfully bound to the instance by checking its `<VCAP_SERVICES>` environment variable, for example, with the `xs env` command:

```
xs env job-scheduler-client
```

The `jobscheduler` service broker provisions these credentials for the client application. The environment variable configuration should look like the following example:

### Sample Code

#### VCAP\_SERVICES Environment Variable

```
"VCAP_SERVICES" : {
  "jobscheduler" : [ {
    "name" : "scheduler-instance",
    "label" : "jobscheduler",
    "tags" : [ "jobscheduler" ],
    "plan" : "default",
    "credentials" : {
      "password" : "Aa_93080244...221922102809335986",
      "user" : "SBSS_6592728326...602494781306527846",
      "url" : "https://<REST_URL>:<REST_PORT>"
    }
  } ]
}
```

## Job Schedule Execution Mode

Job Scheduler supports the following **modes** for applications to execute a job:

- Synchronous Mode  
Suitable for jobs that run for a short span of time, for example, an OData service end point
- Asynchronous Mode  
Suitable for jobs that run for a long span of time, for example, end points which trigger batch processing

## Job Schedule Execution Type

Job Scheduler provides the following **types** of schedules for a job:

- Recurring Schedule  
Runs periodically at a specified time, dates, or interval. Recurring schedules can be created in the following ways:
  - The `repeatInterval` parameter:  
Defines the interval in human-readable text (for example, “2 minutes”), which can be used to set up a recurring schedule. The repeat interval defines the gap between each run of the schedule.
  - The `cron` parameter:  
Defines a `cron` expression (for example, `"cron": "* * * * *`) used to represent a set of times, when the job is executed.
  - The `repeatAt` parameter:  
Defines the exact time, every day, when the job is executed.
- One-Time Schedule  
Runs only once at the specified time. One-time schedules can be created in the following ways:
  - Human-readable text string:  
A human-readable text string that defines the specific time for schedule execution (for example: “10 hours from now”, “3.30pm”, or “Friday at 2am”)
  - Using a Date object, with a pre-defined format, for example,  
`"startTime": {"date": "2015-10-20 4:30 +0000", "format": "YYYY-MM-DD HH:mm Z"}`  
The string is checked against both IETF-compliant RFC 2822 timestamps and ISO-8601.

## Job Scheduler Access

The Job Scheduler can be accessed and used in the following ways during application development:

- APIs:  
The Job Scheduler service offers RESTful and client-specific APIs for Java and Node.js. The administrator **scope** is required to use the Job Scheduler API to maintain run time configurations for jobs and job schedules.
- User Interface:  
The *Job Scheduler Dashboard* is the tool used to manage run time configurations for jobs and job schedules. Administrator authorization is required to maintain jobs and job schedules in the *Job Scheduler Dashboard*.

### ➔ Tip

You can program actions in any programming language or platform. The runtime also supports jobs created in the SAP HANA XS classic version.

## Related Information

- [Default Services in XS Advanced \[page 661\]](#)
- [Service Plans and Resources in XS Advanced \[page 678\]](#)
- [Scheduling Jobs in XS Advanced \[page 687\]](#)

## 8.2 Service Plans and Resources in XS Advanced

A service plan is a particular type of service (for example, a database configuration) that is available for use.

A service broker advertises **service plans** using the catalog. You can use the advertised plans to configure the service you want to create. For example, you can use the “`hdi-shared`” service plan to create a plain HDI container on a shared HANA database system.

### ➔ Tip

The service plan is mapped to a corresponding application “resource type” that is typically specified in the `resources` section of the application’s MTA deployment descriptor (`mtad.yaml`).

For example, the resource type “`com.sap_xs.hana-sbss`” is mapped to the “`hana`” service plan “`sbss`”; the resource type “`com.sap_xs.uaa-devuser`” is mapped to the “`xsuaa`” service plan “`devuser`”.

Table 93: Services, Service Plans, and MTA Resources

Service	Service Plan	Resource	Created Service
auditlog	free	com.sap_xs.auditlog	Audit-log service (unlimited access)
fs-storage	free	com.sap_xs.fs	File-system storage service (unlimited access)
hana	hdi-shared	com.sap_xs.hdi-container	HDI container service
	sbss	com.sap_xs.hana-sbss	Service-broker security service
	schema	com.sap_xs.hana-schema	Plain schema service
	securestore	com.sap_xs.hana-secure-store	SAP HANA secure-store service
managed-hana	hdi-shared	com.sap_xs.managed-hdi-container	Managed HDI container service
	schema	com.sap_xs.managed-hana-schema	Managed SAP HANA secure-store service
	securestore	com.sap_xs.managed-hana-securestore	Managed SAP HANA secure-store service
jobscheduler	default	com.sap_xs.jobscheduler	Global job-scheduler service
portal-services *	site-host	com.sap.portal.site-host	Portal DB site-host service
	site-content	com.sap.portal.site-content	Portal DB content service

Service	Service Plan	Resource	Created Service
	admin-cockpit	-	Portal DB admin cockpit service
sds	default	com.sap.xs.sds	Global streaming analytics service
xsuaa	default	com.sap.xs.uaa	Global UAA service
	devuser	com.sap.xs.uaa-devuser	Development-user UAA service
	space	com.sap.xs.uaa-space	UAA service for a space
	application	com.sap.xs.uaa-application	PaaS-enabled app plan
	broker	com.sap.xs.uaa-broker	PaaS-enabled app plan

### ⚠ Restriction

\* The MTA resource type `com.sap.portal.site-host` is only for use with the SAP Node.js module `@sap/site-entry`. The MTA resource type `com.sap.portal.site-content` is only for use with the SAP Node.js module `@sap/site-content-deployer`.

## Related Information

[MTA Deployment Descriptor Syntax \[page 145\]](#)

## 8.3 Service Types in XS Advanced

Services provided in XS advanced are available in different types.

XS advanced applications can make use of services that are either created by a user or available for general use in the service market place. Services available in the Service Marketplace are managed by a service

broker. The following table lists and explains the types of service to which you can bind your XS advanced applications:

Table 94: XS Advanced Service Types

Service Type	Description	Example
Managed	Services that are available in pre-configured scopes in the Service Marketplace	fs-storage xsuaa hana auditlog portal-services jobscheduler.
User-provided	Services that are provided on demand and by hand for use by applications in a specific, dedicated space. User-provided services are <b>not</b> available in (or managed by) the Service Marketplace and are not associated with a service plan. Unlike managed services, user-provided services do not include any automatic provisioning of resources or credentials; these must be provided manually.	lcm-service-credentials

To display a list the services running in the current organizational space, log in to an XS advanced space and run the `services` command, as illustrated in the following example:

```
[xs | cf] services
Getting services in org "myorg" / space "SAP" as XSA_ADMIN...
Found services:
name           service    plan      bound apps
-----
myHdiService   hana       hdi-shared xmla
myUaaService   xsuaa     space      xmla
xsa-cockpit-db hana       schema    xsa-cockpit
xsa-cockpit-uaa xsuaa     space      xsa-cockpit
lm-service-credentials user-provided product-installer
```

For details of the services available in the service marketplace, and the corresponding service plans, use the `marketplace` command, as illustrated in the following example:

```
[xs | cf] marketplace
Getting services from marketplace...
service      plans          description
-----
fs-storage   free           xs file service...
xsuaa        default, devuser, space  XS UAA Service Broker...
hana         hdi-shared, sbss, schema, securestore SAP HANA database
managed-hana hdi-shared, schema, securestore Run-time service instances for SAP HANA DB
auditlog     free           Audit logs on XS advanced
portal-services site-host, site-content, Creating and accessing portal DB
jobscheduler admin-cockpit  Job Scheduler for repeated tasks
default
```

## Related Information

[Maintaining Application Services in XS Advanced \[page 660\]](#)

[Default Services in XS Advanced \[page 661\]](#)

[Service Plans and Resources in XS Advanced \[page 678\]](#)

[XS CLI: Services Management \[page 874\]](#)

## 8.4 Create a Service Instance

Make a service instance available to applications.

### Context

Developers must create an instance of a service before it can be used and then bind the service to the application that needs to use it. The service instance also requires a **service broker**.

To create an instance of a service controlled by an SAP HANA service broker, perform the following steps:

### Procedure

1. Create an instance of a **service broker**.

The service-broker instance requires a name, for example, “myServiceBroker”:

```
xs create-service-broker <myServiceBroker> <USER> <PASSWORD> http://<service-
broker-url>
```

2. Check that the new service instance is up and running.

Check details of the new service by displaying a list of the service instances available in the service **market place**:

```
xs marketplace
```

3. Create an instance of the **service**.

The new service instance requires a name, for example, “myService” and is controlled by the service-broker instance you already created:

```
xs create-service myService default <myServiceInstance>
```

#### Tip

In this example of how to create a service, “default” refers to the service plan to use. The HDI and UAA services provide additional service plans.

- Bind the service instance to an application.

```
xs bind-service <myApp> <myServiceInstance>
```

- Verify that the application has been bound to the service instance.

To confirm the application-service binding, check the environment variable `<VCAP_SERVICES>`.

```
xs env <myApp>
```

- Unbind the service instance from an application.

```
xs unbind-service <myApp> <myServiceInstance>
```

- Delete the service instance.

```
xs delete-service <myServiceInstance>
```

- Remove the service broker you created.

```
xs delete-service-broker <myServiceBroker>
```

- Verify that the broker has been deleted.

To confirm that the service broker has been removed, check the service marketplace:

```
xs marketplace
```

## Related Information

[The SAP HANA Service-Broker API \[page 683\]](#)

[Maintaining Application Services in XS Advanced \[page 660\]](#)

### 8.4.1 The SAP HANA Service Broker

The service broker manages instances of services used by applications running in the SAP HANA XS advanced model run time.

The service broker is responsible for maintaining and managing instances of service brokers and all available services, for example, Job Scheduler, or XS User Account and Authentication (UAA). Services are bound to the application that requires the services provided by a service instance. Developers can use the service broker to bind service instances to the application that wants to make use of the advertised services. The SAP HANA service broker also manages the SAP HANA databases made available in XS advanced.

A dedicated application-programming interface (API) is provided to enable access to the service broker and any managed services; the API enables you to create and delete service instances (as well as instances of any service brokers that control them) and bind the services to any application. You can also use the commands `xs marketplace` and `xs services` to display a list of the currently available services, either in the marketplace or in a target organizational space.

## Related Information

[Create a Service Instance \[page 681\]](#)

[Maintaining Application Services in XS Advanced \[page 660\]](#)

### 8.4.2 The SAP HANA Service-Broker API

Maintain and manage SAP HANA XS services and service brokers, for example: list, create, delete, bind, and update.

```
xs <Service_Broker_COMMAND> <Service_Broker_NAME>
```

```
xs <Service_COMMAND> <Service_NAME>
```

Table 95: XS CLI: Services Command Overview

Command	Alias	Description
marketplace	m	List available services in the marketplace
services	s	List all services in the target space
managed-service		List all instances of a managed service in the target space
managed-service		Display details of a selected instance of a managed service in the target space
create-service	cs	Create a service instance
update-service		Update a service instance
delete-service	ds	Delete a service instance
delete-service		Rename a service instance
bind-service	bs	Bind a service instance to an application
unbind-service	us	Remove the binding between a service instance and an application
service-keys	sks	List all service keys for a specified service instance.
service-key	sk	Displays a service key for a specified service instance.

Command	Alias	Description
create-service-key	csk	Create a service key for a specified service instance.
delete-service-key	dsk	Delete a service key for a specified service instance.
service-brokers		List all available service brokers
create-service-broker		Create a service broker
delete-service-broker		Delete a service broker
update-service-broker		Update a service broker
rename-service-broker		Rename a service broker
create-user-provided-service	cups	Create a user-provided service instance and make it available for use by applications. *
update-user-provided-service	uups	Update the name-values pairs used to define a user-provided service instance
register-service-url		Register a Uniform Resource Locator (URL) for a named service.
unregister-service-url		Unregister a service URL

**i Note**

To access databases not managed by the SAP HANA Service Broker, you can create “user provided services”. To create a user-provided service, you must provide the service credentials for access to the database and bind the service to the application.

## Related Information

[Create a Service Instance \[page 681\]](#)

[The SAP HANA Service Broker \[page 682\]](#)

[XS CLI: Services Management \[page 874\]](#)

## 8.5 Update a Service Instance

Update configuration details of an existing service instance.

### Context

You can modify details of an existing service and make the modified service available to the applications that use it. As part of the update operation, you can perform the following tasks:

- Change the service plan associated with a service
- Modify the service configuration

Configuration details can be provided either in a JSON formatted file or as a JSON-compliant string as an argument on the command line.

To update the configuration of an existing service plan in SAP HANA XS advanced run time, perform the following steps:

### Procedure

1. Log on to the SAP HANA XS advanced run time where the service that you want to update is running.

```
xs login -a https://hanahost.acme.com:30030 -u <Username> -p <Password> -o  
<myorg> -s <myspace>
```

#### Output Code

```
USERNAME: XSMASTER  
PASSWORD>  
ORG: myorg  
SPACE: myspace  
API endpoint: https://hanahost.acme.com:30030 (API version: 1)  
User: Username  
Org: myorg  
Space: myspace
```

2. Check that the service instance you want to update is running.

To display a list of the service instances running in the current organization and space, run the following command in a command shell:

```
xs services
```

SAP HANA XS advanced displays a list of the currently available services:

#### Output Code

```
xs services  
Getting services in org "myorg" / space "SAP" as XSMASTER...  
Found services:
```

name	service	plan	bound apps
hdi-container	hana	hdi-shared	
hdi-schema	hana	schema	
uaa	xsuaa	default	
deploy-service-ss	hana	securestore	deploy-service
deploy-service-database	hana	schema	deploy-service
deploy-service-uaa	xsuaa	default	deploy-service
product-installer-database	hana	schema	product-installer
product-installer-uaa	xsuaa	default	product-installer

3. Update details of an existing service plan.

You can change the service plan associated with an existing service.

### **Restriction**

It is not possible to update the service plan assigned to an instance of the XS User Account and Authentication (XSUAA) service.

To change the service plan associated with a running service, use the `xs update-service` command as follows:

```
xs update-service <SERVICE_INSTANCE> -p <NEW_SERVICE_PLAN>
```

4. Update details of an existing service's configuration.

Some services are configured with a service descriptor. For example, the XSUAA service for user authentication is configured using the security descriptor `xs-security.json`. You can change selected details of the security configuration and update the service with the modified configuration.

To modify details of an existing XSUAA service's security configuration, use the `xs update-service` command as follows:

```
xs update-service <SERVICE_INSTANCE> -p <SERVICE_PLAN> -c xs-security.json
```

### **Restriction**

When updating the configuration of the XSUAA service, you can add or delete any defined scopes, any security-related attributes, and any role templates. However, you can only **modify** the following elements of an existing service's security configuration:

- A security scope's description and "granted applications"
- A security attribute's description  
When updating the XSUAA service, it is not permitted to change the value type of an attribute defined in the `xs-security.json` application security descriptor.
- A role template's description, scope reference, or attribute reference

## Related Information

[Create a Service Instance \[page 681\]](#)

[Setting Up Security Artifacts \[page 757\]](#)

## 8.6 Scheduling Jobs in XS Advanced

The Job Scheduler service enables you to create and schedule long-running operations or jobs.

In the SAP HANA XS advanced model, the Job Scheduler is an application service. The Job Scheduler service enables you to create and schedule long-running operations or jobs. This service is deployed during the installation of the SAP HANA XS advanced model.

The following table lists the sequence of tasks required to use an instance of the Job Scheduler service:

**i Note**

To configure and setup Job Scheduler you require specific roles and permissions.

Step	Task	Role
1	Configure the Service Broker for Job Scheduler	Space Developer
2	Create a Job Scheduler Service Instance	Space Developer
3	Bind an Application to the Job Scheduler Service	Space Developer
4	Maintain jobs and job schedules	Administrator

### Related Information

[Job Scheduler Services in XS Advanced \[page 675\]](#)

[Roles for Running the Job Scheduler \[page 688\]](#)

[Configure the Service Broker for Job Scheduler \[page 687\]](#)

[Bind an Application to the Job Scheduler Service \[page 690\]](#)

[Maintain Jobs and Job Schedules in XS Advanced \[page 691\]](#)

### 8.6.1 Configure the Service Broker for Job Scheduler

You need to configure the service broker to consume the Job Scheduler service.

### Prerequisites

You have the Space Developer role. For more information about Organizations and spaces, see the topic *Maintaining Organizations and Spaces in SAP HANA XS Advanced Model* in the *SAP HANA Administration Guide*.

## Context

A successful installation of the SAP HANA XS advanced model runtime ensures that the Job Scheduler service is deployed.

## Procedure

1. Set the credentials for the Job Scheduler service broker.

```
xs set-env jobscheduler-broker brokeruser <user name>
xs set-env jobscheduler-broker brokerpassword <password>
```

2. Restart the broker.

```
xs restart jobscheduler-broker
```

3. Register the service broker to consume the service.

```
xs create-service-broker <serviceName> <userName> <password> <brokerURL>
```

### ➔ Tip

<userName> and <password> are the ones that were set when registering credentials in the previous step. To check the broker URL, log on to the space where the application is deployed and use the command `xs apps`. In the list of deployed applications, check the URL corresponding to the jobscheduler-broker application.

### 8.6.1.1 Roles for Running the Job Scheduler

You need to have specific roles to work with the Job Scheduler service.

A role defines the permission (scope) assigned to your user. For example, with the Space Developer role you can create a service instance, bind an instance with an application, or unbind a service instance and so on.

The roles and permissions that you require depend on the type of tasks that a user is required to perform. As a technical user, you will receive all the credentials required to work with the Job Scheduler service from `<VCAP_SERVICES>` at runtime. At application runtime, the service provides the credentials required for the application in the `<VCAP_SERVICES>` environment variable to communicate with the service instance. The application code parses the `<VCAP_SERVICES>` variable and extracts the credentials. These credentials are unique for a binding. The binding process generates the technical user. The Space Developer role will enable you to perform all the technical user tasks.

To perform tasks on the Job Scheduler dashboard, you need to have specific roles. You need the SAP HANA administrator role to assign the roles specific to Job Scheduler. To assign roles as a SAP HANA administrator, access the Job Scheduler dashboard from the [XS Advanced Administrator and Monitoring Tools](#). To assign specific roles to the end users of the Job dashboard, use the [Application Role Builder Tool](#). Using the [Application Role Builder Tool](#) to create a new role collection and then add Job Scheduler roles. The Job Scheduler roles are:

- Administration:  
This role enables the end-user to edit jobs and schedules.
- Viewer  
This role enables the end-user to access the user interface in read-only mode.
- Configuration  
This role enables the end-user to maintain the global configurations of the service through the configuration user interface.

For more information about roles and templates, see *Building Roles for SAP HANA XS Advanced Model Applications* in the *SAP HANA Administration Guide*.

## Related Information

[The Job Scheduler Dashboard \[page 711\]](#)

### 8.6.2 Create a Job Scheduler Service Instance

To use a Job Scheduler service, an application requires a Job Scheduler service instance.

#### Prerequisites

- The Job Scheduler service is installed as part of the SAP HANA XS advanced model installation.
- You have registered the service broker.

#### Context

You create a Job Scheduler service instance to use the service.

#### Procedure

1. Check for Job Scheduler service details in a space.

```
xs marketplace
```

2. Create an instance of the Job Scheduler service.

```
xs create-service <service name> <service plan name><service instance name>
```

```
xs create-service jobscheduler default myinstance
```

---

## Related Information

[Configure the Service Broker for Job Scheduler \[page 687\]](#)

### 8.6.3 Bind an Application to the Job Scheduler Service

To use a Job Scheduler service, you have to bind an application with the Job Scheduler service.

#### Prerequisites

- The service broker and the service instance for the Job Scheduler service are available.
- The application required for binding is deployed in the space.

#### Context

You bind an application to the Job Scheduler service after creating a Job Scheduler service instance.

#### Procedure

1. Check if an instance of the Job Scheduler service is available for binding.

Use the command `xs services` to check for Job Scheduler instances.

2. Bind the Job Scheduler service to the application.

```
xs bind-service <application name><service instance name>
```

3. Restart the binding.

```
xs restart <application name>
```

4. Check if the application is bound.

Check the details of the VCAP\_SERVICES in the environment variable.

```
xs env <application name>
```

## Results

### i Note

To unbind a service instance from an application, use the command:

```
xs unbind-service <application name> <service instance>
```

To delete a service instance, use the command:

```
xs delete-service instance
```

## Related Information

[Create a Job Scheduler Service Instance \[page 689\]](#)

## 8.6.4 Maintain Jobs and Job Schedules in XS Advanced

Maintain run time configurations for jobs and job schedules in SAP HANA XS advanced.

### Prerequisites

- The service broker and the service instance for the Job Scheduler service are available.
- The application using the Job Schedule is deployed in the space and bound to the Job Scheduler service instance.
- You have the authorization scope for POST, PUT, and DELETE requests (for example, `jobscheduler.Admin`).
- To access the *Job Scheduler Dashboard*, you must have the authorization scopes defined in the roles grouped together in one of the following role collections:
  - XS\_CONTROLLER\_ADMIN  
Full access: no access restrictions
  - XS\_CONTROLLER\_USER  
Modify and read-only access
  - XS\_CONTROLLER\_AUDITOR  
Read-only access

### ➔ Tip

Role collections can be assigned to an SAP HANA user in SAP HANA studio by means of user parameters, for example, `XS_RC_XS_CONTROLLER_ADMIN` or `XS_RC_XS_CONTROLLER_USER`, or `XS_RC_XS_CONTROLLER_AUDITOR`.

## Context

To maintain jobs and job schedules, you use the Job Scheduler REST APIs (for example, *Job Creation*, *Job Configuration*, or *Job Deletion*) as illustrated in the following examples.

### i Note

The code examples are not always complete; they are intended for illustration purposes only.

## Procedure

1. Create a new job.

Use the *Job Creation* API (`POST /scheduler/jobs`), as illustrated in the example request:

```
POST /scheduler/jobs HTTP/1.1
Host: localhost:4242
Authorization: Basic YWJjOmRlZg==
Content-Type: application/json
Cache-Control: no-cache
{"name": "validateSalesOrder", "description": "cron job that validates sales
order requests", "action": "http://salesOrderApp.hana.acme.com:40023/
salesOrders/validate", "active": true, "httpMethod": "PUT", "schedules": [
{"cron": "* * * * */10", "description": "this schedule runs every 10
seconds", "data": {"salesOrderId": "1234"}, "active": true, "startTime": {
"date": "2015-10-20 04:30 +0000", "format": "YYYY-MM-DD HH:mm Z"}}]}
```

The response to the job-creation request should look like the following example:

```
{"name": "validateSalesOrder", "action": "http://salesOrderApp.hana.acme.com:
40023/salesOrders/
validate", "active": true, "httpMethod": "PUT", "description": "cron job that
validates sales order
requests", "startTime": null, "endTime": null, "signatureVersion": 0, "schedules": [
{"active": true, "startTime": "2015-10-20
04:30:00", "endTime": null, "description": "every 10 seconds, every 2
minutes", "data": {"salesOrderId": "\\"1234\\""}, "cron": "* * * * */10", "type": "recurring", "scheduleId": "cb5c9def-
e2a0-4294-8a51-61e4db373f99"}, "_id": 3}
Headers:
Connection → keep-alive
Content-Length → 468
Content-Type → application/json; charset=utf-8
Date → Mon, 09 Nov 2016 09:08:53 GMT
ETag → W/"1d4-P7BnAm3yordzbrYyJtpalg"
Location → /scheduler/jobs/3
X-Powered-By → Express
```

2. Modify (configure) a new job.

Use the *Job Configuration* API (`PUT /scheduler/jobs`), as illustrated in the example request:

```
PUT /scheduler/jobs/3 HTTP/1.1
Host: localhost:4242
Authorization: Basic YWJjOmRlZg==
Content-Type: application/json
Cache-Control: no-cache
{"active": true, "user": "abc", "password": "def", "httpMethod": "GET"}
```

The response to the job-configuration request should look like the following example:

```
{"success":true}
Headers:
Connection → keep-alive
Content-Length → 16
Content-Type → application/json; charset=utf-8
Date → Mon, 09 Nov 2016 09:30:36 GMT
ETag → W/"10-c2PoX+nt7m8FOksx1YjAhg"
X-Powered-By → Express
```

3. Delete an existing job.

Use the *Job Deletion API* (`DELETE /scheduler/jobs`), as illustrated in the example request:

```
DELETE /scheduler/jobs/4 HTTP/1.1
Host: localhost:4242
Authorization: Basic YWJjOmRlZg==
Content-Type: application/json
Cache-Control: no-cache
```

The response to the job-deletion request should look like the following example:

```
{"success":true}
Headers:
Connection → keep-alive
Content-Length → 16
Content-Type → application/json; charset=utf-8
Date → Mon, 09 Nov 2016 09:30:36 GMT
ETag → W/"10-c2PoX+nt7m8FOksx1YjAhg"
X-Powered-By → Express
```

4. Create a new job schedule.

Use the *Job Schedule Creation API* (`POST /scheduler/jobs/3/schedules`), as illustrated in the example request:

```
POST /scheduler/jobs/3/schedules HTTP/1.1
Host: localhost:4242
Authorization: Basic YWJjOmRlZg==
Content-Type: application/json
Cache-Control: no-cache
{"repeatEvery":"2 hours","data":{"order_id":"abcd"}, "active":true,
"description":"New Schedule", "startTime":{"date": "2016-04-21", "format":
"YYYY-MM-DD"}}
```

The response to the job-schedule creation request should look like the following example:

```
"repeatInterval":"2
hours","repeatAt":null,"time":null,"cron":null,"data":"{\\"order_id\\":\\"abcd
\\\"}","description":"New
Schedule","type":"recurring","active":true,"startTime":"2016-04-21
18:30:00","endTime":null,"jobId":3,"scheduleId":"0e29c67c-563e-4931-
af08-43acb10813e8"}
Headers:
Connection → keep-alive
Content-Length → 274
Content-Type → application/json; charset=utf-8
Date → Mon, 09 Nov 2016 09:42:13 GMT
ETag → W/"112-rdQSXBVY0u6JNI/Wf0I7w"
Location → /scheduler/jobs/3/schedules/0e29c67c-563e-4931-af08-43acb10813e8
X-Powered-By → Express
```

5. Delete an existing job schedule.

Use the *Job Schedule Deletion API* (`DELETE /scheduler/jobs/3/schedules`), as illustrated in the example request:

```
DELETE /scheduler/jobs/4 HTTP/1.1
Host: localhost:4242
Authorization: Basic YWJjOmRlZg==
Content-Type: application/json
Cache-Control: no-cache
```

The response to the job-schedule deletion request should look like the following example:

```
{"success":true}
Headers:
Connection → keep-alive
Content-Length → 16
Content-Type → application/json; charset=utf-8
Date → Mon, 09 Nov 2016 09:51:39 GMT
ETag → W/"10-c2PoX+nt7m8FOksx1YjAhg"
X-Powered-By → Express
```

## Related Information

[Job Scheduler REST API for XS Advanced \[page 694\]](#)

[The Job Scheduler Dashboard \[page 711\]](#)

### 8.6.4.1 Job Scheduler REST API for XS Advanced

The Job Scheduler APIs enable applications to use the functionality provided in Job Scheduler.

The Job Scheduler-as-a-Service is a microservice component, which enables you to create, schedule, and run application tasks. The component exposes REST endpoints for interaction, with JSON as the format for data communication. The Job Scheduler API for SAP HANA XS advanced includes the commands listed in the following table. For more information about the configuration parameters required for the request, see the API documentation provided with the *Job Scheduler Dashboard* tool.

#### i Note

Access to the APIs is controlled by authorization scopes, for example, `admin` for `POST` and `PUT` requests, or `view` for `GET` requests. Scopes are built into roles, which can be assigned to users in role collections. The Job Scheduler REST APIs are protected with basic authentication.

An application, which has been bound to the Job Scheduler service and wants to interact with the Job Scheduler service, must extract the authentication credentials from the `<VCAP_SERVICES>` environment variable and use these credentials to call the REST APIs. To invoke the API, the user-authentication credentials must be encoded and passed in the “Authorization” header. If the credentials are not passed or they are passed wrongly, the APIs return a response with the status code “401 – Unauthorized”.

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

- [Command Overview](#)
- [Human-Readable Dates](#)
- [Time Formats](#)

## Command Overview

Table 96: XS Advanced Job Scheduler REST API

API	Description	Required Scope
<a href="#">Job Creation</a>	Used to create a job. Job creation can accept a collection of job schedules to be created.	admin
<a href="#">Job Configuration</a>	Configure a job with updated run time information. The API can also be used to create a job if a Job with the Job Name in the URI segment, is not found.	admin
<a href="#">Job Deletion</a>	Delete a job and purge all its run time information such as job schedules and logs.	admin
<a href="#">Job Schedule Creation</a>	Create a job schedule for a specified job. All job configuration values (Action URL, HTTP Method, User, Password & Job Activation Status) are valid for the newly created schedule. A job schedule will only run if both the job and the schedule are active.	admin
<a href="#">Job Schedule Modification</a>	Configure the run time information of a job schedule for a specified job. All job configuration values (for example: Action URL, HTTP Method, User, Password, and Job Activation Status) remain valid for the modified schedule.	admin
<a href="#">Job Schedule Deletion</a>	Delete and purge run time information of the job schedule of the specified job. All related information like job schedule configurations and logs are purged. The processing of the schedule is also immediately stopped.	admin
<a href="#">Bulk Job Schedule Activation</a> <a href="#">Bulk Job Schedule Deactivation</a>	This is a utility API used to activate or deactivate all existing schedules of a job. This API triggers the immediate processing (or a halt in processing) of all job schedules for the specified job.	admin
<a href="#">Job Details</a>	Retrieve the saved details and configurations of a specified job. If the <code>displaySchedules</code> parameter is not provided, the schedules for the job are not returned and only the job details are returned.	view
<a href="#">Job Schedule Details</a>	Retrieve the saved details and configurations of a specified job schedule & optionally the generated logs for the schedule.	view

API	Description	Required Scope
<a href="#">Bulk Job Schedule Deletion</a>	Delete and purge run time information of all the currently configured job schedules of the specified job. All related information like job schedule configurations and logs are purged. The processing of the schedules is also immediately stopped.	admin
<a href="#">Job Run Log Update</a>	Used by applications, to inform the Job Scheduler about the status of an asynchronous, long-running job run.	admin

## Job Creation

To create a job schedule, at least one of the fields `repeatAt`, `repeatEvery`, `cron` and `time` must be used. The response from the job creation API is a JSON body with the job details, including the ID of the job.

- **Route**

POST /scheduler/jobs

- **Response**

A JSON body containing the job details, including the ID of the job with status code “201-CREATED”, if the call was successful. A location header with the relative path to the job-details is included in the response.

### Sample Code

```
POST /scheduler/jobs HTTP/1.1
content-type:application/json;charset=utf-8
host: https://scheduler.service.acme.com
content-length: 500
{"name":"hello_world", "description": "greets the world periodically",
 "action":"http://httpbin.org/basic-auth/abc/def", "active": true,
 "httpMethod":"GET", "schedules": [{"repeatEvery":"2 minutes", "description":
 "every 2 minutes, run this schedule", "data":{"time":"abc"}, "active": true},
 {"cron":"* * * * *", "description": "every minute, run this schedule", "data":
 {"time":"abc"}, "active": true}]}]
```

#### Response:

```
Status: 201 CREATED
Location: /scheduler/jobs/109
Content-Type: application/json; charset=utf-8
Body: {"_id":109,"name":"hello_world","description":"greets the world
periodically","action":"http://httpbin.org/basic-auth/abc/
def","active":true,"user":null,"httpMethod":"GET","schedules":
[{"scheduleId":"a66cbddd4-42ce-4c36-b61a-66bc8be6c2d0","description":"every 2
minutes, run this schedule","data":
{"time":"abc"},"type":"recurring","active":true,"startTime":null,"endTime":null
,"repeatInterval":"2 minutes"}, {"scheduleId":"24d6f8f0-d156-4d48-
b678-44d353e700d2","description":"every minute, run this schedule","data":
{"time":"abc"},"type":"recurring","active":true,"startTime":null,"endTime":null
,"cron":"* * * * *"}]}
```

The job schedule creation request is defined with the parameters listed in the following table:

**i** Note

Parameters marked with an asterisk (\*) are mandatory.

Table 97: Job Creation: Request Body Fields

Request Field	Type	Description
name *	String	<p>The unique name of the job to be created</p> <p><b>i</b> Note</p> <p>If a job with the same name for the technical user credentials already exists, the job creation request fails.</p>
description	String	Describes the user-defined job
action *	String	The fully qualified URL endpoint to be called when the job runs, for example: <code>http://host.acme.com/app/call</code>
active	Boolean	Defines if the job should be activated on creation. Allowed values are: <ul style="list-style-type: none"><li>• <code>false</code> (default) The job is in inactive mode on creation</li><li>• <code>true</code> The job is activated on creation</li></ul>
httpMethod	String	The HTTP method to be used to call the end-point URL for the job action . Allowed values are: GET, POST (default), PUT, and DELETE
startTime	Object	The start time for the job. If the start time is specified for the job, the scheduler checks if a start time is provided for the schedule as well. If a start time is provided for the schedule, it is used for determining the start of the schedule run. If no job-schedule start time is defined, the start time for the job is used. The date and time-formats must be specified as strings.
endTime	Object	The end time for the job. If the end time is specified for a job, the scheduler checks if an end time is provided for the schedule as well. If an end time is provided for the schedule, it is used for determining the end of the schedule run. If not, the end time for the job is used. The date and time-formats must be specified as strings.
schedules *	Array	The array of job schedule objects, to be created on job creation.

The `schedules` parameter can be used to provide details of the job schedule (as properties of each job schedule object); the following table lists the permitted properties:

Table 98: Schedule Parameter Fields

Schedule Field	Type	Description
<code>data</code>	object	Optional data to be passed to the job action endpoint when invoked. Typically, the custom data is sent based on the HTTP method configured for invoking the end point URL, for example: <code>{"dataParam": "somevalue"}</code>
<code>time</code>	string or object	For one-time schedules, the parameter denoting the time at which the task executes. A human-readable text can be used to specify the time, for example, "3.30pm" or "tomorrow at 2am". If an object is used, the date and time-formats must be specified as strings.
<code>repeatEvery</code>	string	For recurring schedules, the parameter denoting when the schedule should run. The parameter supports the use of human readable formats.
<code>repeatAt</code>	string	For recurring schedules, the parameter denoting the exact time when the job schedule must run. A human-readable text can be used to denote a specified time, for example, "3.30pm" or "tomorrow at 2am", if the schedule runs repeatedly.
<code>cron</code>	string	For recurring schedules, the parameter denoting the cron pattern. It must be a valid crontab format, for example: <code>"* * * * * /10"</code>
<code>startTime</code>	object	The time when the job scheduling should start. The date and time-formats must be specified as strings.
<code>endTime</code>	object	The time when the job scheduling should end. The date and time-formats must be specified as strings
<code>description</code>	string	The user-provided description of the job schedule

## Job Configuration

Configure a job with updated run time information. The API can also be used to create a job if a Job with the Job Name in the URI segment, is not found. If the API is being used to create a job, the parameters must conform to the same constraints as provided in the Job Creation API

- **Route**

```
PUT /scheduler/jobs/:jobId
PUT /scheduler/jobs/:jobName
```

`:jobId` is the ID of the job previously created using the Job Creation API. If the job name is used in the URI, it is first checked if the job with the name, exists. If no such named job exists, the API tries to create the job. If it does exist, the API configures the job with the details provided in the request body.

## Note

If the API is used to create a job, care must be taken to ensure that the job name in the request URI matches the name of the job in the request body. If the names do not match, an error is returned.

### • Response

If the API finds an existing job, the response has a status code of "200-OK", if the call was successful. The response has a status code of "201-CREATED", if the API is used to create a new job; for new jobs, a location header containing the relative path to the job-details is returned in the response.

## Sample Code

```
PUT /schedule/jobs/5 HTTP/1.1
content-type:application/json; charset=utf-8
host:https://scheduler.service.acme.com
content-length: 500
{"active": true, "user":"abc", "password":"def", "httpMethod": "GET"}
```

```
Response:
status: 200 OK
content-type: application/json; charset=utf-8
{"success": true}
```

## Sample Code

```
PUT /schedule/jobs/jobwhichdoesnotexist HTTP/1.1
content-type:application/json; charset=utf-8
host:https://scheduler.service.acme.com
content-length: 500
{"name":"jobwhichdoesnotexist", "jobDescription": "greets the world
periodically", "action":"http://httpbin.org/basic-auth/abc/def","active":true,
"httpMethod":"GET", "schedules": [{"repeatEvery":"2 minutes",
"scheduleDescription": "every 2 minutes, run this schedule", "data":
{"time":"abc"}, "active": true}, {"cron":"* * * * *", "scheduleDescription":
"every 4 minutes, run this schedule", "data":{"time":"abc"}, "active": false}]}  
Response:
```

```
status: 201 CREATED
content-type: application/json; charset=utf-8
{"_id":120,"name":"jobwhichdoesnotexist","description":"","action":"http://
httpbin.org/basic-auth/abc/
def","active":true,"user":null,"httpMethod":"GET","schedules":
[{"scheduleId":"b373469c-c6d4-4d5f-a002-c56f18455dc5","description":"Default
Schedule","data":
{"time":"abc"},"type":"recurring","active":true,"startTime":null,"endTime":null
,"repeatInterval":"2 minutes"}, {"scheduleId":"2f98471c-26de-4293-ae53-
e4a16e1513f5","description":"Default Schedule","data":
{"time":"abc"},"type":"recurring","active":false,"startTime":null,"endTime":nul
l,"cron":"* * * * *"}]}
```

The job schedule configuration request is defined with the parameters listed in the following table:

Table 99: Job Creation: Request Body Fields

Request Field	Type	Description
active	Boolean	Defines if the job should be activated on configuration. Allowed values are: <ul style="list-style-type: none"><li>• <code>false</code> (default) The job is in inactive mode when configured</li><li>• <code>true</code> The job is active when configured</li></ul>
user	String	The name of the user account to run the configured job
password	String	The password for the user account to run the configured job
httpMethod	String	The HTTP method to be used to call the end-point URL for the job action . Allowed values are: GET, POST (default), PUT, and DELETE
startTime	Object	The start time for the job. If the start time is specified for the job, the scheduler checks if a start time is provided for the schedule as well. If a start time is provided for the schedule, it is used for determining the start of the schedule run. If no job-schedule start time is defined, the start time for the job is used. The date and time-formats must be specified as strings.
endTime	Object	The end time for the job. If the end time is specified for a job, the scheduler checks if an end time is provided for the schedule as well. If an end time is provided for the schedule, it is used for determining the end of the schedule run. If not, the end time for the job is used. The date and time-formats must be specified as strings.

## Job Deletion

Delete a job and purge all its run time information such as job schedules and logs.

- **Route**

```
DELETE /scheduler/jobs/:jobId
```

- **Response**

If the call is successful, the response has a status code “200–OK” and includes a JSON response  
`{"success": true}`.

### Sample Code

```
DELETE /schedule/jobs/:jobId HTTP/1.1
content-type:application/json;charset=utf-8
host:https://scheduler.service.acme.com
```

```
Response: Status: 200 OK
Content-Type: application/json;charset=utf-8
{"success":true}
```

## Job Schedule Creation

Create a job schedule for a specified job. All job configuration values (Action URL, HTTP Method, User, Password & Job Activation Status) are valid for the newly created schedule. A job schedule will only run if both the job and the schedule are active.

- **Route**

```
POST /scheduler/jobs/:jobId/schedules
```

- **Response**

If the call is successful, the response has a status code of "201-CREATED". A location header with the relative path to the schedule-details, is returned in the response.

### Sample Code

```
POST /schedule/jobs/:jobId/schedules HTTP/1.1
content-type:application/json;charset=utf-8
host:https://scheduler.service.acme.com
content-length: 500
{"repeatEvery":"2 hours","data":{"param":"abc"}, "active":true,
"description":"New Schedule", "startTime":{"date": "2015-04-21", "format":
"YYYY-MM-DD"}}
```

Response:

```
Status: 200 OK
Content-Type: application/json; charset=utf-8
{"scheduleId":"0252c255-8f57-4fd5-aa47-c7b344ac9a46", "name":"greet the
world2", "data":{"param":"abc"}, "type":"recurring", "priority":
0, "action":"http://httpbin.org/basic-auth/abc/
def", "nextRunAt":"2015-04-23T03:04:32.856Z", "startTime":"2015-04-20T18:30:00.00
0Z", "endTime":null, "repeatInterval":"2 hours", "active":true, "description":"New
Schedule", "jobId":132}
```

Table 100: Job Schedule Creation Parameters

Request Field	Type	Description
time	string or object	For one-time schedules, the parameter denoting the time at which the task executes. A human-readable text can be used to specify the time, for example, "3.30pm" or "tomorrow at 2am". If an object is used, the date and time-formats must be specified as strings.
repeatEvery	string	For recurring schedules, the parameter denoting when the schedule should run. The parameter supports the use of human readable formats.
repeatAt	string	For recurring schedules, the parameter denoting the exact time when the job schedule must run. A human-readable text can be used to denote a specified time, for example, "3.30pm" or "tomorrow at 2am", if the schedule runs repeatedly.
cron	string	For recurring schedules, the parameter denoting the cron pattern. It must be a valid crontab format, for example: "* * * * */10"

Request Field	Type	Description
data	object	The parameter denoting optional data to be passed to the job action endpoint when invoked. Typically, the custom data is sent based on the HTTP method configured for invoking the end point URL, for example: { "dataParam": "somevalue" }
startTime	object	The time when the job scheduling should start. The date and time-formats must be specified as strings.
endTime	object	The time when the job scheduling should end. The date and time-formats must be specified as strings
active	Boolean	Defines if the job should be activated on configuration. Allowed values are: <ul style="list-style-type: none"> <li>• <code>false</code> (default) The job is in inactive mode when configured</li> <li>• <code>true</code> The job is active when configured</li> </ul>
description	string	The user-provided description of the job schedule

## Job Schedule Modification

Configure the run time information of a job schedule for a specified job. All job configuration values (for example: Action URL, HTTP Method, User, Password, and Job Activation Status) remain valid for the modified schedule.

- **Route**  
`PUT /scheduler/jobs/:jobId/schedules/:scheduleId`
- **Response**  
 If the call is successful, the response has a status code of 200– OK.

Calling this API stops further scheduling of the previously configured job schedule and, if activated, the processing for the newly configured schedule is started. This API cannot be used to change the scheduling mode for the job schedule. For example, if the schedule was created as a recurring “cron”-type schedule, it cannot be changed to a “repeatEvery”-type schedule. However, existing schedule values can be changed.

### Sample Code

```
PUT /schedule/jobs/:jobId/schedules/:scheduleId HTTP/1.1
content-type:application/json;charset=utf-8
host:https://scheduler.service.acme.com
content-length: 500
{"description": "Edited Schedule", "startTime": {"date": "2013-02-08
09:30:26.123"}, "endTime": {"date": "2015-06-08 09:30:26.123"}, "active":
true, "cron": "* * * * *"}
```

**Response:**  
 Status: 200 OK  
 Content-Type: application/json; charset=utf-8

```
{"scheduleId": "80e23846-734e-4b4b-a130-159a492ec482", "name": "greet the world3", "data": {"time": "abc"}, "type": "recurring", "priority": 0, "action": "http://httpbin.org/basic-auth/abc/def", "nextRunAt": "2015-04-23T03:58:21.358Z", "startTime": "2013-02-08T04:00:26.12Z", "endTime": "2015-06-08T04:00:26.123Z", "active": true, "description": "Edited Schedule", "jobId": "136", "cron": "* * * * *"}}
```

Table 101: Job Schedule Modification Parameters

Request Field	Type	Description
time	string or object	For one-time schedules, the parameter denoting the time at which the task executes. A human-readable text can be used to specify the time, for example, "3.30pm" or "tomorrow at 2am". If an object is used, the date and time-formats must be specified as strings.
repeatEvery	string	For recurring schedules, the parameter denoting when the schedule should run. The parameter supports the use of human readable formats.
repeatAt	string	For recurring schedules, the parameter denoting the exact time when the job schedule must run. A human-readable text can be used to denote a specified time, for example, "3.30pm" or "tomorrow at 2am", if the schedule runs repeatedly.
cron	string	For recurring schedules, the parameter denoting the cron pattern. It must be a valid crontab format, for example: "* * * * * /10"
data	object	The parameter denoting optional data to be passed to the job action endpoint when invoked. Typically, the custom data is sent based on the HTTP method configured for invoking the end point URL, for example: { "dataParam": "somevalue" }
startTime	object	The time when the job scheduling should start. The date and time-formats must be specified as strings.
endTime	object	The time when the job scheduling should end. The date and time-formats must be specified as strings
active	Boolean	Defines if the job should be activated on configuration. Allowed values are: <ul style="list-style-type: none"> <li>• false (default) The job is in inactive mode when configured</li> <li>• true The job is active when configured</li> </ul>
description	string	The user-provided description of the job schedule

## Job Schedule Deletion

Delete and purge run time information of the job schedule of the specified job. All related information like job schedule configurations and logs are purged. The processing of the schedule is also immediately stopped.

## Caution

This API removes all the run time configuration information of the job schedule, irrespective of whether the schedule is active or not.

- **Route**

```
DELETE /scheduler/jobs/:jobId/schedules/:scheduleId
```

- **Response**

If the call is successful, the response has a status code, "200-OK" and includes a JSON response

```
{"success": true}
```

## Sample Code

```
DELETE /scheduler/jobs/:jobId/schedules/:scheduleId HTTP/1.1
content-type:application/json;charset=utf-8
host:https://scheduler.service.acme.com
```

```
Response: {"success": true}
Status Code: 200 OK
```

## Bulk Job Schedule Activation/Deactivation

This is a utility API used to activate or deactivate all existing schedules of a job.

- **Route**

```
POST /scheduler/jobs/:jobId/schedules/activationStatus
```

- **Response**

If the call is successful, the response has a status code, "200-OK" and includes a JSON response

```
{"success": true}
```

## Sample Code

```
POST /scheduler/jobs/:jobId/schedules/activationStatus HTTP/1.1
content-type:application/json;charset=utf-8
host:https://scheduler.service.acme.com
{"activationStatus": true}
```

```
Response: {"success": true}
Status Code: 200 OK
```

Table 102: Bulk Job Schedule Activation Parameters

Request Field	Type	Description
activationStatus	Boolean	<p>The desired activation status of the job schedules for the job. Allowed values for the activation status are:</p> <ul style="list-style-type: none"> <li>• <code>false</code> (default) All job schedules for the specified job should be <b>deactivated</b></li> <li>• <code>true</code> All job schedules for the specified job should be activated</li> </ul>

## Job Details

Retrieve the saved details and configurations of a specified job.

- **Route**

```
GET /scheduler/jobs/:jobId?displaySchedules=true
GET /scheduler/jobs?jobId=:jobId&displaySchedules=true Route
GET /scheduler/jobs?name=:jobName&displaySchedules=true
```

- **Response**

If the call is successful, the response has a status code, "200-OK" and includes a JSON response with the schedule details, for example: `{"schedules": [{"data": {"time": "abc"}, "type": "recurring", "repeatInterval": "2 minutes", "active": false, "startTime": null, "endTime": null, "repeatAt": null, "scheduleId": "0d3b4cc1-0f7b-4ee6-ab12-63d474b900f2", "description": "Default Schedule"}, {"data": {"time": "abc"}, "type": "recurring", "cron": "* * * * *", "active": false, "startTime": null, "endTime": null, "repeatAt": null, "scheduleId": "1b1bb70f-cada-46c9-9974-a7a1b87ba24f", "description": "Default Schedule"}], "name": "greet the world2", "description": "", "action": "http://httpbin.org/basic-auth/abc/def", "user": null, "httpMethod": "GET", "active": false, "_id": 111}`.

### Sample Code

```
GET /scheduler/jobs/:jobId?displaySchedules=true HTTP/1.1
content-type:application/json; charset=utf-8
host:https://scheduler.service.acme.com
```

**Response:**

```
Status: 200 OK
Content-type: application/json; charset=utf-8
{"schedules": [{"data": {"time": "abc"}, "type": "recurring", "repeatInterval": "2 minutes", "active": false, "startTime": null, "endTime": null, "repeatAt": null, "scheduleId": "0d3b4cc1-0f7b-4ee6-ab12-63d474b900f2", "description": "Default Schedule"}, {"data": {"time": "abc"}, "type": "recurring", "cron": "* * * * *", "active": false, "startTime": null, "endTime": null, "repeatAt": null, "scheduleId": "1b1bb70f-cada-46c9-9974-a7a1b87ba24f", "description": "Default Schedule"}], "name": "greet the world2", "description": "", "action": "http://httpbin.org/basic-auth/abc/def", "user": null, "httpMethod": "GET", "active": false, "_id": 111}
```

Table 103: Job Details Parameters

Request Field	Type	Description
displaySchedules	Boolean	<p>Display details of the job schedules for the job. Allowed values for the job details are:</p> <ul style="list-style-type: none"> <li>• <code>false</code> Do <b>not</b> display details of job schedules for the specified job</li> <li>• <code>true</code> Display details of job schedules for the specified job</li> </ul>
jobId	String	The job ID needed to query for the job details. This can be passed as a URI segment parameter or as a query parameter.
name	String	The job name needed to query the job details. This can be passed as a query parameter

## Job Schedule Details

Retrieve the saved details and configurations of a specified job schedule & optionally the generated logs for the schedule. Either `:jobId` or `:name` is required to invoke this API. If `displayLogs` is not provided, the logs for the schedule are not returned and only the schedule details are returned.

- **Route**

```
GET /scheduler/jobs/:jobId/schedules/:scheduleId?displayLogs=true
```

- **Response**

If the call is successful, the response has a status code, "200-OK" and includes a JSON response with the schedule details, for example: {"data":

```
{"time":"abc","type":"recurring","repeatInterval":"2
minutes","plannedTime":"2015-04-19T15:12:44.000Z","active":true,"startTime":null
,"endTime":null,"repeatAt":null, [...]}.
```

### Sample Code

```
GET /scheduler/jobs/112/schedules/550d1b96-8002-4d0d-850e-368aaa591671?
displayLogs=true
HTTP/1.1 content-type:application/json; charset=utf-8
host:https://scheduler.service.acme.com
```

**Response:**

```
Status: 200 OK
Content-Type: application/json; charset=utf-8
{"data":{"time":"abc","type":"recurring","repeatInterval":"2
minutes","plannedTime":"2015-04-19T15:12:44.000Z","active":true,"startTime":nul
l,"endTime":null,"repeatAt":null,"logs":
[{"text":null,"httpStatus":null,"executionTime":null,"status":"SCHEDULED","sche
duleTime":"2015-04-19T15:10:53.000Z","completionTime":null}],"scheduleId":"550d
1b96-8002-4d0d-850e-368aaa591671","description":"Default Schedule"}
```

Table 104: Job Schedule Details Parameters

Request Parameter	Type	Description
displayLogs	Boolean	Controls whether the API should return ( <code>true</code> ) all the generated logs for the job schedule or not ( <code>false</code> )

## Bulk Job Schedule Deletion

Delete and purge run time information of all the currently configured job schedules of the specified job. All related information like job schedule configurations and logs are purged. The processing of the schedules is also immediately stopped.

### ⚠ Caution

This API removes all the run time configuration information of the job schedule, irrespective of whether the schedule is active or not.

- **Route**

```
DELETE /scheduler/jobs/:jobId/schedules
```

- **Response**

If the call is successful, the response has a status code, "200-OK" and includes a JSON response  
`{"success": true}`.

### ☰ Sample Code

```
DELETE /scheduler/jobs/:jobId/schedules HTTP/1.1
content-type:application/json; charset=utf-8
host:https://scheduler.service.acme.com
```

```
Response: {"success": true}
Status Code: 200 OK
```

## Job Run Log Update

Inform the Job Scheduler about the status of an asynchronous, long-running job run. This API must be invoked by the application after the asynchronous execution of the job has completed, with the status of the job run and optionally some text about the job execution.

### ⚠ Caution

This API must be invoked by the application after the **asynchronous** execution of the job has completed, with the status of the job run and optionally some text about the job execution.

- **Route**

```
PUT /scheduler/jobs/:jobId/schedules/:scheduleId/runs/:runId
```

### **i** Note

Parameters marked with an asterisk (\*) are mandatory.

Table 105: Job Run Log Update Parameters

Request Parameter	Type	Description
success *	Boolean	Indicates that the job run was successful ( <code>true</code> ) or failed ( <code>false</code> )
message	String	Additional log/text about the job run

## Human Readable Dates

The job scheduler for XS advanced supports human readable dates and ranges for the parameters `time`, `repeatAt` and `repeatEvery`, which are used for configuring job schedules. The job scheduler uses an embedded English language date parser for this facility. Valid human readable strings for the parameters are shown below:

### **i** Note

The date parser expects a valid readable string; invalid strings will either throw parser errors or cause the job scheduling to happen inconsistently.

Table 106: Date and Time Parameters

Parameter	Comments	Examples
<code>time</code>	Designates a particular timestamp for running a job schedule. If an invalid string is provided, the scheduler falls back to the current timestamp and runs the schedule immediately. The following example strings are valid for the <code>time</code> parameter:	"10 hours from now" "20 minutes from now" "in 2 hours" "tomorrow at 4pm" "next week monday at 5am" "9pm tonight" "3.30pm"

Parameter	Comments	Examples
repeatAt	<p>Represents a convenient way to create daily timestamp-based schedules. The string should designate a particular timestamp for repeatedly running a job schedule. This follows the same pattern as the recommendations for the "time" parameter, barring a few discrepancies. While the text for the "time" parameter must denote something concrete and in the future, the 'repeatAt' must designate a timestamp, which is valid and constant daily. If an invalid string is used, the scheduler falls back to the current timestamp and runs the schedule immediately.</p> <div style="background-color: #f9e79f; padding: 10px;"> <p><b>i Note</b></p> <p>Second-based precision can sometimes be inaccurately timed; timezones must be specified using the offset (in hours), for example, "+07:00"</p> </div>	<p>"4.40pm"          "18.40"          "6.20am"          "17.20:30 "          "09:30:26.123+07:00"</p>
repeatInterval	<p>The string should designate a interval to repeat the job execution. Word strings for denoting the numeric value are not supported yet. For example, for "twenty minutes", use "20 minutes" to denote the interval. Supported time-units for this parameter are "years", "months", "weeks", "days", "hours", "minutes", "seconds".</p>	<p>"10 hours "          "2 days "          "3 seconds"</p>

## Date and Time Formats in Job Schedule Parameters

The date-time parameters for job schedules (for example, `startTime`, `endTime`, and `time`) can be passed as objects, with the mandatory `date` field denoting the date as a string and an optional `format` field denoting a date-time format for correctly parsing the user-provided date value. If the parameters are passed as strings, they must be valid date representations, in either the ISO-8601 or IETF-compliant RFC 2822 formats. For object representations, the following rules apply:

- **Date field as input**

If only the date field is provided as input, the string is checked against both IETF-compliant RFC 2822 time stamps and ISO-8601. If the date string is of an unknown format, the parser displays an error. For ISO-8601 compliant dates, calendar dates (for example, "2013-02-08"), week dates ("2013-W06-5"), ordinal dates ("2013-039") and time-based dates ("2013-02-08 09+07:00") are all supported.

- **Date string format**

If the format of the date string is customized, an optional format string can be passed. The allowed parsing tokens are as described in the following table:

Table 107: Date and Time Parameters

Input Token	Example	Description
YYYY	2014	4 digit year

Input Token	Example	Description
YY	14	2 digit year
Q	1-4	Quarter of year. Sets month to first month in quarter
M MM	1-12	Month number
MMM MMMM	January- Dec	Month name in locale
D DD 1- 31		Day of month
Do	1st- 31st	Day of month with ordinal
DDD DDDD	1-365	Day of year
X	1410715640.579	Unix Timestamp
x	1410715640579	Unix Timestamp (ms)
gggg	2015	Locale 4 digit week year
gg	15	Locale 2 digit week year
w ww	1- 53	Locale week of year
e	1-7	Locale day of week
GGGG	2015	ISO 4-digit week year
GG	15	ISO 2-digit week year
W WW	1- 53	ISO week of year
E	1-7	ISO day of week
H HH	0 -23	24 Hour Time
h hh	1 -12	12 hour time used with 'a A'
a A	am pm	Post or ante meridiem
m mm	0 -59	Minutes
s ss	0 -59	Seconds
S	0 -9	Tenths of a second
SS	0 -99	Hundredths of a second
SSS	0 -999	Thousandths of a second
Z ZZ	+12:00	Offset from UTC as +HH:mm, +-HHmm, or Z

## Date-Time Format Examples

- startTime  
"startTime": {"date": "2015-10-20 4:30 +0000", "format": "YYYY-MM-DD HH:mm Z"}  
4.30 UTC on 20th Oct 2015
- endTime  
"endTime": {"date": "2015-W06-5"}  
Friday, February 06, 2015
- time  
"time": {"date": "2010-10-20 4:30", "format": "YYYY-MM-DD HH:mm"}  
4.30 Local Time (the timezone for the scheduler service is considered here)

## Related Information

[Scheduling Jobs in XS Advanced \[page 687\]](#)

### 8.6.4.2 The Job Scheduler Dashboard

The Job Scheduler dashboard enables you to manage jobs for a service instance.

The dashboard lists the available jobs. Choose a job to create a schedule or to view existing schedules.

## How to access the Job Scheduler Dashboard

### 1. Get Dashboard URL

To access the Job Scheduler dashboard, you can SSH to the SAP XS Advanced server and perform the following steps:

1. List the applications running on the server using the command: `xs apps`.
2. Under the *URL* column, identify the URL specific for the relevant application ("jobscheduler-dashboard" for Job Scheduler).
3. Copy the URL to any Web browser to launch the application.

#### i Note

If a valid certificate is not available, the Web browser indicates an issue with the certificate. To resolve the issue, add the required certificate.

4. Provide the log on credentials to access the application.

### 2. Permission to Access the Dashboard

You need specific roles to access Job Scheduler. For more information, see *Roles for Running the Job Scheduler* in the related section. The SAP HANA Administrator creates a Role Collection and adds the Job Scheduler roles to the Role Collection. For more information, see *Maintaining the SAP HANA XS Advanced Model Run Time* in the *SAP HANA Administration Guide*.

The various screens available on the dashboard and their description are as below:

- **Configuration**

This screen enables you to maintain global configuration required for functioning of a specific Job Scheduler service instance.

- **Max. Invocation Attempts:** This value indicates the number of attempts made by Job Scheduler to reach job action endpoint before deactivating a job. If Job Scheduler consecutively fails to reach the endpoint, it sets the job inactive. The default value of this parameter is 3.
- **Asynchronous Execution timeout (ms):** This value indicates the duration (in milliseconds) that the Job Scheduler awaits response for the asynchronous job from the application endpoint. If the application does not provide a response in the specified duration, the run status is set to COMPLETED/ UNKNOWN.

- **Jobs**

This screen lists all the jobs created for a specific service instance. You can delete a job or navigate to the job details by choosing the job name.

- **Overview**

This screen appears if you select a job from the list of jobs on the [Jobs](#) screen. It displays the details of the selected job. You can edit a job.

- **Schedules**

This screen enables you to create and configure schedules for a job. To access schedules, choose a job listed on the dashboard. You will see the [Schedules](#) screen. For more information about schedules, see *Scheduling Jobs in XS Advanced* in the related information section. Choose a schedule to see the history and logs corresponding to the schedule. To display run logs of schedule, choose [Logs](#).

- **Action History**

This screen maintains history of a job or the schedules for a specific job.

➔ Tip

To access the [Job Scheduler Dashboard](#), you must have the authorization scopes defined in the roles grouped together in one of the default role collections, for example, [XS\\_CONTROLLER\\_ADMIN](#), or [XS\\_CONTROLLER\\_USER](#). Role collections can be assigned to an SAP HANA user in SAP HANA studio by means of user parameters, for example, `XS_RC_XS_CONTROLLER_ADMIN` or `XS_RC_XS_CONTROLLER_USER`, or `XS_RC_XS_CONTROLLER_AUDITOR`.

## Related Information

[Roles for Running the Job Scheduler \[page 688\]](#)

# 9 Creating the Client User Interface

Create the user interface (GUI) to display the database content (HTML pages).

The application module for the client user interface (`web/` in the example below) also contains the application-route description (`xs-app.json`), which is used by the application router module. You set up the application routes description in a subsequent step.

## Sample Code

```
JavaScript-AppName
|- db/
|   |- package.json
|   |- src/
|       |- .hdiconfig
|       \- mytable.hdbdd
|- web/                               # Application descriptors
|   |- xs-app.json                    # Application routes configuration
|   |- package.json                  # Application router details/dependencies
|   \- resources/                     # Static resources
|- js/
|   |- start.js
|   |- package.json
|   \- src/
|       \- odata/
|           |- resources/
|           \- services/
|               |- srv1.xsodata
|               \- srvN.xsodata
|- security/
|   \- xs-security.json
\- mtad.yaml
```

## 9.1 Building User Interfaces with SAPUI5 for SAP HANA

*UI development toolkit for HTML5* (SAPUI5) is a user interface technology that is used to build and adapt client applications based on SAP HANA. You can install SAPUI5 in the SAP HANA studio to build user interfaces delivered by SAP HANA's Web server.

The SAPUI5 runtime is a client-side HTML5 rendering library with a rich set of standard and extension controls. It provides a lightweight programming model for desktop as well as mobile applications. Based on JavaScript, it supports Rich Internet Applications (RIA) such as client-side features. SAPUI5 complies with OpenAjax and can be used with standard JavaScript libraries.

## SAPUI5 Demo Kit

The SAPUI5 Demo Kit contains:

- A **Developer Guide** which contains information about the programming languages used, open source technology, development tools, and APIs
- An **Explored app**, which provides a detailed view of almost every control including detailed information about the properties, aggregations, events, and methods. The SAPUI5 Developer Guide also provides running samples including a code view, from which you can easily copy the required code snippets
- An **API reference** with JavaScript documentation for the Framework and Control API
- **Demo Apps** with includes and showcases real samples
- **Icons** with an overview of all icons included with SAPUI5.

## Related Information

[SAPUI5 Demo Kit \(version 1.32.7\)](#)

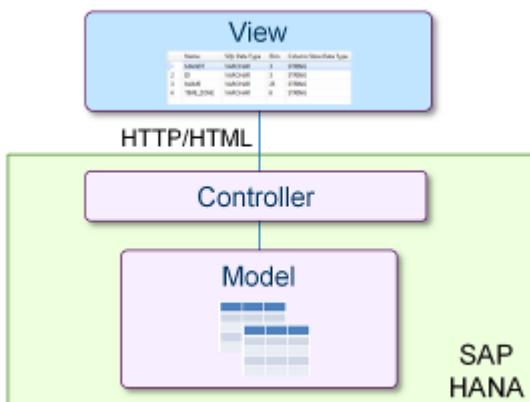
## 9.2 Consuming Data and Services with SAPUI5 for SAP HANA

SAP HANA Extended Application Services (SAP HANA XS) can be used to expose the database data model, with its tables, views and database procedures, to UI clients.

You can expose an SAP HANA model using OData services or by writing native server-side JavaScript code that runs in the SAP HANA context. You can also use SAP HANA XS to build dynamic HTML5 client applications, for example, using SAPUI5 for SAP HANA.

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

- View  
UI rendering occurs completely in the client (SAPUI5, browser, mobile applications)
- Controller  
Procedural (control-flow) logic is defined in (XS) JavaScript, SQLScript or an OData service
- Model  
All application artifacts are stored in SAP HANA



**Figure 4: SAP HANA Application Development with SAP HANA XS**

Each of the levels illustrated in the graphic (view, control, model) is manifested in a particular technology and dedicated languages. After you have defined the data model with design-time artifacts and the equivalent runtime objects, you develop the control-flow logic to expose the data, for example, using server-side JavaScript or an OData service. With the data model and control-flow logic in place, you can build the presentation logic to view the exposed data in a UI client application using SAPUI5 for SAP HANA. For example, you can use an SAPUI5 client to request and display data exposed by an OData service; the UI could include buttons that trigger operations performed by SAP HANA XS JavaScript service; and the data displayed is retrieved from data end points defined in your data model, for example, SQLScript or CDS.

## Related Information

[SAPUI5 Demo Kit \(version 1.32.7\)](#)

# 10 Maintaining XS Advanced Application Routes and Destinations

Define application routes and destinations.

In XS Advanced Programming Model, the application router (“approuter”) is the entry point for a business application and is used for routing incoming requests to the appropriate microservices. The application routes are defined in the application descriptor `xs-app.json`, a JSON-formatted file that is mandatory for the approuter. In the following example, the approuter is located in the `web/` directory, so the `xs-app.json` application descriptor must be placed there, too.

## Sample Code

### Application Routes Configuration

```
JavaScript-AppName
|- db/
|  |- package.json
|  |- src/
|  |  |- .hdiconfig
|  |  \- mytable.hdbdd
|- web/                               # Application descriptors
|  |- xs-app.json                      # Application routes configuration
|  |- package.json                     # Application router details/dependencies
|  \- resources/
|- js/
|  |- start.js
|  |- package.json
|  \- src/
|     \- odata/
|        |- resources/
|        \- services/
|           |- srv1.xsodata
|           \- srvN.xsodata
|- security/
|  \- xs-security.json
|- manifest.yml
\- mtad.yaml
```

## Related Information

[Configure the XS Advanced Application Router \[page 717\]](#)

[XS Advanced Application-Router Resource Files \[page 720\]](#)

## 10.1 Configure the XS Advanced Application Router

The application routes to the available microservices are described in the `xs-app.json` file.

### Prerequisites

- File name
  - `xs-app.json`  
Configures the application router and describes the routes.

#### i Note

Without the application router configuration, no access to the dependent application is possible.

- `package.json`  
Contains the start command for the application router and a list of package dependencies.
- Location  
Application's Web resources folder, for example, `/path/appname/web`
- Format  
JavaScript Object Notation (JSON)

### Context

The application router is used to serve static content, authenticate users, rewrite URLs, and forward or proxy requests to other micro services while propagating user information. The application-router configuration file `xs-app.json` configures the application router and describes the routes. To configure the application router, perform the following steps:

### Procedure

1. Create the application resource-file structure.

For example, `/path/appname/`

2. Create a sub-folder for the static Web-resources module.

The sub-folder for the Web-resources module must be located in the application root folder, for example, `/path/<myAppName>/web`

#### ➔ Tip

The Web-resource module uses `@sap/approuter` as a dependency; the Web-resources module also contains the configuration and static resources for the application.

3. Create sub-folder for the application's static resources.

Static resources can include the following file and components: `index.html`, style sheets (`.css` files), images and icons, etc.). Typically, static resources for a Web application are placed in a subfolder of the Web module, for example, `/path/<myAppName>/web/resources`.

4. Create the application-router configuration file.

The application router configuration file `xs-app.json` must be located in the application's Web-resources folder, for example, `/path/<appname>/web`.

#### Sample Code

```
<myAppName>
| - web/                                # Application descriptors
|   | - xs-app.json                      # Application routes configuration
|   | - package.json                     # Application router details/dependencies
|   \- resources/
```

5. Define details of the application's routes, destinations, and security scopes.

The contents of the `xs-app.json` file must use the required JSON syntax. For more information, see [XS Advanced Application Router Configuration Syntax](#) in *Related Information* below.

- a. Create the required "destinations configuration".

#### Sample Code

```
<myAppName>/web/xs-app.json
```

```
{
  "welcomeFile": "index.html",
  "routes": [
    {
      "source": "/sap/ui5/1(.*)",
      "target": "$1",
      "localDir": "sapui5"
    },
    {
      "source": "/rest/addressbook/testdataDestructor",
      "destination": "backend",
      "scope": "node-hello-world.Delete"
    },
    {
      "source": "/rest/.*",
      "destination": "backend"
    },
    {
      "source": "^/(.*)",
      "localDir": "resources"
    }
  ]
}
```

- b. Add the routes (destinations) for the specific application (for example, `node-hello-world`) to the `env` section of the application's deployment manifest (`manifest.yml`).

Every route configuration that forwards requests to a micro service has property `destination`. The `destination` is a name that refers to the same name in the `destinations` configuration. The `destinations` configuration is specified in an environment variable passed to the approuter application.

## Sample Code

```
<myAppName>/manifest.yml

- name: node-hello-world
  host: myHost-node-hello-world
  domain: xsapps.acme.ondemand.com
  memory: 100M
  path: web
  env:
    destinations: >
      [
        {
          "name": "backend",
          "url": "https://myHost-node-hello-world-
backend.ondemand.acme.com",
          "forwardAuthToken": true
        }
      ]
]
```

6. Add a package descriptor for the XS advanced application router.

The package descriptor (`package.json`) describes the prerequisites and dependencies for the application and starts the application router, too. Add a `package.json` file to the root folder of your application's Web resources module (`web/`).

## Sample Code

```
<myAppName>
|- web/
|   |- xs-app.json
|   |- package.json
|   \- resources/
```

# Application descriptors  
# Application routes configuration  
**# Application router details/dependencies**

The basic `package.json` file for your Web-resources module (`web/`) should look similar to the following example:

## Sample Code

```
{
  "name": "node-hello-world-approuter",
  "dependencies": {
    "@sap/approuter": "^2.7.1"
  },
  "scripts": {
    "start": "node node_modules/@sap/approuter/approuter.js"
  }
}
```

### ➔ Tip

The start script (for example, `approuter.js`) is mandatory; the start script is executed after application deployment.

## Related Information

[XS Advanced Application Routes and Destinations \[page 739\]](#)

[XS Advanced Application-Router Resource Files \[page 720\]](#)

[The XS Advanced Application Descriptor \[page 721\]](#)

[XS Advanced Application Router Configuration Syntax \[page 725\]](#)

### 10.1.1 XS Advanced Application-Router Resource Files

The routing configuration for an application is defined in one or more so-called "destinations" that are defined in a destinations configuration.

The application router is used to serve static content, authenticate users, rewrite URLs, and forward (or proxy) requests to other micro services while propagating user information. The following table lists the configuration files used to define routes for XS advanced, multi-target applications:

Table 108: XS Advanced Application-Router Resource Files Overview

File	Description	Mandatory
package.json	The package descriptor is used by the node.js package manager (npm) to start the application router; in the "dependencies": {} section	Yes
xs-app.json	The application descriptor contains the configuration used by the application router (for example, destinations for request forwarding)	Yes
resources/	A folder that contains all static resources which should be served by the application router. If no resources / folder is present, the application router will not serve any static content. However, it still forwards requests to the configured destinations.  <b>Tip</b> Static resources can be stored in multiple folders, and the folder name "resources /" is optional.	No
default-services.json	Defines the configuration for one or more special User Account and Authentication (UAA) services for local development; this is typically used for testing in local environments.	-

## Related Information

[Configure the XS Advanced Application Router \[page 717\]](#)

## 10.1.2 The XS Advanced Application Descriptor

Understand the contents of the file used to configure the XS advanced application router.

The application descriptor is a file that contains the configuration information used by the application router. The file is named `xs-app.json` and its content is formatted according to JavaScript object Notation (JSON) rules.

### Headers

In many cases back-end nodes need to respond to client requests with URLs. Since micro services are accessed through the application router component, the clients always request the application router's URL. The back end nodes need to know the actual request URL in order to build URLs that are accessible by the client. If back-end nodes respond to client requests with URLs, these URLs need to be accessible to the client. For this reason, the application router passes the following `x-forwarding-*` headers to the client:

- `x-forwarded-host`  
Contains the **host** header value that was sent by the client to the application router
- `x-forwarded-proto`  
Contains the **protocol** that was used by the client to connect to the application router
- `x-forwarded-for`  
Contains the **address** of the client which connects to the application router
- `x-forwarded-path`  
Contains the original **path** which was requested by the client from the approuter

#### Caution

When the application router forwards a request to a destination, it blocks the header `host`.

"Hop-by-hop" headers are meaningful only for a single transport-level connection; these headers are not forwarded by the application router. The following headers are classified as "Hop-By-Hop" headers:

- Connection
- Keep-Alive
- Public
- Proxy-Authenticate
- Transfer-Encoding
- Upgrade

You can configure the application router to send additional HTTP headers, for example, by using the `httpHeaders` environment variable.

### Sessions

The XS advanced model's application router establishes a session with the client (browser) using a session cookie. The application router intercepts all session cookies sent by back-end services and stores them in its

own session. To prevent collisions between the various session cookies, back-end session cookies are not sent to the client. On request, the application router sends the cookies back to the respective back-end services so the services can establish their own sessions.

### i Note

Non-session cookies from back-end services **are** forwarded to the client, which might cause collisions between cookies. Applications should be able to handle cookie collisions.

## Session Contents

A session established by the XS advanced application router typically contains the following elements:

- Redirect location  
The location to redirect to after login; if the request is redirected to a UAA login form, the original request URL is stored in the session so that, after successful authentication, the user is redirected back to it.
- CSRF token  
The CSRF token value if it was requested by the clients. For more information about protection against Cross Site Request Forgery see [CSRF Protection \[page 723\]](#) below.
- OAuth token  
The JSON Web Token (JWT) fetched from the User Account and Authentication service (UAA) and forwarded to back-end services in the `Authorization` header. The client never receives this token. The application router refreshes the JWT automatically before it expires (if the session is still valid). By default, this routine is triggered 5 minutes before the expiration of the JWT, but it can also be configured with the `<JWT_REFRESH>` environment variable (the value is set in minutes). If `<JWT_REFRESH>` is set to 0, the refresh action is disabled.
- OAuth scopes  
The scopes owned by the current user, which is used to check if the user has the authorizations required for each request
- Back-end session cookies  
All session cookies sent by back-end services

## Scaling

The application router keeps all established sessions in local memory, and if multiple instances of the application router are running, there is no synchronization between the sessions. To scale the application router for multiple instances, session stickiness must be enabled so that each HTTP session is handled by the same application router instance. In SAP HANA XS advanced model's on-premise run time, session stickiness is enabled, if the SAP Web Dispatcher is used as a router. In the more recent versions of the SAP HANA XS Advanced run time, the SAP Web Dispatcher is used as a router by default.

## Sizing and Memory Configuration

The application-router process should run with at least 256MB memory. The amount of memory actually required depends on the application the router is serving. The following aspects have an influence on the application's memory usage:

- Concurrent connections
- Active sessions
- Size of the Java Web Token
- Back-end session cookies

If the application router is running in an environment with limited memory, set the heap limit to about 75% of available memory. For example, if the application router memory is limited to 256MB, add the following command to your `package.json`:

### Sample Code

Set Application Router Memory Usage in `package.json` File

```
"scripts": {  
  "start": "node --max-old-space-size=192 node_modules/@sap/approuter/  
  approuter.js"  
}
```

### Note

For detailed information about memory consumption in different scenarios, see the Sizing Guide for the Application Router located in `approuter/approuter.js/doc/sizingGuide.md`

## CSRF Protection

The application router enables CSRF protection for any HTTP method that is not `GET` or `HEAD` and the route is not public. A path is considered public, if it does not require authentication. This is the case for routes with `authenticationType: none` or if authentication is disabled completely via the top level property `authenticationMethod: none`.

To obtain a CSRF token one must send a `GET` or `HEAD` request with a `x-csrf-token: fetch` header to the application router. The application router will return the created token in a `x-csrf-token: <token>` header, where `<token>` will be the value of the CSRF token.

If a CSRF protected route is requested with any of the above mentioned methods, `x-csrf-token: <token>` header should be present in the request with the previously obtained token. This request must use the same session as the fetch token request. If the `x-csrf-token` header is not present or is invalid, the application router will return status code “403 - Forbidden”.

## Troubleshooting

The XS advanced application router uses the `@sap/logging` package, which means that all of the typical logging features are available to control application logging. For example, to set all logging and tracing to the most detailed level, set the `<XS_APP_LOG_LEVEL>` environment variable to “debug”. For the on-premise XS advanced model run time, you can use the following command to set the logging level to “debug” without restarting the target application:

### Note

This could lead to a very large amount of data being written to the application logs and trace files. The asterisk wild card (\*) enables options that trace sensitive data that is then written to the logs and traces.

### Sample Code

```
xs setLoggingLevel <application-name> '*' debug
```

### Tip

Logging levels are application-specific and case-sensitive; they can be defined with lower-case characters (for example, “debug”) or upper-case characters (for example, “DEBUG”). An error occurs if you set a logging level incorrectly, for example, using lower-case characters “debug” where the application defines the logging level as “DEBUG”.

You can enable additional traces of the incoming and outgoing requests by setting the environment variable `<REQUEST_TRACE>` to true. When enabled basic information will be logged for every incoming and outgoing request of the application router.

The `@sap/logging` package sets the header `x-request-id` in the application router's responses. This is useful if you want to search the application router's logs and traces for entries that belong to a particular request execution. Note that the application router does not change the headers received from the back end and forwarded to the client. If the back end is a Node.js application which uses the `@sap/logging` package (and also sets the `x-request-id` header), then the value of the header that the client receives is the one coming from the back end and not the one from the application router itself.

## Related Information

[XS Advanced Application Router Configuration Syntax \[page 725\]](#)

[XS Advanced Application Router Environment Variables \[page 741\]](#)

[Configure the XS Advanced Application Router \[page 717\]](#)

[XS Advanced Application-Router Resource Files \[page 720\]](#)

## 10.1.3 XS Advanced Application Router Configuration Syntax

The application description defined in the `xs-app.json` file contains the configuration information used by the application router.

The following example of an `xs-app.json` application descriptor shows the JSON-compliant syntax required and the properties that either must be set or can be specified as an additional option.

### Code Syntax

```
{  
    "welcomeFile": "index.html",  
    "authenticationMethod": "route",  
    "sessionTimeout": 10,  
    "pluginMetadataEndpoint": "/metadata",  
    "routes": [  
        {  
            "source": "^/sap/ui5/1(.*)$",  
            "target": "$1",  
            "destination": "ui5",  
            "csrfProtection": false  
        },  
        {  
            "source": "/employeeData/(.*)",  
            "target": "/services/employeeService/$1",  
            "destination": "employeeServices",  
            "authenticationType": "xsuaa",  
            "scope": ["$XSAPPNAME.viewer", "$XSAPPNAME.writer"],  
            "csrfProtection": true  
        },  
        {  
            "source": "^(.*)$",  
            "target": "/web/$1",  
            "localDir": "static-content",  
            "replace": {  
                "pathSuffixes": ["/abc/index.html"],  
                "vars": ["NAME"]  
            }  
        }  
    ],  
    "login": {  
        "callbackEndpoint": "/custom/login/callback"  
    },  
    "logout": {  
        "logoutEndpoint": "/my/logout",  
        "logoutPage": "/logout-page.html"  
    },  
    "destinations": {  
        "employeeServices": {  
            "logoutPath": "/services/employeeService/logout",  
            "logoutMethod": "GET"  
        }  
    },  
    "compression": {  
        "minSize": 2048  
    },  
    "whitelistService": {  
        "endpoint": "/whitelist/service"  
    },  
    "websockets": {  
        "enabled": true  
    },  
    "errorPage": [  
        {"status": [400, 401, 402], "file": "/custom-err-4xx.html"},  
        {"status": 501, "file": "/custom-err-501.html"}  
    ]  
}
```

```
    ]  
}
```

## welcomeFile

The Web page served by default if the HTTP request does not include a specific path, for example, index.html.

Table 109: XS Advanced Application-Router Parameters

Property	Type	Mandatory	Values
welcomeFile	String	No	HTML page, for example, index.html

### Code Syntax

```
"welcomeFile": "index.html",
```

## authenticationMethod

The method used to authenticate user requests, for example: “route” or “none” (no authentication).

### Code Syntax

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

Table 110:

Property	Type	Mandatory	Values
authenticationMethod	String	No	<ul style="list-style-type: none"><li>• route Authentication type is defined in the routes configuration</li><li>• none Disables authentication for all routes</li></ul>

### Caution

If authenticationMethod is set to “none”, logon with User Account and Authentication (UAA) is disabled.

## routes

Defines all route objects, for example: source, target, and, destination.

Table 111: Application Router: Routes Properties

Property	Type	Mandatory	Description
source	RegEx	Yes	<p>Regular expression that matches an incoming request path. To ensure the RegEx matches the complete path, wrap it with ^ and \$, for example, "^/sap/ui5/1(.*)\$".</p> <div style="background-color: #ffffcc; padding: 10px;"><p><b>i Note</b></p><ul style="list-style-type: none"><li>• A request matches a particular route if its path contains the specified regular expression.</li><li>• Use "matchCase" to specify if the regular expression defined in path is matched in a case-sensitive (true) way or not (false). Default is true.</li></ul></div>
httpMethods	Array of upper-case HTTP methods	No	<p>HTTP methods that will be served by this route; the supported methods are: DELETE, GET, HEAD, OPTIONS, POST, PUT, TRACE, and PATCH.</p> <div style="background-color: #ffffcc; padding: 10px;"><p><b>➔ Tip</b></p><p>If this option is not specified, the route will serve any HTTP method.</p></div>
target	String	No	The incoming request path is rewritten to this target
destination	String	No	The destination to which the incoming request should be forwarded. This must match one of the name properties defined in the destinations environment variable..
scope	String	No	The authorization scope required to access the target path. The scope itself is defined in the application's security descriptor ( <code>xs-security.json</code> ), for example, " <code>\$\$XSAPPNAME.Display</code> " or " <code>\$\$XSAPPNAME.Create</code> ".
localDir	String	No	The directory from which application router serves static content (for example, from the application's web/ module)

Property	Type	Mandatory	Description
replace	Object	No	An object that contains the configuration for replacing placeholders with values from the environment. It is only relevant for static resources.
csrfProtection	Boolean	No	Toggle whether this route needs CSRF token protection. The default value is "true". The application router enforces CSRF protection for any HTTP request that changes state on the server side, for example: PUT, POST, or DELETE.
authenticationType	String	No	The value can be "xsuaa", "basic" or "none". The default value is "xsuaa". When "xsuaa" is used the specified UAA server will handle the authentication (the user is redirected to the UAA's login form). The basic mechanism works with SAP HANA users. If "none" is used then no authentication is needed for this route.
cacheControl	String	No	A string representing the value of the Cache-Control header, which is set on the response when serving static resources. By default the Cache-Control header is not set.

**i Note**

This value is relevant only for static resources.

## Code Syntax

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

**i Note**

The properties `target`, `destination`, and `localDir` are optional. However, at least one of them **must** be defined.

The properties `destination` and `localDir` cannot be used together in the same route.

If there is no route defined for serving static content via `localDir`, a default route is created for “`resources`” directory as follows:

#### Sample Code

```
{  
  "routes": [  
    {  
      "source": " ^/(.*)$ ",  
      "localDir": "resources"  
    }  
  ]  
}
```

#### Note

If there is at least one route using `localDir`, the default route is not added.

The `httpMethods` option allows you to split the same path across different targets depending on the HTTP method. For example:

#### Sample Code

```
"routes": [  
  {  
    "source": " ^/app1/(.*)$ ",  
    "target": "/before/$1/after",  
    "httpMethods": ["GET", "POST"]  
  }  
]
```

This route will be able to serve only GET and POST requests. Any other method (including extension ones) will get a 405 Method Not Allowed response. The same endpoint can be split across multiple destinations depending on the HTTP method of the requests:

#### Sample Code

```
"routes": [  
  {  
    "source": " ^/app1/(.*)$ ",  
    "destination": "dest-1",  
    "httpMethods": ["GET"]  
  },  
  {  
    "source": " ^/app1/(.*)$ ",  
    "destination": "dest-2",  
    "httpMethods": ["DELETE", "POST", "PUT"]  
  }  
]
```

The sample code above will route GET requests to the target dest-1, DELETE, POST and PUT to dest-2, and any other method receives a 405 Method Not Allowed response. It is also possible to specify catchAll routes, namely those that do not specify httpMethods restrictions:

### Sample Code

```
"routes": [
{
  "source": "^/app1/(.*)$",
  "destination" : "dest-1",
  "httpMethods": ["GET"]
},
{
  "source": "^/app1/(.*)$",
  "destination" : "dest-2"
}
]
```

In the sample code above, GET requests will be routed to dest-1, and all the rest to dest-2.

## replace

The replace object configures the placeholder replacement in static text resources.

### Sample Code

```
{
  "replace": {
    "pathSuffixes": ["index.html"],
    "vars": ["escaped_text", "NOT_ESCAPED"]
  }
}
```

The replace keyword requires the following properties:

Table 112: Replacement Properties for Static Resource URLs

Property	Type	Description
pathSuffixes	Array	An array defining the path suffixes that are relative to localDir. Only files with a path ending with any of these suffixes will be processed.
vars	Array	A white list with the environment variables that will be replaced in the files matching the suffix specified in pathSuffixes.

The supported tags for replacing environment variables are: {{ENV\_VAR}} and {{{ENV\_VAR}}} . If there such an environment variable is defined, it will be replaced, otherwise it will be just an empty string.

### Note

Any variable that is replaced using two-brackets syntax {{ENV\_VAR}} will be HTML-escaped; the triple brackets syntax {{{ENV\_VAR}}} is used when the replaced values do not need to be escaped and all values will remain unchanged. For example, if the value of the environment variable is ab"cd the result will be ab&quot;cd.

If your application descriptor `xs-app.json` contains a route like the one illustrated in the following example,

```
{  
  "source": "^/get/home(.*)",  
  "target": "$1",  
  "localDir": "resources",  
  "replace": {  
    "pathSuffixes": ["index.html"],  
    "vars": ["escaped_text", "NOT_ESCAPED"]  
  }  
}
```

And your application uses the following `index.html` start file:

### Sample Code

```
<html>  
  <head>  
    <title>{{escaped_text}}</title>  
    <script src="{{NOT_ESCAPED}}/index.js"/>  
  </head>  
</html>
```

Then, in `index.html`, `{{escaped_text}}` and `{{{NOT_ESCAPED}}}` will be replaced with the value defined in the environment variables `<escaped_text>` and `<NOT_ESCAPED>`.

### Note

All `index.html` files are processed; if you want to apply the replacement only to specific files, you must set the path relative to `localDir`. In addition, all files should comply with the UTF-8 encoding rules.

The content type returned by a request is based on the file extension specified in the route. The application router support the following file types:

- `.json` (`application/json`)
- `.txt` (`text/plain`)
- `.html` (`text/html`) **default**
- `.js` (`application/javascript`)
- `.css` (`text/css`)

The following table illustrates some examples of the `pathSuffixes` properties:

Table 113: Examples of the `pathSuffixes` Property

Example	Result
<pre>{ "pathSuffixes":   [".html"] }</pre>	All files with the extension <code>.html</code> under <code>localDir</code> and its subfolders will be processed.
<pre>{ "pathSuffixes": ["/abc/ main.html", "some.html"] }</pre>	For the suffix <code>/abc/main.html</code> , all files named <code>main.html</code> which are inside a folder named <code>abc</code> will be processed.  For the suffix <code>some.html</code> , all files with a name that ends with " <code>some.html</code> " will be processed, for example: <code>some.html</code> , <code>awesome.html</code> .

Example	Result
{ "pathSuffixes": [ ".html" ] }	All files with the name "some.html" will be processed. For example: some.html, /abc/some.html.

## sessionTimeout

Define the amount of time (in minutes) for which a session can remain inactive before it closes automatically (times out); the default time out is 15 minutes.

### i Note

The `sessionTimeout` property is no longer available; to set the session time out value, use the environment variable `<SESSION_TIMEOUT>`.

### Sample Code

```
{
  "sessionTimeout": 40,
}
```

With the configuration in the example above, a session timeout will be triggered after 40 minutes and involves central log out.

## login

A redirect to the application router at a specific endpoint takes place during OAuth2 authentication with the User Account and Authentication service (UAA). This endpoint can be configured in order to avoid possible collisions, as illustrated in the following example:

### Sample Code

Application Router "login" Property

```
"login": {
  "callbackEndpoint": "/custom/login/callback"
}
```

### Tip

The default endpoint is "/login/callback".

## logout

Define any options that apply if you want your application to have central log out end point. In this object you can define an application's central log out end point by using the `logoutEndpoint` property, as illustrated in the following example:

### Sample Code

```
"logout": {  
    "logoutEndpoint": "/my/logout"  
}
```

Making a GET request to “/my/logout” triggers a client-initiated central log out. Central log out can be initiated by a client or triggered due to a session timeout, with the following consequences:

- Client-initiated central log out
  - Deletes the user session
  - Requests the log out paths for all your back-end services (if you provided these paths in the `destinations` property).
  - Redirects to the User Account and Authentication service (UAA), if such a service is provided, and logs out from there.

### Tip

It is not possible to redirect back to your application after logging out from UAA.

- Session time out
  - Deletes the user session
  - Requests the log out paths for all your back-end services (if you provided these paths in the `destinations` property).

You can use the `logoutPage` property to specify the Web page in one of the following ways:

- URL path  
The UAA service redirects the user back to the application router, and the path is interpreted according to the configured routes.

### Note

The resource that matches the URL path specified in the property `logoutPage` should not require authentication; for this route, the property `authenticationType` must be set to “none”.

In the following example, `my-static-resources` is a folder in the working directory of the application router; the folder contains the file `logout-page.html` along with other static resources.

### Sample Code

```
{  
    "authenticationMethod": "route",  
    "logout": {  
        "logoutEndpoint": "/my/logout",  
        "logoutPage": "/logout-page.html"  
    },
```

```

    "routes": [
      {
        "source": "^/logout-page.html$",
        "localDir": "my-static-resources",
        "authenticationType": "none"
      }
    ]
  }
}

```

- Absolute HTTP(S) path

The UAA will redirect the user to a page (or application) different from the application router.

#### Sample Code

```

"logout": {
  "logoutEndpoint": "/my/logout",
  "logoutPage": "http://acme.com/employees.portal"
}

```

## destinations

Specify any additional options for your destinations. The destinations section in `xs-app.json` extends the destination configuration in the deployment manifest (`manifest.yml`), for example, with some static properties such as a logout path.

#### Sample Code

```

{
  "destinations": {
    "node-backend": {
      "logoutPath": "/ui5logout",
      "logoutMethod": "GET"
    }
  }
}

```

The following syntax rules apply:

Table 114: Application Router: Destination Properties

Property	Type	Mandatory	Description
logoutPath	String	No	The log out end point for your destination. The <code>logoutPath</code> will be called when central log out is triggered or a session is deleted due to a time out. The request to <code>logoutPath</code> contains additional headers, including the JWT token.

Property	Type	Mandatory	Description
logoutMethod	String	No	The logoutMethod property specifies the HTTP method with which the logoutPath will be requested, for example, POST, PUT, GET; the default value is POST

## compression

The `compression` keyword enables you to define if the application router compresses text resources before sending them. By default, resources larger than 1KB are compressed. If you need to change the compression size threshold, for example, to “2048 bytes”, you can add the optional property “`minSize": <size_in_KB>`”, as illustrated in the following example.

### Sample Code

```
{
  "compression": {
    "minSize": 2048
  }
}
```

You can disable compression in the following ways:

- Global  
Within the compression section add “`enabled": false`”
- Front end  
The client sends a header “`Accept-Encoding`” which does not include “`gzip`”.
- Back end  
The application sends a header “`Cache-Control`” with the “`no-transform`” directive.

Table 115: Application Router: Compression Properties

Property	Type	Mandatory	Description
<code>minSize</code>	Number	No	Text resources larger than this size will be compressed.
<code>enabled</code>	Boolean	No	Globally disables or enables compression. The default value is true.

### Note

If the `<COMPRESSION>` environment variable is set it will overwrite any existing values.

## pluginMetadataEndpoint

Adds an endpoint that serves a JSON string representing all configured plugins.

### Sample Code

```
{  
  "pluginMetadataEndpoint": "/metadata"  
}
```

### Note

If you request the relative path /metadata of your application, a JSON string is returned with the configured plug-ins.

## whitelistService

Enable the white-list service to help prevent against click-jacking attacks. Enabling the white-list service opens an endpoint accepting GET requests at the relative path configured in the endpoint property, as illustrated in the following example:

### Sample Code

```
{  
  "whitelistService": {  
    "endpoint": "/whitelist/service"  
  }  
}
```

If the white-list service is enabled in the application router, each time an HTML page needs to be rendered in a frame, the white-list service is used check if the parent frame is allowed to render the content in a frame.

### Host Names and Domain Names

The white-list service reads a list of allowed host names and domains defined in the environment variable <CJ\_PROTECT\_WHITELIST>; the content is a JSON list of objects with the following properties:

Table 116: White List of Host and Domain Names

Property	Type	Mandatory	Description
protocol	String	No	URI scheme, for example “HTTP”.
host	String	Yes	A valid host name, for example, acme . com . hostname, or a domain name defined with an asterisk (*) *. acme . com.
port	String/Number	No	Port string or number containing a valid port.

The following snippet shows an example of the resulting JSON array:

#### Sample Code

```
[  
  {  
    "protocol": "http",  
    "host": "*.acme.com",  
    "port": 12345  
  },  
  {  
    "host": "hostname.acme.com"  
  }  
]
```

#### Note

Matching is done against the properties provided. For example, if only host name is provided, the white-list service returns “framing: true” for all, and matching will be for all schemata and protocols.

### Return Value

The white-list service accepts only GET requests, and returns a JSON object as the response. The white-list service call uses the parent origin as URI parameter (URL encoded) as follows:

#### Sample Code

```
GET url/to/whitelist/service?parentOrigin=https://parent.domain.acme.com
```

The response is a JSON object with the following properties; property “active” has the value false only if <CJ\_PROTECT\_WHITELIST> is not provided:

#### Sample Code

```
{  
  "version" : "1.0",  
  "active" : true | false,  
  "origin" : "<same as passed to service>",  
  "framing" : true | false  
}
```

The “active” property enables framing control; the “framing” property specifies if framing should be allowed. By default, the Application Router (approuter.js) sends the X-Frame-Options header with value the SAMEORIGIN.

#### Tip

If the white-list service is enabled, the header value probably needs to be changed, see the X-Frame-Options header section for details about how to change it.

## websockets

The application router can forward web-socket communication. Web-socket communication must be enabled in the application router configuration, as illustrated in the following example. If the back-end service requires authentication, the upgrade request should contain a valid session cookie. The application router supports the destination schemata "ws", "wss", "http", and "https".

### Sample Code

```
{  
  "websockets": {  
    "enabled": true  
  }  
}
```

### ⚠ Restriction

A web-socket ping is not forwarded to the back-end service.

## errorPage

Errors originating in the application router show the HTTP status code of the error. It is possible to display a custom error page using the `errorPage` property.

The property is an array of objects, each object can have the following properties:

Table 117: Application Router: `errorPage` Properties

Property	Type	Mandatory	Description
<code>status</code>	Number/Array	Yes	HTTP status code.
<code>file</code>	String	Yes	File path relative to the working directory of the application router.

In the following code example, errors with status code "400", "401" and "402" will show the content of `./custom-err-4xx.html`; errors with the status code "501" will display the content of `./http_resources/custom-err-501.html`.

### Sample Code

```
{ "errorPage" : [  
  {"status": [400,401,402], "file": "./custom-err-40x.html"},  
  {"status": 501, "file": "./http_resources/custom-err-501.html"}  
]
```

### **i** Note

The contents of the `errorPage` configuration section have no effect on errors that are not generated by the application router.

## Related Information

[XS Advanced Application Router Environment Variables \[page 741\]](#)

[Application Security Descriptor Configuration Syntax \[page 767\]](#)

[The XS Advanced Application Descriptor \[page 721\]](#)

[Configure the XS Advanced Application Router \[page 717\]](#)

## 10.1.4 XS Advanced Application Routes and Destinations

The application router is the single point of entry for an application.

The application router is used to serve static content, propagates user information, and acts as a proxy to forward requests to other micro services. The routing configuration for an application is defined in one or more destinations. The application router configuration is responsible for the following tasks:

- Main user-entry point to application service
- Serves static content
- Serves routes and destinations
- Rewrite URLs
- Forwards requests to other services
- Propagates user information
- Handles authentication
- Manages HTML5 application/browser sessions

### **i** Note

The application router does not manage server caching. The server cache must be set (with e-tags) in the server itself. The cache should contain not only static content but container resources, too, for example, relating to OData metadata.

In Cloud-based scenarios, a request is routed by means of domain names and a central router component. This requires a DNS wild card entry so that all requests are redirected to the "Cloud Router" which does dispatching and load balancing. For SAP on-Premise, it is also possible (as a fall back) to provide a routing service based on ports. In this scenario, the router opens a dedicated in-bound port for each application name, and each incoming request to an in-bound port is mapped to a range of outgoing addresses. The central router uses the mapping for load-balancing and forwards the request to the appropriate target.

## Destinations

A destination defines the back end connectivity. In its simplest form, a destination is a URL to which requests are forwarded. Destinations are either defined in the application deployment descriptor (`manifest.yml`).

### Note

The “name” and “url” properties are mandatory.

The following example shows destinations defined in the `destinations` environment variable.

### Sample Code

```
[  
  {  
    "name": "backend",  
    "url": "http://localhost:3000",  
    "forwardAuthToken": true  
  }  
]
```

### Note

The value of the destination “name” property (“name”: “backend” in the example below) must match the destination property of a route in the corresponding application-router configuration file (`xs-app.json`). The “name” and “url” properties are mandatory.

Destinations are defined in an environment variable for the approuter application. The destinations are typically specified in the application manifest (`manifest.yml`) file. The router information is added to the `env: destinations` section of the `manifest.yml`, as illustrated in the following example.

### Sample Code

```
---  
applications:  
- name: node-hello-world  
  port: <approuter-port> #the port used for the approuter  
  memory: 100M  
  path: web  
  env:  
    destinations: >  
      [  
        {  
          "name": "backend",  
          "url": "http://<hostname>:<node-port>",  
          "forwardAuthToken": true  
        }  
      ]  
  services:  
    - node-uaa
```

## Related Information

[Configure the XS Advanced Application Router \[page 717\]](#)  
[XS Advanced Application-Router Resource Files \[page 720\]](#)  
[XS Advanced Application Router Configuration Syntax \[page 725\]](#)  
[Configure the XS Advanced Application Router \[page 717\]](#)

### 10.1.5 XS Advanced Application Router Environment Variables

A list of environment variables that can be used to configure the application router.

The following table lists the environment variables that you can use to configure the application router. The table also provides a short description of each variable and, where appropriate, an example of the configuration data.

Table 118: Application Router Configuration Variables

Variable	Description
<code>httpHeaders</code>	Configure the application router to return additional HTTP headers in its responses to client requests
<code>SESSION_TIMEOUT</code>	Set the time to trigger an automatic central log out from the User Account and Authentication (UAA) server.
<code>SEND_XFRAMEOPTIONS</code>	Set or change the X-Frame-Options header
<code>CJ_PROTECT_WHITELIST</code>	A list of allowed server or domain origins to use when checking for click-jacking attacks
<code>WS_ALLOWED_ORIGINS</code>	A list of the allowed server (or domain) origins that the application router uses to verify requests
<code>JWT_REFRESH</code>	Configures the automatic refresh of the JSON Web Token (JWT) provided by the User Account and Authentication (UAA) service to prevent expiry (default is 5 minutes).
<code>UAA_SERVICE_NAME</code>	Specify the <b>exact</b> name of the UAA service to bind to an application.
<code>INCOMING_CONNECTION_TIMEOUT</code>	Specify the maximum time (in milliseconds) for a client connection. If the specified time is exceeded, the connection is closed.
<code>TENANT_HOST_PATTERN</code>	Define a regular expression to use when resolving tenant host names in the request's host name.
<code>COMPRESSION</code>	Configure the compression of resources before a response to the client.

Variable	Description
<code>SECURE_SESSION_COOKIE</code>	Configure the enforcement of the Secure flag of the application router's session cookie.
<code>REQUEST_TRACE</code>	Enable additional traces of the incoming and outgoing requests
<code>CORS</code>	Provide support for cross-origin requests, for example, by allowing the modification of the request header.

## httpHeaders

If configured, the application router sends additional HTTP headers in its responses to a client request. You can set the additional HTTP headers in the `<httpHeaders>` environment variable. The following example configuration shows how to configure the application router to send two additional headers in the responses to the client requests from the application `<myApp>`:

```
xs set-env <myApp> httpHeaders "[ { \"X-Frame-Options\": \"ALLOW-FROM http://acme.com\" }, { \"Test-Additional-Header\": \"1\" } ]"
```

or

```
xs set-env <myApp> httpHeaders '[{ "X-Frame-Options": "ALLOW-FROM http://acme.com" }, { "Test-Additional-Header": "1" }]'
```

### Tip

To ensure better security of your application set the Content-Security-Policy header. This is a response header which informs browsers (capable of interpreting it) about the trusted sources from which an application expects to load resources. This mechanism allows the client to detect and block malicious scripts injected into the application. A value can be set via the `<httpHeaders>` environment variable in the additional headers configuration. The value represents a security policy which contains directive-value pairs. The value of a directive is a whitelist of trusted sources.

Usage of the Content-Security-Policy header is considered second line of defense. An application should always provide proper input validation and output encoding.

## SESSION\_TIMEOUT

You can configure the triggering of an automatic central log-out from the User Account and Authentication (UAA) service if an application session is inactive for a specified time. A session is considered to be inactive if no requests are sent to the application router. The following command shows how to set the environment variable `<SESSION_TIMEOUT>` to 40 (forty) minutes for the application `<myApp1>`:

```
xs set-env <myApp1> SESSION_TIMEOUT 40
```

### Note

You can also set the session timeout value in the application's `manifest.yml` file, as illustrated in the following example:

### Sample Code

```
- name: myApp1
  memory: 100M
  path: web
  env:
    SESSION_TIMEOUT: 40
```

### Tip

If the authentication type for a route is set to "xsuaa" (for example, "`authenticationType": "xsuaa"`), the application router depends on the UAA server for user authentication, and the UAA server might have its own session timeout defined. To avoid problems caused by unexpected timeouts, it is recommended that the session timeout values configured for the application router and the UAA are identical."

## SEND\_XFRAMEOPTIONS

By default, the application router sends the `X-Frame-Options` header with the value "SAMEORIGIN". You can change this behavior either by disabling the sending of the default header value (for example, by setting `SEND_XFRAMEOPTIONS` environment variable to false) or by overriding the value, for example, by configuring additional headers (with the `<httpHeaders>` environment variable).

The following example shows how to disable the sending of the `X-Frame-Options` for a specific application, `myApp1`:

```
xs set-env <myApp1> SEND_XFRAMEOPTIONS false
```

## CJ\_PROTECT\_WHITELIST

The `<CJ_PROTECT_WHITELIST>` specifies a list of origins (for example, host or domain names) that do not need to be protected against click-jacking attacks. This list of allowed host names and domains are used by the application router's white-list service to protect XS advanced applications against click-jacking attacks. When an HTML page needs to be rendered in a frame, a check is done by calling the white-list service to validate if the parent frame is allowed to render the requested content in a frame. The check itself is provided by the white-list service.

The following example shows how to add a host name to the click-jacking protection white list for the application, `myApp1`:

```
xs set-env <myApp1> CJ_PROTECT_WHITELIST {<protocol>, <hostname>, <portNr>}
```

The content is a JSON list of objects with the properties listed in the following table:

Table 119: White List of Host and Domain Names

Property	Type	Mandatory	Description
protocol	String	No	URI scheme, for example "HTTP".
host	String	Yes	A valid host name, for example, acme . com . hostname, or a domain name defined with an asterisk (*) * . acme . com.
port	String/Number	No	Port string or number containing a valid port.

#### i Note

Matching is done against the properties provided. For example, if only the host name is provided, the white-list service matches all schemata and protocols.

```
xs set-env <myApp1> CJ_PROTECT_WHITELIST {<*.acme.com>}
```

## WS\_ALLOWED\_ORIGINS

When the application router receives an upgrade request, it verifies that the `origin` header includes the URL of the application router. If this is not the case, then an HTTP response with status 403 is returned to the client. This origin verification can be further configured with the environment variable `<WS_ALLOWED_ORIGINS>`, which defines a list of the allowed origins the application router uses in the verification process.

#### i Note

The structure of the `<WS_ALLOWED_ORIGINS>` variable is the same as the variable `<CJ_PROTECT_WHITELIST>`.

```
xs set-env <myApp1> WS_ALLOWED_ORIGINS {<*.acme.com>}
```

## JWT\_REFRESH

The `JWT_REFRESH` environment variable is used to configure the application router to refresh a JSON Web Token (JWT) for an application, by default, 5 minutes before the JWT expires, if the session is active.

```
xs set-env <myApp1> JWT_REFRESH 1
```

If the JWT is close to expiration and the session is still active a JWT refresh will be triggered `<JWT_REFRESH>` minutes before expiration. The default value is 5 minutes. To disable the automatic refresh, set the value of `<JWT_REFRESH>` to 0 (zero).

## UAA\_SERVICE\_NAME

The `UAA_SERVICE_NAME` environment variable enables you to configure an instance of the User Account and Authorization service for a specific application, as illustrated in the following example:

```
xs set-env <myApp1> UAA_SERVICE_NAME <myUAAServiceName>
```

### i Note

The details of the service configuration is defined in the `<VCAP_SERVICES>` environment variable, which is not configured by the user.

## INCOMING\_CONNECTION\_TIMEOUT

The `INCOMING_CONNECTION_TIMEOUT` environment variable enables you to set the maximum time (in milliseconds) allowed for a client connection, as illustrated in the following example:

```
xs set-env <myApp1> INCOMING_CONNECTION_TIMEOUT 60,000
```

If the specified time is exceeded, the connection is closed. If `INCOMING_CONNECTION_TIMEOUT` is set to zero (0), the connection-timeout feature is disabled. The default value for `INCOMING_CONNECTION_TIMEOUT` is 120,000 ms (2 min).

## TENANT\_HOST\_PATTERN

The `TENANT_HOST_PATTERN` environment variable enables you to specify a string containing a regular expression with a capturing group. The requested host is matched against this regular expression. The value of the first capturing group is used as the tenant Id., as illustrated in the following example:

```
xs set-env <myApp1> TENANT_HOST_PATTERN
```

## COMPRESSION

The `COMPRESSION` environment variable enables you to configure the compression of resources before a response to the client, as illustrated in the following example:

```
xs set-env <myApp1> COMPRESSION
```

## SECURE\_SESSION\_COOKIE

The `SECURE_SESSION_COOKIE` can be set to `true` or `false`. By default, the `Secure` flag of the session cookie is set depending on the environment the application router runs in. For example, when the application router is

running behind a router that is configured to serve HTTPS traffic, then this flag will be present. During local development the flag is not set.

### Note

If the `Secure` flag is enforced, the application router will reject requests sent over unencrypted connections.

The following example illustrates how to set the `SECURE_SESSION_COOKIE` environment variable:

```
xs set-env <myApp1> SECURE_SESSION_COOKIE true
```

## REQUEST\_TRACE

You can enable additional traces of the incoming and outgoing requests by setting the environment variable `<REQUEST_TRACE>` to "true". If enabled, basic information is logged for every incoming and outgoing request to the application router.

The following example illustrates how to set the `REQUEST_TRACE` environment variable:

```
xs set-env <myApp1> REQUEST_TRACE true
```

### Tip

This is in addition to the information generated by the Node.js package `@sap/logging` that is used by the XS advanced application router.

## CORS

The `CORS` environment variable enables you to provide support for cross-origin requests, for example, by allowing the modification of the request header. Cross-origin resource sharing (CORS) permits Web pages from other domains to make HTTP requests to your application domain, where normally such requests would automatically be refused by the Web browser's security policy.

Cross-origin resource sharing (CORS) is a mechanism that allows restricted resources on a Web page to be requested from another domain (protocol and port) outside the domain (protocol and port) from which the first resource was served. The CORS configuration enables you to define details to control access to your application resource from other Web browsers. For example, you can specify where requests can originate from or what is allowed in the request and response headers. The following example illustrates a basic CORS configuration:

```
[  
  {  
    "uriPattern": "\\\\route1$"  
    "allowedMethods": [  
      "GET"  
    ],  
    "allowedOrigin": [  
      {  
        "origin": "http://www.  
        "methods": ["GET"]  
      }  
    ]  
  }  
]
```

```

        "host": "host.acme.com",
        "protocol": "https",
        "port": 345
    }
],
"maxAge": 3600,
"allowedHeaders": [
    "Authorization",
    "Content-Type"
],
"exposeHeaders": [
    "customHeader1",
    "customHeader2"
],
"allowedCredentials": true
}
]

```

The CORS configuration includes an array of objects with the following properties, some of which are mandatory:

Table 120: Available Settings for CORS Options

CORS Property	Type	Mandatory	Description
uriPattern	String	Yes	A regular expression (RegExp) representing the source routes to which the CORS configuration applies. To ensure that the RegExp matches the complete path, surround it with ^ and \$, for example, "uriPattern": "^\\route1\$". Defaults: none
allowOrigin	Array	Yes	A comma-separated list of objects each of which contains a host name, port and protocol that are allowed by the server, for example: [ { "host": "www.acme.com" } ] or [ { "host": ".acme.com" } ].
allowMethods	Array	No	<p>A comma-separated list of HTTP methods that are allowed by the server, for example, "GET", "POST". If <code>allowMethods</code> is defined but no method is specified, the default "GET", "POST", "HEAD", "OPTIONS" (all) applies.</p> <p><b>Tip</b> The specified methods must be upper-case, for example, GET. Matching of the method type is case-sensitive.</p>

CORS Property	Type	Mandatory	Description
allowHeaders	Array	No	A comma-separated list of request headers that are allowed by the server. the default values are as follows: [ "Origin", "Accept", "X-Requested-With", "Content-Type", "Access-Control-Request-Method", "Access-Control-Request-Headers" ].
maxAge	String	No	A single value specifying the length of time (in seconds) a preflight request should be cached for. A negative value the prevents CORS filter from adding this response header to the pre-flight response. If maxAge is defined but no value is specified, the default time of "1800" seconds applies.
exposeHeaders	Array	No	A comma-separated list of <b>response</b> headers that are allowed to be exposed. If exposeHeaders is defined but no response header is specified for exposure, no default value is supplied.
allowedCredentials	Boolean	No	A Boolean flag that indicates whether the specified resource supports user credentials. The default setting is "true".

It is also possible to include the CORS configuration in either the `manifest.yml` or the `manifest-op.yml` file. The code in the following example enables the CORS configuration for any route with a source URI pattern that matches the RegExp `"^/route1$"`:

### Sample Code

#### CORS Configuration in the `manifest.yml` File

```
- name: node-hello-world
  memory: 100M
  path: web
  env:
    CORS: >
      [
        {
          "allowedOrigin": [
            {
              "host": "my_host",
              "protocol": "https"
            }
          ],
          "uriPattern": "^/route1$"
        }
      ]

```

## Related Information

[Configure the XS Advanced Application Router \[page 717\]](#)

[XS Advanced Application Router Configuration Syntax \[page 725\]](#)

[The XS Advanced Application Descriptor \[page 721\]](#)

## 10.2 Extend the XS Advanced Application Router

Use middleware and custom extension scripts to extend the functionality of the XS advanced application router.

### Context

There are various ways you can extend the functionality provided with the XS advanced application router. You can configure XS advanced JavaScript applications to use the application router as a regular Node.js package or start the application router from a dedicated script. You can inject custom middleware into the application router using predefined slots, and ensure that proper control is maintained over any injected application-router middleware.

### Procedure

1. Start the application router with your own JavaScript.

Instead of starting the application router directly, you can configure your XS advanced application to use its own script to start the application router, as illustrated in the following example:

#### Sample Code

```
var approuter = require('approuter');
var ar = approuter();
ar.start();
```

2. Inject custom middleware into the application router.

- a. Inject middleware directly in the application-router startup script.

The application router uses the connect framework, which enables you to reuse all injected “connect” middleware within the application router, for example, directly in the application start script:

#### Sample Code

```
var approuter = require('approuter');
var ar = approuter();
ar.beforeRequestHandler.use('/my-ext', function myMiddleware(req, res, next) {
    res.end('Request handled by my extension!');
});
ar.start();
```

## → Tip

To facilitate troubleshooting, always provide a name for your custom middleware.

- b. Add some `use` calls to middleware functions.

The `path` argument is optional. You can also chain `use` calls, as illustrated in the following example:

### Sample Code

```
var approuter = require('approuter');
var morgan = require('morgan');
var ar = approuter();
ar.beforeRequestHandler
  .use(morgan('combined'))
  .use('/my-ext', function myMiddleware(req, res, next) {
    res.end('Request handled by my extension!');
  });
ar.start();
```

The application router also allows you to insert custom middleware into additional "slots": `first`, `beforeRequestHandler`, and `beforeErrorHandler`.

- c. Return control to the application-router middleware, if necessary.

If your middleware does not complete the request processing, call `next` to return control to the application router middleware, as illustrated in the following example:

### Sample Code

```
ar.beforeRequestHandler.use('/my-ext', function myMiddleware(req, res,
  next) {
  res.setHeader('x-my-ext', 'passed');
  next();
});
```

3. Configure application-specific extensions to the XS advanced application router.

You can use an extension file, for example, `my-ext.js` to insert middleware at the following points: `first`, `beforeRequestHandler`, and `beforeErrorHandler`.

### Sample Code

Example Approuter Extension Configuration (`my-ext.js`)

```
module.exports = {
  insertMiddleware: {
    first: [
      function logRequest(req, res, next) {
        console.log('Got request %s %s', req.method, req.url);
      }
    ],
    beforeRequestHandler: [
      {
        path: '/my-ext',
        handler: function myMiddleware(req, res, next) {
          res.end('Request handled by my extension!');
        }
      }
    ]
  }
};
```

```
    }
};
```

## Related Information

[Custom Middleware Injection \[page 753\]](#)

[XS Advanced Application Router Extensions \[page 751\]](#)

[XS Advanced Application Router Extension API \[page 754\]](#)

[Maintaining XS Advanced Application Routes and Destinations \[page 716\]](#)

### 10.2.1 XS Advanced Application Router Extensions

Configure application-specific extensions to the XS advanced application router.

An application-router extension is defined by an object with the following properties:

- `insertMiddleware` - describes the middleware provided by this extension
  - `first`, `beforeRequestHandler`, and `beforeErrorHandler`  
An array of middleware, where each one can be either of the following elements:
    - A middleware function (invoked on all requests)
    - An object with the following properties:
      - `path`  
Handle requests only for this path
      - `handler`  
The middleware function to invoke

#### Sample Code

Example Approuter Extension Configuration (`my-ext.js`)

```
module.exports = {
  insertMiddleware: {
    first: [
      function logRequest(req, res, next) {
        console.log('Got request %s %s', req.method, req.url);
      }
    ],
    beforeRequestHandler: [
      {
        path: '/my-ext',
        handler: function myMiddleware(req, res, next) {
          res.end('Request handled by my extension!');
        }
      }
    ]
  };
};
```

The extension configuration can be referenced in the corresponding application's start script, as illustrated in the following example:

#### Sample Code

```
var approuter = require('approuter');
var ar = approuter();
ar.start({
  extensions: [
    require('./my-ext.js')
  ]
});
```

## Customize the Command Line

By default the application router handles its command-line parameters, but that can be customized as well.

An `<approuter>` instance provides the property `cmdParser` that is a commander instance. It is configured with the standard application router command line options. You can add custom options like this:

#### Sample Code

```
var approuter = require('approuter');
var ar = approuter();
var params = ar.cmdParser
  // add here custom command line options if needed
  .option('-d, --dummy', 'A dummy option')
  .parse(process.argv);
console.log('Dummy option:', params.dummy);
```

To disable the handling of command-line options in the application router, reset the `ar.cmdParser` property to "false", as illustrated in the following example:

```
ar.cmdParser = false;
```

## Related Information

[XS Advanced Application Router Extension API \[page 754\]](#)

[Extend the XS Advanced Application Router \[page 749\]](#)

[Custom Middleware Injection \[page 753\]](#)

[XS Advanced Application Router Extension API \[page 754\]](#)

## 10.2.2 Custom Middleware Injection

Extend the application router by injecting so-called middleware plug-ins and using the “Connect” framework.

The application router uses the connect framework, for more information, see [Connect framework](#) in the *Related Links* below. You can reuse all injected “connect” middleware within the application router, for example, directly in the application start script:

### Sample Code

```
var approuter = require('approuter');
var ar = approuter();
ar.beforeRequestHandler.use('/my-ext', function myMiddleware(req, res, next) {
  res.end('Request handled by my extension!');
});
ar.start();
```

### Tip

To facilitate troubleshooting, always provide a name for your custom middleware.

The path argument is optional. You can also chain `use` calls.

### Sample Code

```
var approuter = require('approuter');
var morgan = require('morgan');
var ar = approuter();
ar.beforeRequestHandler
  .use(morgan('combined'))
  .use('/my-ext', function myMiddleware(req, res, next) {
    res.end('Request handled by my extension!');
});
ar.start();
```

The application router defines the following slots where you can insert custom middleware:

- `first` - right after the connect application is created, and before any application router middleware. At this point security checks are not performed yet.

### Tip

This is good place for infrastructure logic, for example, logging and monitoring.

- `beforeRequestHandler` - before standard application router request handling, that is static resource serving or forwarding to destinations.

### Tip

This is a good place to handle custom REST API requests.

- `beforeErrorHandler` - before standard application router error handling.

➔ Tip

This is a good place to capture or customize error handling.

If your middleware does not complete the request processing, call `next` to return control to the application router middleware, as illustrated in the following example:

 Sample Code

```
ar.beforeRequestHandler.use('/my-ext', function myMiddleware(req, res, next) {
  res.setHeader('x-my-ext', 'passed');
  next();
});
```

## Related Information

[Middleware Connect Framework](#) ↗

[Extend the XS Advanced Application Router \[page 749\]](#)

[XS Advanced Application Router Extensions \[page 751\]](#)

[XS Advanced Application Router Extension API \[page 754\]](#)

## 10.2.3 XS Advanced Application Router Extension API

A detailed list of the features and functions provided by the application router extension API.

The application router extension API enables you to create new instances of the application router, manage the approuter instance, and insert middleware using the Node.js “connect” framework. This section contains detailed information about the following areas:

- Approuter Extension API Reference
- Middleware Slot

## Approuter Extension API Reference

The application router uses the “Connect” framework for the insertion of middleware components. You can reuse all connect middleware within the application router directly.

Table 121: Approuter Extension API Functions

API Function	Description
<code>approuter()</code>	Creates a new instance of the application router

API Function	Description
first	<p>Defines a "middleware slot" (a slot for the insertion of middleware) at the top of the application router middleware stack.</p> <p><b>Tip</b> This is a good place to insert infrastructure logic, for example, logging or monitoring.</p>
beforeRequestHandler	<p>Defines a "middleware slot" before the standard application route definitions.</p> <p><b>Tip</b> This is a good place to handle custom REST API requests.</p>
beforeErrorHandler	<p>Defines a "middleware slot" before the standard application route error handlers.</p> <p><b>Tip</b> This is a good place to capture or customize error handling.</p>
start(options, callback)	<p>Starts the application router with the given options.</p> <ul style="list-style-type: none"> <li>options - this argument is optional. If provided, it should be an object containing the following properties: <ul style="list-style-type: none"> <li>port - a TCP port the application router will listen to (server port).</li> <li>workingDir - the working directory for the application.</li> <li>extensions - an array of extensions, each one is an object with a name and a version.</li> <li>xsappContext - An object representing the content of xs-app.json. It takes precedence over the content of xs-app.json.</li> </ul> </li> <li>callback - optional function with signature callback(error). It is called when the application router has started successfully or an error occurred. If not provided and an error occurs (for example if the port is already in use), the process will terminate.</li> </ul>
close(callback)	<p>Stops the application router.</p> <ul style="list-style-type: none"> <li>callback - optional function with signature callback(error). It is called when the application router has stopped successfully or an error occurred.</li> </ul>

## Middleware Slot

Manage the insertion of middleware slots with the application router.

Table 122: Middleware Slot Management

Function	Description
<code>use(path, handler)</code>	<p>Inserts a request handling middleware in the current slot.</p> <ul style="list-style-type: none"><li>• <code>path</code> Handle only requests starting with this path (string, optional)</li><li>• <code>handler</code> A middleware function to invoke (function, mandatory)</li></ul> <p>Returns <code>this</code> for chaining.</p>

## Related Information

[Maintaining XS Advanced Application Routes and Destinations \[page 716\]](#)

[XS Advanced Application Router Extensions \[page 751\]](#)

[Middleware Connect Framework](#) ↗

# 11 Setting Up Security Artifacts

Developers create authorization information for business users in their environment; the information is deployed in an application and made available to administrators who complete the authorization setup and assign the authorizations to business users.

Developers store authorization information as design-time role templates in the security descriptor file `xs-security.json`. Using the `xsuaa` service broker, they deploy the security information in a dedicated XS advanced application. The XS advanced administrators view the authorization information in role templates, which they use as part of the run-time configuration. The administrators use the role templates to build roles, which are aggregated in role collections. The role collections are assigned, in turn, to business users.

The tasks required to set up authorization artifacts in SAP HANA XS advanced are performed by two distinct user roles: the application developer and the SAP HANA administrator. After the deployment of the authorization artifacts as role templates, the administrator of the SAP HANA XS advanced application uses the role templates provided by the developers for building role collections and assigning them to business users in the *SAP HANA XS Administration Tools* section of the *SAP HANA Administration Guide*.

## i Note

To test authorization artifacts after deployment, developers can use the role templates to build role collections and assign authorization to business users in the *SAP HANA XS Administration Tools*.

Table 123: Setting Up Authorization Artifacts

Step	Task	User Role	Tool
1	Specify the security descriptor file containing the functional authorization scopes for your application	Application developer	Text editor
2	Create role templates for the XS advanced application using the security descriptor file	Application developer	Text editor
3	Create a service instance from the <code>xsuaa</code> service in XS advanced using the service broker	Application developer	XS advanced CLI tool
4	Bind the service instance to the XS advanced application by including it into the manifest file	Application developer	Text editor
5	Deploy the XS advanced application	Application developer	XS advanced CLI tool
6	If required, create a new role in the XS advanced application role builder using role templates	XS advanced administrator	Application role builder
7	Create a role collection and assign roles to it	XS advanced administrator	Application role builder
8	Assign the role collection to a SAML 2.0 identity provider or to SAP HANA database users	XS advanced administrator	Application role builder, and SAML IDP Tool
9	Assign the users to roles using the role collections	XS advanced administrator	User interface of XS Advanced

## Related Information

[Assign Functional Authorization Scopes for an XS Advanced Application \[page 777\]](#)

[Create the Security Descriptor for Your XS Advanced Application \[page 763\]](#)

[Create an Instance of the XS UAA Service \[page 773\]](#)

[Bind the XS UAA Service Instance to the XS Advanced Application \[page 776\]](#)

## 11.1 Maintaining Application Security in XS Advanced

Set up the components required in the context of application security.

Application security is maintained in the `xs-security.json` file; the contents of the `xs-security.json` file cover the following areas:

- Functional authorizations
  - Scopes for functional authorization checks (see the related link)
- Attributes
  - A list of as-yet undefined information or sources (for example, the name of a cost center, or details of a role)
- Role-templates
  - A description of one or more roles (for example, “employee” or “manager”) to apply to a user and any attributes that apply to the roles; the role template is used to build a role

### Example

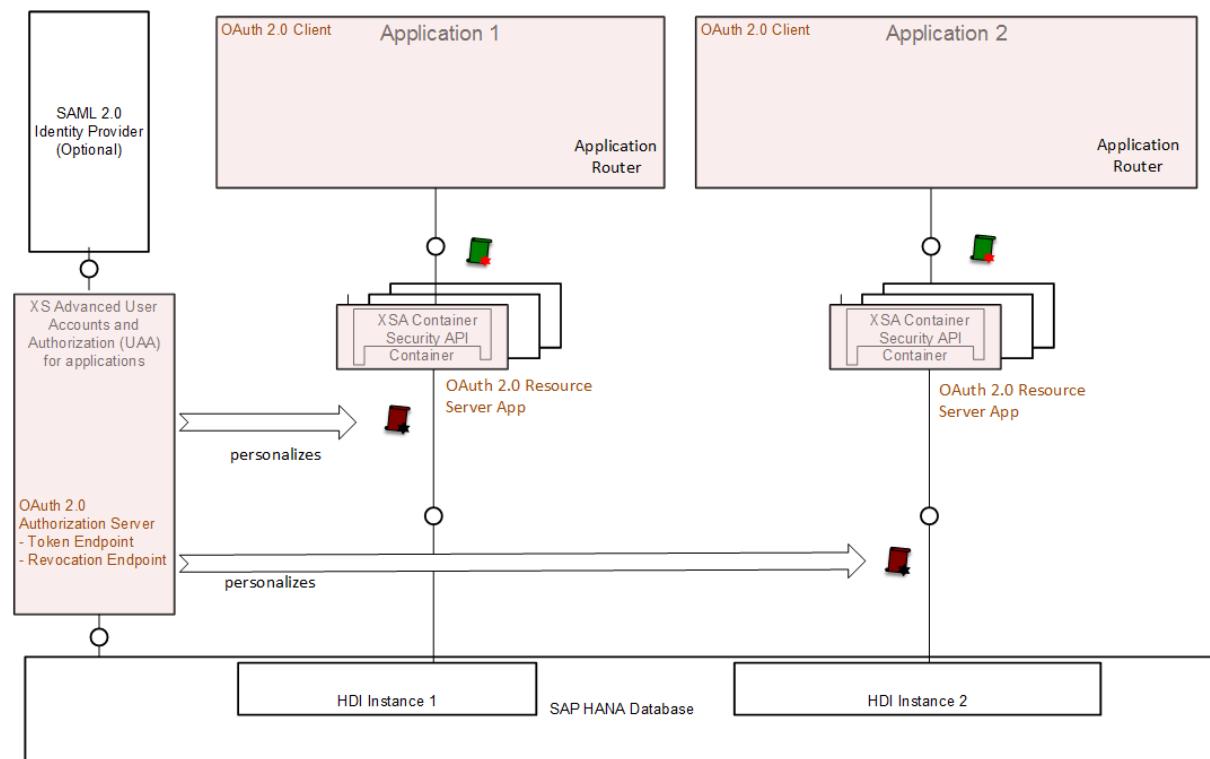
#### Sample Code

```
AppName
|- db/
|   |- package.json
|   |- src/
|       |- .hdiconfig
|       |- .hdinamespace
|       |- myEntity.hdbcards
|       |- myDataType.hdbcards
|           \- myDoc.hdbcards
|- web/
|   |- xs-app.json
|   \- resources/
|- js/
|   |- start.js
|   |- package.json
|   \- src/
\- xs-security.json      # Security deployment artifacts/scopes/auths
|- manifest.yml
\- mtad.yaml
```

## Security Architecture in XS Advanced

XS Advanced applications contain content (for example, Web content, micro services), which is deployed to different containers. Static Web content is deployed into the application router, application logic into node.js and Java containers, database logic into the SAP HANA database. Access to the content requires user authentication and the appropriate authorization.

XS advanced applications authenticate using OAuth 2.0. They use the User Account and Authorization (UAA) service of XS advanced as OAuth 2.0 authorization server, the application router as OAuth 2.0 client, and application logic running in node.js and Java backend services as OAuth 2.0 resource server.



The authentication process is triggered by the application router component, which is configured in the design-time artifact `xs-app.json`, if required. Authorization restricts the access to resources based on defined user permissions. Resources in the context of XS advanced applications are services provided by a container (for example, an OData Web service) or SAP HANA database artifacts such as tables in an HDI container.

## Related Information

[Authorization in SAP HANA XS Advanced \[page 778\]](#)

## 11.1.1 OAuth 2.0 in XS Advanced

Since SAP HANA XS advanced uses OAuth 2.0 for authentication, the architecture needs to provide the OAuth 2.0 elements OAuth 2.0 client, OAuth 2.0 authorization server, and OAuth 2.0 resource server.

You want to integrate an application in the environment of XS advanced model. This means that the XS advanced platform needs to know your application. Using the User Account and Authentication (UAA) service of XS advanced, you initially integrate your application into the XS advanced platform environment. The application needs to authenticate against the User Account and Authorization service. The authentication concept XS advanced uses is OAuth 2.0.

In this context, the UAA acts as OAuth 2.0 authorization server. The application router itself is the OAuth 2.0 client. To integrate the application into XS advanced authentication, you must create a service instance of the xsuaa service and bind it to the application router and the application containers. From the OAuth 2.0 perspective, the containers (for example the node.js container) are OAuth 2.0 resource servers. Their container security API validates the access tokens against the UAA.

The UAA uses OAuth 2.0 access tokens with the JSON web token (JWT) format for authenticating at the containers, the OAuth 2.0 resource servers.

Principal propagation is needed in the SAP HANA database, as the business users are “virtual database users”, and the database connection uses a technical user. The SAP HANA database is a dedicated component that is called by SAP HANA XS advanced model. For using OAuth 2.0, the SAP HANA database needs to be able to receive and evaluate OAuth 2.0 access tokens. For this reason, the SAP HANA database uses SAML 2.0 bearer tokens.

### Related Information

<http://ietf.org/> ↗

## 11.1.2 Building an Authorization Concept for Users of an XS Advanced Application Instance

Business users in an SAP HANA XS advanced application should have different authorizations because they work in different jobs.

For example in the framework of a leave request process, there are employees who want to create and submit leave requests, managers who approve or reject, and payroll administrators who need to see all approved leave requests to calculate the leave reserve.

The authorization concept of a leave request application should cover the needs of these three employee groups. This authorization concept includes elements such as roles, scopes, and attributes.

## Related Information

[Example Scenario for Leave Request Authorizations \[page 761\]](#)

### 11.1.3 Example Scenario for Leave Request Authorizations

You want to create roles and authorizations for the agents in a leave request work flow.

This work flow includes the following agents:

- Employee
- Manager (approver)
- Payroll administrator

## Work Flow

The example used here to explain the different authorizations required for the different people in a leave-request scenario assumes the following work-flow:

Table 124: Leave-Request Work Flow

Step	Action	Role
1	Create a leave request	Employee
2	Approve a leave request	Manager
3	Calculate available leave reserves	Payroll Administrator

## Employee Authorizations

The employee John Doe needs the authorizations to enable him to create, edit, view, and delete his own requests for leave. He also needs to be able to delete an approved leave requests, for example, if due to a change of plans he decided to go to work on that day after all.

Table 125: Employee Authorizations

Name	Role	Authorization	Attribute
John Doe	Employee	Create	Own
		Edit	Own
		View	Own

Name	Role	Authorization	Attribute
		Delete	Own

## Manager Authorizations

John Doe's manager, Julia Moore, needs the authorizations that enable her to approve John's leave requests and edit them; she cannot, however, delete them. In addition, she needs the authorization that allows her (as a manager) to view, approve, and edit the leave requests of all employees in her department. Since she is also an employee, she needs to be able to create, view, edit, and delete her own leave requests.

Table 126: Manager (Approver) Authorizations

Name	Role	Authorization	Attribute
Julia Moore	Employee	Create	Own
		Edit	Own
		View	Own
		Delete	Own
	Manager	Approve	Managed employees
		Edit	Managed employees
		View	Managed employees

## Payroll Administrator Authorizations

A payroll administrator of the human resources department is only interested in viewing all approved leave requests. He needs the total number of approved leave requests and of days off so that he can feed them into the payroll system, which will calculate the leave reserve. He is not involved in any way in the approval process. Since the payroll administrator, Mark Smith, is an employee as well, he also needs the authorization to create, view, edit, and delete his own leave requests.

Table 127: Payroll Administrator Authorizations

Name	Role	Authorization	Attribute
Mark Smith	Employee	Create	Own
		Edit	Own
		View	Own

Name	Role	Authorization	Attribute
		Delete	Own
	Payroll Administrator	View	All employees

As we can see, even a simple example such as the one described here involves a number of different authorizations defined in the corresponding attributes. However, the case could be extended further, for example, to enable the manager to view the leave of employees in related departments. With this additional authorization, the manager can see who is available as a substitute for the employees during their respective vacation.

An administrator in the XS advanced application can use role templates, scopes, and attributes to shape the roles so that they match the needs and requirements of your organization.

## Related Information

[Maintaining Application Security in XS Advanced \[page 758\]](#)

[Building an Authorization Concept for Users of an XS Advanced Application Instance \[page 760\]](#)

## 11.2 Create the Security Descriptor for Your XS Advanced Application

The security descriptor defines details of an application's security-related dependencies.

### Prerequisites

- File name  
`xs-security.json`
- Location  
Either of the following folders:
  - Application root folder, for example, `/path/appname/`
  - Application security folder, for example, `/path/appname/security`
- Format  
JavaScript Object Notation (JSON)

## Context

The `xs-security.json` file defines the security and deployment options for an application; the information in the application-security file is used at application-deployment time, for example, for authentication and authorization.

## Procedure

1. Create the application resource-file structure.

For example, `/path/appname/`

2. Create a sub-folder for the JavaScript security files.

The security sub-folder should be located in the application root folder, for example, `/path/appname/security`.

3. Create the application security file.

The application security file `xs-security.json` can be located either in the application root folder `/path/appname/` or the security folder `/path/appname/security`.

4. Define details of the application's security-related dependencies.

The contents of the `xs-security.json` file must follow the required JSON syntax.

## Related Information

[The Application Security Descriptor \[page 764\]](#)

[Application Security Descriptor Configuration Syntax \[page 767\]](#)

### 11.2.1 The Application Security Descriptor

A file that defines the details of the authentication methods and authorization types to use for access to your application.

The `xs-security.json` file uses JSON notation to define the security options for an application; the information in the application-security file is used at application-deployment time, for example, for authentication and authorization. Applications can perform scope checks (functional authorization checks with Boolean result) and checks on attribute values (instance-based authorization checks). The `xs-security.json` file declares the scopes and attributes on which to perform these checks. This enables the User Account and Authentication (UAA) service to limit the size of an application-specific JSON Web Token (JWT) by adding only relevant content.

Scope checks can be performed by the application router (declarative) and by the application itself (programmatic, for example, using the container API). Checks using attribute values can be performed by the

application (using the container API) and on the database level using the data control language SAP HANA DCL.

The contents of the `xs-security.json` are used to configure the OAuth 2.0 client; the configuration is shared by all components of an SAP multi-target application. The contents of the `xs-security.json` file cover the following areas:

- Authorization scopes  
A list of limitations regarding privileges and permissions and the areas to which they apply
- Attributes  
A list of as-yet undefined information or sources (for example, the name of a cost center)
- Role templates  
A description of one or more roles to apply to a user and any attributes that apply to the roles

The following example shows a simple application-security file:

### Sample Code

#### Example `xs-security.json` File

```
{  
  "xsappname" : "node-hello-world",  
  "scopes" : [ {  
      "name" : "$XSAPPNAME.Display",  
      "description" : "display" },  
      {  
        "name" : "$XSAPPNAME.Edit",  
        "description" : "edit" },  
        {  
          "name" : "$XSAPPNAME.Delete",  
          "description" : "delete" }  
    ],  
  "attributes" : [ {  
      "name" : "Country",  
      "description" : "Country",  
      "valueType" : "string" },  
      {  
        "name" : "CostCenter",  
        "description" : "CostCenter",  
        "valueType" : "int" }  
    ],  
  "role-templates": [ {  
      "name" : "Viewer",  
      "description" : "View all books",  
      "scope-references" : [  
        "$XSAPPNAME.Display" ],  
      "attribute-references": [ "Country" ]  
    },  
    { "name" : "Editor",  
      "description" : "Edit, delete books",  
      "scope-references" : [  
        "$XSAPPNAME.Edit",  
        "$XSAPPNAME.Delete" ],  
      "attribute-references" : [  
        "Country",  
        "CostCenter" ]  
    }  
  ]  
}
```

## Scopes

Scopes are functional authorizations that are assigned to users by means of security roles, which are mapped to the user group(s) to which the user belongs. Scopes are used for authorization checks by the application router and the application's HDI container. The authorization checks can be performed declaratively (for example, by the application router or in the Java run-time container) or programmatically, for example, in either the Java or the Node.js run-time containers.

- Application router

URL prefix patterns can be defined and a scope associated with each pattern. If one or more matching patterns are found when a URL is requested, the application router checks whether the OAuth 2.0 access token included in the request contains at least the scopes associated with the matching patterns. If not all scopes are found, access to the requested URL is denied. Otherwise, access to the URL is granted.

- Application container

The container security API includes the `hasScope` method that allows you to programmatically check whether the OAuth 2.0 access token used for the current request has the appropriate scope. Among other things, the access token contains a list of scopes that the user is allowed to access in the current context. Each scope name must be unique in the context of the current XS advanced UAA installation.

## Attributes

You can define attributes so that you can perform checks based on a source that is not yet defined. Attributes are used to define **instance**-based authorizations, which change according to the context, for example, location, time-zone, or cost center. In the example `xs-security.json` file included here, the check is based on a cost center, whose name is not known at design time, because the name of the cost center differs according to context.

## Role Templates

A role template describes a role and any attributes that apply to the role.

It is important to remember that a role template must be instantiated; you can do this with the role-building tools provided in the [XS Administration Tools](#). This is especially true with regards to any attributes defined in the role template and their concrete values, which are subject to customization and, as a result, cannot be provided automatically. Role templates that only contain local scopes can be instantiated without user interaction. The same is also true for foreign scopes where the scope "owner" has declared his consent in a kind of white list (for example, either for "public" use or for known "friends").

### **i** Note

If a role template does not contain any attributes, then a role is generated automatically with the same name as the role template. Otherwise, a role based on a role template can be generated using the XS administration tools for XS advanced.

The version information defined in the role template can be used to check if an instantiated role needs to be updated. This assumes that the instantiated role contains not only a reference to the role template from which it was derived but also the corresponding version information.

**i** Note

The resulting application-specific role instance needs to be assigned to one or more user groups.

## Related Information

[Create the Security Descriptor for Your XS Advanced Application \[page 763\]](#)

[Application Security Descriptor Configuration Syntax \[page 767\]](#)

## 11.2.2 Application Security Descriptor Configuration Syntax

The syntax required to set the properties and values defined in the `xs-security.json` application-security description file.

 Sample Code

XS Advanced Security Descriptor (`xs-security.json`)

```
{  
  "xsappname" : "node-hello-world",  
  "scopes" : [ {  
      "name" : "$XSAPPNAME.Display",  
      "description" : "display" },  
      {  
        "name" : "$XSAPPNAME.Edit",  
        "description" : "edit" },  
        "grant-as-authority-to-apps" : ["RequestingApp"]  
      {  
        "name" : "$XSAPPNAME.Delete",  
        "description" : "delete" }  
    ],  
  "attributes" : [ {  
      "name" : "Country",  
      "description" : "Country",  
      "valueType" : "string" },  
      {  
        "name" : "CostCenter",  
        "description" : "CostCenter",  
        "valueType" : "string" }  
    ],  
  "role-templates": [ {  
      "name" : "Viewer",  
      "description" : "View all books",  
      "scope-references" : [ "$XSAPPNAME.Display" ],  
      "attribute-references" : [ "Country" ]  
    },  
    {  
      "name" : "Editor",  
      "description" : "Edit, delete books",  
      "scope-references" : [ "$XSAPPNAME.Edit",
```

```
        "attribute-references" : ["Country", "CostCenter"]
    }
]
"authorities": ["$ACCEPT_GRANTED_AUTHORITIES"]
}
```

## xsappname

Use the `xsappname` property to specify the name of the SAP HANA XS application(s) to which the security description applies.

### Sample Code

```
"xsappname" "<app-name>,"
```

The `xsappname` property represents a developer-defined part of a system-generated OAuth Client ID. Since an OAuth Client ID must be unique within a given identity zone, the `xsappname` property must be defined in such a way as to guarantee the uniqueness of the corresponding OAuth Client ID.

### Tip

It is important to ensure that the name defined in `xsappname` can be easily recognized by other users when configuring additional security components, for example, when XS advanced administrators are using the SAP HANA XS Advanced Admin Tools to create roles from role templates.

## Naming Conventions

Bear in mind the following restrictions regarding the length and content of an application name in the `xs-security.json` file:

- The following characters can be used in an XS advanced application name: “aA”-“zZ”, “0”-“9”, “-” (dash), “\_” (underscore), “/” (forward slash), and “\” (backslash)
- The maximum length of an XS advanced application name is 128 characters.

## scopes

In the application-security file (`xs-security.json`), the `scopes` property defines an array of one or more security scopes that apply for an application. You can define multiple `scopes`; each scope has a name and a short description. The list of scopes defined in the `xs-security.json` file is used to authorize the OAuth client of the XS application with the correct set of **local** and **foreign** scopes; that is, the permissions the application requires to be able to respond to all requests.

## Sample Code

Defining Local Scopes in the Security Descriptor

```
"scopes": [
  {
    "name" : "$XSAPPNAME.Display",
    "description" : "display" },
  {
    "name" : "$XSAPPNAME.Edit",
    "description" : "edit" },
  {
    "name" : "$XSAPPNAME.Delete",
    "description" : "delete" }
],
```

Scopes are mostly “local”; that is, application-specific. Local scopes are checked by the application's own application router or checked programmatically within the application's run-time container. In the event that an application needs access to other SAP HANA XS services on behalf of the current user, the provided access token needs to contain the necessary “foreign” scopes. Foreign scopes are not provided by the application itself; they are checked by other sources outside the context of the application.

In the `xs-security.json` file, “local” scopes must be prefixed with the variable `<$XSAPPNAME>`; at run time, the variable is replaced with the name of the corresponding local application name. “Foreign” scopes, on the other hand, must be prefixed with the name of the “foreign” application itself.

### Tip

The variable `<$XSAPPNAME>` is defined in the application's deployment manifest description (`mtad.yaml` or `manifest.yml`).

## Granting Authorization Scopes to other Applications

If two XS advanced applications are bound to the User Account and Authorization (UAA) service, an authorization scope defined in one application can be granted to the second application on request. The application requesting the scope authority must specify this acceptance (of externally defined scopes) in the “authorities” section of its own security descriptor, and the application granting the scope authority must flag the specified scope as “grantable” in its own security descriptor, as illustrated in the following example:

## Sample Code

Defining Local Scopes in the Security Descriptor

```
"scopes": [
  {
    "name" : "$XSAPPNAME.Display",
    "description" : "display" },
  {
    "name" : "$XSAPPNAME.Edit",
    "description" : "edit" },
  {
    "name" : "$XSAPPNAME.Delete",
    "description" : "delete" },
  {
    "name" : "$XSAPPNAME.ForeignCall",
    "description" : "Enable calls into scope-granting app",
    "grant-as-authority-to-apps" : ["RequestingApp"]
  }
]
```

],

## Naming Conventions

Bear in mind the following restrictions regarding the length and content of a scope name:

- The following characters can be used in a scope name: “aA”-“zZ”, “0”-“9”, “-” (dash), “\_” (underscore), “/” (forward slash), “\” (backslash), “:” (colon), and the “.” (dot)
- Scope names cannot start with a leading dot “.” (for example, .myScopeName1)
- The maximum length of a scope name, including the fully qualified application name, is 193 characters.

## attributes

In the application-security file (`xs-security.json`), the `attributes` property enables you to define an array listing one or more attributes that are applied to the role templates also defined in the `xs-security.json` file. You can define multiple attributes.

### Sample Code

Defining Attributes in the XS Advanced Application's Security Descriptor

```
"attributes" : [
    {
        "name" : "Country",
        "description" : "Country",
        "valueType" : "string" },
    {
        "name" : "CostCenter",
        "description" : "CostCenter",
        "valueType" : "string" }
],
```

The `valueType` key can take the following values:

Table 128: `valueType` Parameter

Key	Description	Example
<code>name</code>	The name of the attribute with a value to apply when building the role template	Country
<code>description</code>	A short summary of the attribute defined	Country
<code>valueType</code>	The type of value expected for the defined attribute; possible values are: “string”, “int” (integer), or “date”	int

## Naming Conventions

Bear in mind the following restrictions regarding the length and content of attribute names in the `xs-security.json` file:

- The following characters can be used to declare an xs-security attribute name in XS advanced: “aA”-“zZ”, “0”-“9”, “\_” (underscore)
- The maximum length of an XS advanced security attribute name is 64 characters.

## role-templates

In the application-security file (`xs-security.json`), the `role-templates` property enables you to define an array listing one or more roles (with corresponding scopes and any required attributes), which are required to access a particular application module. You can define multiple `role-templates`, each with its own scopes and attributes.

### Sample Code

Defining Role Templates in the XS Advanced Application's Security Descriptor

```
"role-templates": [
  {
    "name" : "Editor",
    "description" : "View, edit, delete books",
    "scope-references" : [
      "$XSAPPNAME.Edit", "$XSAPPNAME.Delete"
    ],
    "attribute-references": [
      "Country", "CostCenter"
    ]
  },
]
```

A role template must be instantiated. This is especially true with regards to any attributes defined in the role template and the concrete attribute values, which are subject to customization and, as a result, cannot be provided automatically. Role templates that only contain “local” scopes can be instantiated without user interaction. The same is also true for “foreign” scopes where the scope **owner** has declared consent in a kind of white list (for example, either for “public” use or for known “friends”).

### Note

The resulting (application-specific) role instances need to be assigned to the appropriate user groups.

Table 129: role-template parameters

Key	Description	Example
name	The name of the role to build from the role template	“Viewer”
description	A short summary of the role to build	“View all books”

Key	Description	Example
scope-references	The security (authorization) <b>scope</b> to apply to the application-related role	"\$XSAPPNAME.Display"
attribute-references	One or more attributes to apply to the built role	[ "Country", "CostCenter" ]

## Naming Conventions

Bear in mind the following restrictions regarding the length and content of role-template names in the `xs-security.json` file:

- The following characters can be used to declare an xs-security role-template name in XS advanced: "aA"- "zZ", "0"-“9”, “\_” (underscore)
- The maximum length of an XS advanced role-template name is 64 characters.

## authorities

To enable one (requesting) application to accept and use the authorization scopes granted by another application, each application must be bound to its own instance of the XS UAA service; this is required to ensure that the applications are registered as OAuth 2.0 clients at the UAA. You must also add an “authorities” section to the security descriptor file of the application that is requesting an authorization scope. In the “authorities” section of the requesting application’s security descriptor, you can either specify that **all** scope authorities configured as grantable in the granting application should be accepted by the application requesting the scopes, or alternatively, only individual, named, scope authorities:

- Request and accept **all** authorities flagged as "grantable" in the granting application's security descriptor

### Sample Code

Specifying Scope Authorities in the XS Advanced Requesting Application's Security Descriptor

```
"authorities": ["$ACCEPT_GRANTED_AUTHORITIES"]
```

- Request and accept only the **specified** scope authority that is flagged as grantable in the specified granting application's security descriptor:

### Sample Code

Specifying Scope Authorities in the XS Advanced Requesting Application's Security Descriptor

```
"authorities": ["<GrantingApp>.ForeignCall", "<GrantingApp2>.ForeignCall", ]
```

Since both the requesting application and the granting application are bound to UAA services using the information specified in the respective application's security descriptor, the requesting application can accept and use the specified authority from the granting application. The requesting application is now authorized to access the granting application and perform some tasks.

### Note

The granted authority always includes the prefixed name of the application that granted it. This information tells the requesting application which granting application has granted which set of authorities.

The scope authorities listed in the requesting application's security descriptor must be specified as "grantable" in the granting application's security descriptor.

### Tip

To flag a scope authority as "grantable", add the "grant-as-authority-to-apps" property to the respective scope in the granting application's security descriptor.

## Related Information

[Create the Security Descriptor for Your XS Advanced Application \[page 763\]](#)

[XS Advanced Application Router Configuration Syntax \[page 725\]](#)

## 11.3 Create an Instance of the XS UAA Service

Use the service broker to create an instance of the `xsuaa` service in XS advanced.

## Context

If you want to grant users access to an XS advanced application, you must create an instance of the XS advanced User Account and Authorization service (XSUAA) for this application; the XSUAA service instance acts as an OAuth 2.0 client for the bound application.

## Procedure

1. Create a service instance based on the `xsuaa` service and using the service plan "default" and the security descriptors defined in the `xs-security.json` file.

```
xs create-service <service_name> <service_plan> <service_instance_name> -c xs-security.json
```

### Example

```
xs create-service xsuaa default authorizationtest-uaa -c xs-security.json
```

The name of the OAuth 2.0 client you created is similar to the following example:

sb-577a76db-21f9-412b-ad4e-a740f3991136.

## 2. Display a list of the OAuth 2.0 clients that are available.

To check that your new service instance is among the list of currently available OAuth 2.0 clients, log on to the UAA using the following URL and a known UAA user:

```
http://<host_name>:<uaa_port>
```

- Log on to the XS controller.

```
xs login -a <api_endpoint> -u <username> -p <password>
```

- List the applications and services currently running.

The information displayed in the console includes URL and port numbers.

```
xs apps
```

```
xs services
```

## 3. Update details of an existing XSUAA service's configuration.

The XSUAA service for user authentication is configured using the security descriptor `xs-security.json`. You can change selected details of the security configuration and update the XSUAA service with the modified configuration.

To modify details of an existing XSUAA service's security configuration, use the `xs update-service` command as follows:

```
xs update-service <SERVICE_INSTANCE> -c xs-security.json
```

### Restriction

It is not possible to update the service plan assigned to an instance of the XS User Account and Authentication (XSUAA) service.

When updating the configuration of the XSUAA service, you can add or delete any defined scopes, security-related attributes, or role templates. However, **modifications** are only permitted to the following elements of an existing service's security configuration:

- A security scope's description and "granted applications"
- A security attribute's description  
When updating the XSUAA service, it is not permitted to change the value type of an attribute defined in the `xs-security.json` application security descriptor.
- A role template's description, scope reference, or attribute reference

### Restriction

When updating the configuration of an XSUAA service instance, it is only possible to delete an attribute reference; it is not possible to add or modify them.

## Related Information

[Create a Service Instance \[page 681\]](#)

[Modifications and Restrictions for XSUAA Service Updates \[page 775\]](#)

### 11.3.1 Modifications and Restrictions for XSUAA Service Updates

Restrictions apply to which parts of the XS advanced security description you can change and apply when updating an instance of the XSUAA service.

The XS advanced service broker allows you to update the configuration of an existing service, for example, to change the service plan associated with a service. You can also use a service update to modify the XSUAA service's security configuration, for example, by adding, deleting, or changing the artifacts defined in the `xs-security.json` security descriptor. If you want to use a service-update operation to make changes to the XSUAA service's security configuration, bear in mind that certain restrictions apply, as illustrated in the following table:

Table 130: Permitted Changes to `xs-security.json` when Updating the XSUAA Service

Security Artifacts	Add	Delete	Change
Authorization Scope	✓	✓	You can only modify the scope's: <ul style="list-style-type: none"><li>description</li><li>granted-apps</li></ul>
Attribute	✓	✓	You can only modify the attribute's: <ul style="list-style-type: none"><li>description</li></ul> <p><b>i Note</b> It is not possible to change an attribute's <code>valueType</code> during a service-update operation.</p>
Role template	✓	✓	You can only modify a role template's: <ul style="list-style-type: none"><li>description</li><li>scope-references</li><li>attribute-references</li></ul> <p><b>⚠ Restriction</b> An attribute reference can only be deleted; it cannot be modified during a service-update operation.</p>

## Related Information

[Create the Security Descriptor for Your XS Advanced Application \[page 763\]](#)

[The Application Security Descriptor \[page 764\]](#)

[Application Security Descriptor Configuration Syntax \[page 767\]](#)

## 11.4 Bind the XS UAA Service Instance to the XS Advanced Application

You must bind the UAA service instance you created to the XS advanced application that is going to use it.

### Context

You must bind the service instance that acts as OAuth 2.0 client to your XS advanced application. The name of the service is defined in the `services` property in the application's manifest file (`manifest.yml`). This manifest file is located in the root folder of your XS advanced application, for example `node-hello-world`.

### Procedure

Bind an application to a service instance:

```
xs bind-service <APP_NAME> <SERVICE_INSTANCE>
```

#### Example

```
xs bind-service hello-world xsuaa
```

You can now deploy your application with the authorization artifacts that were created earlier.

## Related Information

[Create a Service Instance \[page 681\]](#)

[XS Advanced Application Routes and Destinations \[page 739\]](#)

[XS CLI: Services Management \[page 874\]](#)

## 11.5 Assign Functional Authorization Scopes for an XS Advanced Application

Declare authorization scopes for your application in the security descriptor and then assign the security descriptor file to the application.

### Context

The functional authorization scopes for your application must be known to the User Account and Authorization (UAA) service. To do this, you declare the scopes of your application in the security descriptor file named `xs-security.json` and then you assign the security descriptor file to the application.

Once you have created an OAuth 2.0 client using the service broker, you specify the `xs-security.json` file that is relevant for your application. Using this security descriptor file, the OAuth 2.0 client that has been created for your application receives permission to access the scopes of your application automatically.

### Procedure

1. Define the authorization scopes for your application.

Authorization scopes for XS advanced applications are defined in the application's security descriptor (`xs-security.json`). You also need to provide a reference for each authorization scope in a corresponding role template. The following example of an `xs-security.json` file defines scopes for viewing and creating data:

#### Sample Code

Authorization Scopes in the Security Descriptor (`xs-security.json`)

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

2. Assign the `xs-security.json` file to the XS advanced application for which you defined the authorization scopes.

You can use the following command to :

```
xs create-service <service_name> <service_plan> -c <security_descriptor_file>
```

#### Example

```
xs create-service xsuaa default -c xs-security.json
```

This creates an instance of the UAA service with the scopes defined in the security descriptor and ensures it is bound to the corresponding application.

#### Note

After the developer has created the role templates and deployed them to the relevant application, it is the administrator's task to use the role templates to build roles, aggregate the roles into role collections, and assign the role collections to business users in the application. For more information, see the section about maintaining the XS advanced run-time environment in the *SAP HANA Administration Guide*.

## Related Information

[Create the Security Descriptor for Your XS Advanced Application \[page 763\]](#)

### 11.5.1 Authorization in SAP HANA XS Advanced

Authorization restricts access to resources and services based on defined user permissions.

SAP HANA XS advanced applications contain content that is deployed to different database containers. Access to the content requires not only user authentication but also the appropriate **authorization**.

The access-control process controlled by the authorization policy can be divided into the following two phases:

- Authorization  
Defined in the deployment security descriptor (`xs-security.json`), where access is authorized
- Policy enforcement  
The general rules by which requests for access to resources are either approved or disapproved.

Access enforcement is based on user identity and performed in the following distinct application components:

- Application router  
After successful authentication at the application router, the application router starts an authorization check (based on scopes).
- Application containers  
For authentication purposes, containers, for example the node.js and Java containers, receive an HTTP header `Authorization: Bearer <JWT token>` from the application router; the token contains details of the user and scope. This token **must** be validated using the XS advanced Container Security API. The validation checks are based on scopes and attributes defined in the XS advanced deployment security descriptor (`xs-security.json`).

- SAP HANA database  
Every XS advanced application uses its own dedicated database schema and its own dedicated (technical) database user for requests to connect to the database (for example, to manage persistent application data). This architecture ensures data isolation between the XS advanced applications. The technical database user is assigned the SAP HANA privileges required by the application; this defines the boundaries for all SAP HANA XS advanced users using that application.

 Note

Instance-based authorization checks using the data control language (DCL) are also supported; the checks are based on attributes, for example, those defined in the XS deployment security descriptor (`xs-security.json`).

## Related Information

[Maintaining Application Security in XS Advanced \[page 758\]](#)

[The Application Security Descriptor \[page 764\]](#)

[Scopes for Functional Authorization Checks \[page 779\]](#)

[The Data Control Language \[page 780\]](#)

### 11.5.1.1 Scopes for Functional Authorization Checks

Authorization in the application router and container are handled by scopes. Scopes are groupings of functional organizations defined by the application.

From a technical point of view, the resource server (the relevant security container API) provides a set of services (the resources) for functional authorizations. The functional authorizations are organized in scopes.

Scopes are used to define the authorizations required by business users for access to the features provided by a specific SAP HANA XS advanced application, for example: view, change, or delete. Scopes are created when you deploy or update the application. Scopes are defined in an application's security descriptor file `xs-security.json`. It is possible to define so-called "local" (application-specific) scopes and, if necessary, also any "foreign" scopes, which are valid for other applications, too.

 Tip

Scopes can be checked declaratively or programmatically. For example, a declarative check of the scopes can be performed by the application router or in the Java run-time container; a programmatic check can be performed in either the Java or Node.js run-time containers.

The security descriptor is also used to define a set of role templates for a named application; the role templates contain the authorizations ("scope-references") for business users' actions, for example: viewing, editing, or deleting data. Information retrieved from the user's identity (such as location, department, or cost center) is stored in attributes.

After the developer has created the role templates and deployed them to the relevant application, it is the administrator's task to use the role templates to build roles, aggregate roles into role collections, and assign the role collections to business users in the application.

To assign scopes to an application, you need to perform the following steps:

1. Create an instance of the User Account and Authentication (UAA) service.

You can use the service broker to create an instance of the UAA service. This ensures the creation of a new OAuth 2.0 client in the UAA.

2. Assign scopes for the application.

Scopes are configured and assigned in the application descriptor file `xs-security.json`, which can be placed in the root folder of the application or in a `security` folder in your XS advanced project along with a `package.json` file.

### → Tip

For access to SAP HANA objects, the privileges of the technical user (container owner) apply; for business data, instance-based authorizations using DCL are used.

The following example, of a simple security descriptor for an XS advanced application shows how to define authorization scopes and reference them in role templates:

#### Sample Code

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

## Related Information

[The Application Security Descriptor \[page 764\]](#)

### 11.5.1.2 The Data Control Language

The Data Control Language (DCL) is the language used in Core Data Services (CDS) to define instance-based authorization models which control access to data based on user attributes, for example: a user name, an organizational unit, or a country. Attributes can be defined by applications, and it is possible to map SAML 2.0 attributes with SAP HANA XS advanced attributes. To use them in the XS advanced application, the developers need to deploy the artifacts to the XS advanced application.

CDS DCL is designed as an extension to SQL's DCL; among other things, DCL introduces the notion of predicated privileges that allow dynamic grants based on the attribute values of entity instances. DCL uses **rules** to restrict access to data, as illustrated in the following example:

### Sample Code

Rules in CDS DCL

```
grant select on ddl.salesOrderView  
    where customerOrgUnit = SESSION_CONTEXT('XS_ORGUNIT');
```

XS advanced applications declare attributes ("ORGUNIT" in the example above) which they need for instance-based authorizations, so that only users registered in the named organizational unit are allowed (when authenticated) to access the information. These attributes are integrated in rules using `SESSION_CONTEXT('XS_<attribute_name>')`. These attributes are obtained from the Attribute Provider Service during the authentication process and passed to the SAP HANA database by a SAML bearer JWT token.

### Note

In XS advanced, this authorization concept coexists with the authorization concept based on role templates deployed in XS advanced applications. You can use the concepts together or independently of another. The application-based authorization concept is based on role templates. The authorization check of a business user occurs on the XS advanced application instance.

## Related Information

[Create a CDS Access-Policy Document \(XS Advanced\) \[page 332\]](#)

## 11.6 Enable Applications to Perform Tasks with the Authority of Other Applications

You can use authorization scopes to enable an application to perform tasks on behalf of another application and in the second application's container.

## Context

In XS advanced, an application can be configured to grant the authorization defined in its own scopes to another application on request, for example, to allow the requesting (external) application to perform tasks on behalf of the local application; that is, with the authority of the local application, and in the local application's

container. To configure one application to perform tasks with the authority of another application, perform the following steps:

## Procedure

1. Enable an XS advanced application to grant scope authorities to another (requesting) XS advanced application.

In the security descriptor of the application granting the use of a scope authority, use the "grant-as-authority-to-apps" property to configure individual scopes to be "grantable" to another authorized application on request.

```
"scopes": [
  {
    "name": "$XSAPPNAME.foreigncall",
    "description": "Enable Calls into Granting App",
    "grant-as-authority-to-apps": ["RequestingApp"]
  },
]
```

2. Bind the application granting the scope authority to its respective UAA service.

After you make the necessary changes to the scopes in the security descriptor of the application that is granting the scope authorities, you must rebind the application to its UAA service instance.

```
cf bind-service GrantingApp xsuaa -c xs-security.json
```

### ➔ Tip

If no such instance of the `xsuaa` service exists, then create it using the `xs create-service` command.

3. Enable the requesting application to accept the scope authorities granted by another application.

In the security descriptor of the application requesting the authorization scopes granted by another application, list the requested scopes and the application granting them in the "authorities" property. You can configure **all** granted scopes to be accepted and used, or individual named scopes, as indicated below:

- Accept all granted authorities:

```
"authorities": ["$ACCEPT_GRANTED_AUTHORITIES"],
```

- Accept the specified authorities from the named applications:

```
"authorities": ["<GrantingApp>.ForeignCall", "<GrantingApp2>.ExternalCall"],
```

4. Bind the requesting application to its respective UAA service instance.

After you have made the necessary changes to the security descriptor of the application requesting, accepting, and using the scope authorities granted by the first application, you must rebind the requesting application to its UAA service instance so that it updates its configuration with the scope changes made to the security descriptor (`xs-security.json`).

```
cf bind-service RequestingApp xsuaa -c xs-security.json
```

### ➔ Tip

If no such instance of the `xsuaa` service exists, then create it using the `xs create-service` command.

5. Test the requesting and granting of the configured scopes and authorities.

Start all applications and test that the requesting application receives the granted authority and can perform the planned task with the required authorization in the granting application's container.

## Related Information

[Application Security Descriptor Configuration Syntax \[page 767\]](#)

[Create an Instance of the XS UAA Service \[page 773\]](#)

[Bind the XS UAA Service Instance to the XS Advanced Application \[page 776\]](#)

## 11.6.1 Authorization Scopes and Authorities

An overview of the configuration required to enable one XS advanced application to perform a task with the authority of another XS advanced application.

In the XS advanced run-time environment, it is possible to authorize one application to make calls to other applications to trigger an action on the other application's behalf. For example, you could configure one application to trigger a scheduled job that cleans up unwanted logs and collects run-time "garbage" on behalf (and with the authority) of another application. In this scenario, one application grants the authorities, and another application requests the authorities, accepts them, and uses the authorities to access the granting application and perform tasks on the granting application's behalf. By enabling applications to grant and use authorities in this way, it is possible to ensure that not only the communication between the applications but also the execution of the task itself run securely; that is, with the required permissions and in the correct application run-time container.

To enable the exchange and use of authorities between applications, at least two applications are required: one that grants the authority, and a second, named, application that performs the desired action on behalf of the application granting the authority. In the security descriptor (`xs-security.json`) of the application granting a named scope as an authority, you need to flag the specified scope as "grantable", for example, using the property "`grant-as-authority-to-apps`", as illustrated in the following example:

### Code Syntax

Granting Scopes as Authorities (`xs-security.json` of Granting Application)

```
"scopes": [
  {
    "name": "$XSAPPNAME.foreigncall",
    "description": "Enable Calls into Granting App",
    "grant-as-authority-to-apps": ["RequestingApp"]
  }
]
```

In the application that requires the granted authority, you need to add the property “authorities” to the requesting application’s security descriptor and either list the individual, named authorities that can be used, or specify that **all** authorities granted by another application can be accepted and used, as illustrated in the following example:

### Code Syntax

Requesting and Using Scopes as Authorities (`xs-security.json` of Requesting Application)

```
"authorities": ["$ACCEPT_GRANTED_AUTHORITIES"],
```

Each of the applications involved in the exchange of authorities must be bound to its own an instance of the XS advanced User Account and Authorization (UAA) service; the service binding creates the respective OAuth 2.0 clients and enables secure communication between the applications involved in the exchange of authorities. The UAA binding also enables the application requesting an authority to authenticate itself at the granting application **on its own behalf** using the OAuth 2.0 client credentials.

### Tip

When an OAuth 2.0 client acts on behalf of a user, it requires authorization scopes, which are defined in its own security descriptor, (`xs-security.json`). When an OAuth 2.0 client acts on its **own** behalf and no user is involved, the client requires “authorities”, which are defined in the security descriptor of another application - the granting application. An authority is a scope that is flagged as “grantable” so that it can be granted to another application in order to enable the requesting application to perform tasks with the authority of the granting application.

## Related Information

[Enable Applications to Perform Tasks with the Authority of Other Applications \[page 781\]](#)

[Application Security Descriptor Configuration Syntax \[page 767\]](#)

[The Application Security Descriptor \[page 764\]](#)

# 12 HDI Artifact Types and Build Plug-ins Reference

The XS advanced deployment infrastructure supports a wide variety of database artifact types, for example, tables, types, views.

In XS advanced, design-time artifacts are distinguished by means of a unique file suffix that must be mapped to an HDI build plug-in. The following example of an abbreviated HDI configuration file (`.hdiconfig`) illustrates how the design-time artifact types `.hdbcalsulationview` and `.hdbcds` are mapped to their corresponding HDI build plug-in:

## Restriction

All “`hdi*`” file-suffixes (for example, “`.hdiconfig`” or “`.hdinamespace`”) are reserved by the SAP HANA Deployment Infrastructure (HDI) and cannot be (re)configured.

## Code Syntax

```
.hdiconfig

{
  "plugin_version" : "0.0.0.0",    // optional, defaults to 0.0.0.0
  "file_suffixes" : {
    "hdbcalsulationview" : {
      "plugin_name" : "com.sap.hana.di.calsulationview",
      "plugin_version": "2.0.0.0" //optional, default=top-level version
    },
    "hdbcds" : {
      "plugin_name" : "com.sap.hana.di.cds",
      "plugin_version": "2.0.0.0"
    },
    "<file suffix 2>" : {
      "plugin_name" : "<plugin name>",
      "plugin_version": "<plugin_version>" // optional
    }
}
```

The values of the `plugin_version` fields are interpreted as the minimal required SAP HANA version, which follows the format `<major>.<minor>.<revision>.<patch>`, for example, `2.0.0.0` for SAP HANA 2.0 SPS00 revision 0 patch 0. The three level plugin version format used in SAP HANA 1.0, for example, `13.1.0`, is still valid; values in the old format will be internally converted to the appropriate four level format for SAP HANA 2.0 revisions.

In SAP HANA 1.0 there was no “top-level” `plugin_version` entry, and the “bottom-level” `plugin_version` entries were mandatory.

From SAP HANA 2.0, the version of all plugins shipped with SAP HANA is the same as (and equal to) the SAP HANA version. Consequently, there should normally be no need for the bottom level `plugin_version` entries, since the result of the compatibility check would depend on the highest entry in use only. To summarize, if no particular minimum SAP HANA version is required, it is sufficient to provide the top-level

entry or no entry at all. However, if present, bottom-level entries override the top-level entry or the default `plugin_version` of "0.0.0.0", respectively.

The following table lists in alphabetical order the design-time artifacts you can develop and deploy with the SAP HANA Deployment Infrastructure (HDI). For more information about the syntax required for a particular type of design-time artifact and the configuration of the corresponding build plug-in, choose the artifact type in the following table.

Table 131: Design-Time HDI Artifact Types

A - P	P - T	T - Z
CSV *	Procedure	Text Configuration
.? (txt, copy only)	Projection View	Text Dictionary
AFL Lang Procedure **	Projection View Configuration	Text Extraction Rule
Analytic Privilege	Public Synonym	Text Extraction Include
Calculation View	Result Cache	Text Extraction Lexicon
Core Data Services	Role	Text Mining Configuration
Constraint	Role Configuration	Trigger
DocStore Collection	Search Rule Set	View
Drop/Create Table	Sequence	Virtual Function
Full Text Index	Synonym	Virtual Function Configuration
Function	Synonym Configuration	Virtual Package
Graph Workspace	Statistics	Virtual Procedure
Index	Structured Privilege	Virtual Procedure Configuration
Map Reduce Job ***	_SYS_BIC Synonym	Virtual Table
Jar	Table	Virtual Table Configuration
Library	Table Data	Properties
Logical Schema Definition	Table Type	Tags

### i Note

\* CSV: Comma-separated-values file

\*\* AFL Lang Procedure: Application function library language procedure

\*\*\* Map Reduce Job: Deprecated, use Virtual Package instead

## Related Information

[The HDI Container Configuration File \[page 180\]](#)

[HDI Run-Time Name spaces \[page 185\]](#)

## 12.1 AFL Language Procedures (.hdbafllangprocedure)

Transform a design-time Application Function Library (AFL) language procedure resource into a corresponding database procedure object.

### Application Function Library (AFL) Language Procedure

The AFLLang Procedure plug-in (`hdbafllangprocedure`) transforms a design-time Application Function Library (AFL) language procedure resource into a corresponding database procedure object that wraps the specified AFL function based on the given table types for the parameters. The file format uses the JSON syntax.

#### Example Artifact Code

The following code shows a simple example of an AFLLang Procedure definition for XS advanced HDI:

##### Code Syntax

```
/src/qa1_f3.hdbafllangprocedure

{
  "procedure" : "com.sap.hana.example::QA1_F3_PROC",
  "area" : "QA",
  "function" : "QA1_F3",
  "parameters" : [
    {
      "type" : "com.sap.hana.example::INPUT_TABLE_TYPE",
      "direction" : "IN"
    },
    {
      "type" : "com.sap.hana.example::OUTPUT_TABLE_TYPE",
      "direction" : "OUT"
    }
  ]
}
```

Specifying the procedure name is optional. If the procedure name is not specified, it is derived from the procedure-definition file name. For example, if the file is “/src/PROC.hdbafllangprocedure” and the name

space is "com.sap.hana.example", then the name of the run-time object is "com.sap.hana.example::PROC".

#### Note

The corresponding AFL C++ libraries must already be installed in the SAP HANA system by an administrator; the libraries cannot be installed at design-time.

### Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plug-in configuration should look like the following example:

#### Code Syntax

```
.hdiconfig

"hbafllangprocedure" : {
    "plugin_name" : "com.sap.hana.di.afllangprocedure",
    "plugin_version": "2.0.0.0"
}
```

### Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.2 Analytic Privileges (.hdbanalyticprivilege)

Transform a design-time XML-based analytic-privileges resource into a analytic-privileges object in the database.

The analytic privileges plugin (hdbanalyticprivilege) transforms a design-time XML-based analytic-privileges resource into a analytic-privileges object in the database. The file format uses the XML syntax. The referenced views must be defined using the WITH STRUCTURED PRIVILEGE CHECK clause.

### Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plugin configuration should look like the following example:

#### Code Syntax

```
.hdiconfig

"hdbanalyticprivilege" : {
    "plugin_name" : "com.sap.hana.di.analyticprivilege",
    "plugin_version": "2.0.0.0"
}
```

```
}
```

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.3 Calculation Views (.hdbcalsulationview)

Transform a design-time calculation view description into a set of view database objects.

The Calculation View plugin (.hdbcalsulationview) transforms a design-time calculation view description into a set of view database objects. The file format uses an XML-based syntax.

### Example Artifact Code

The following code shows a simple example of a CDS model (context, entity, and view) for XS advanced HDI:

#### Code Syntax

```
/src/PROJECTION.hdbcalsulationview

<?xml version="1.0" encoding="UTF-8"?>
<Calculation:scenario id="com.sap.hana.example::projection"
    outputViewType="Projection">
<dataSources>
    <DataSource id="D1" type="DATA_BASE_TABLE">
        <resourceUri>com.sap.hana.example::TAB1</resourceUri>
    </DataSource>
</dataSources>
<calculationViews>
    <calculationView xsi:type="Calculation:ProjectionView" id="Projection_1">
        <viewAttributes>
            <viewAttribute id="K1"/>
            <viewAttribute id="K2"/>
        </viewAttributes>
        <calculatedViewAttributes>
            <calculatedViewAttribute datatype="integer" id="CC">
                <formula>40 + 2</formula>
            </calculatedViewAttribute>
        </calculatedViewAttributes>
        <input node="#D1">
            <mapping xsi:type="Calculation:AttributeMapping" target="K1"
source="K1"/>
            <mapping xsi:type="Calculation:AttributeMapping" target="K2"
source="K2"/>
        </input>
    </calculationView>
</calculationViews>
<logicalModel id="Projection_1">
    <attributes>
        <attribute id="K1">
            <keyMapping columnObjectName="Projection_1" columnName="K1"/>
        </attribute>
        <attribute id="K2">
```

```

<keyMapping columnObjectName="Projection_1" columnName="K2"/>
</attribute>
<attribute id="CC">
    <keyMapping columnObjectName="Projection_1" columnName="CC"/>
</attribute>
</attributes>
<calculatedAttributes>
    <calculatedAttribute id="CalcAttr" hidden="false">
        <keyCalculation datatype="INTEGER">
            <formula>40 + 2</formula>
        </keyCalculation>
    </calculatedAttribute>
</calculatedAttributes>
</logicalModel>
</Calculation:scenario>

```

## Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plugin configuration should look like the following example:

### Code Syntax

```
.hdiconfig

"hdbscalculationview" : {
    "plugin_name" : "com.sap.hana.di.calculationview",
    "plugin_version": "2.0.0.0"
}
```

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.4 Constraints (.hdbconstraint)

Transform a design-time constraint into a constraint on database tables.

The foreign-key constraint plugin (.hdbconstraint) transforms a design-time constraint into a constraint on database tables. The file format uses a DDL-style syntax which is similar to the SQL syntax in the corresponding SQL command `ALTER TABLE ADD CONSTRAINT`. However, to be consistent with other DDL-style artifacts, in the design-time definition specified in the .hdbconstraint file, the constraint is specified in the following way: `CONSTRAINT <myConstraint> ON table [...]`, as illustrated in the following example:

### Restriction

The constraint plugin (hdbconstraint) only supports FOREIGN KEY constraints.

## Example Artifact Code

The following code shows a simple example of a foreign-key constraint definition for XS advanced HDI:

### Code Syntax

```
/src/A_CONSTRAINT.hdbconstraint

CONSTRAINT A_CONSTRAINT
ON BASE_TABLE
FOREIGN KEY (FIELD) REFERENCES OTHER_TABLE (FIELD) ON UPDATE CASCADE
```

### Tip

Unique constraints are handled by the `.hbdbindex` plugin.

## Plug-in Configuration

In the configuration file for the HDI container (`.hdiconfig`), the plugin configuration should look like the following example:

### Code Syntax

```
.hdiconfig

"hbdbconstraint" : {
  "plugin_name" : "com.sap.hana.di.constraint",
  "plugin_version": "2.0.0.0"
}
```

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.5 Copy Only (.txt)

Transform an arbitrary design-time resource into a deployed object.

The copy-only plugin transforms an arbitrary design-time resource into a deployed object by copying it from the container's work file system to the deployed file system. Such a design-time resource does not have a representation inside the run-time container. It is only accessible by means of SAP HANA DI APIs.

## Plug-in Configuration

In the configuration file for the HDI container (`.hdiconfig`), the plugin configuration should look like the following example:

### Code Syntax

```
.hdiconfig  
  "txt" : {  
    "plugin_name" : "com.sap.hana.di.copyonly",  
    "plugin_version": "2.0.0.0"  
  }
```

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.6 Core Data Services (.hdbcards)

Transform a design-time SAP HANA CDS resource into the database objects defined in the CDS document.

The SAP HANA Core Data Services (CDS) plugin (`hdbcards`) transforms a design-time SAP HANA CDS resource into the database objects defined in the CDS document, for example, tables, views, types.

### ⚠ Restriction

Text analysis `CONFIGURATION` and `MINING CONFIGURATION` properties for a table column or a full-text index are not supported.

### Example Artifact Code

The following code shows a simple example of a CDS model (context, entity, and view) for XS advanced HDI:

### Code Syntax

```
/src/EandV.hdbcards  
  
namespace com.sap.hana.example;  
using "com.sap.hana.example::CUSTOMERS" as CUSTOMERS;  
context EandV {  
  entity E {  
    key id : Integer;  
  };  
  view V as select from CUSTOMERS {  
    NAME  
  };  
};
```

## Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plugin configuration should look like the following example:

### Code Syntax

```
.hdiconfig

"hdcds" : {
    "plugin_name" : "com.sap.hana.di.cds",
    "plugin_version": "2.0.0.0"
}
```

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.7 Document Store Collections (.hdbcollection)

Transforms a design-time document-collection resource into a collection database object.

The Collection plug-in transforms a design-time, document-collection resource (specified in a .hdbcollection artifact) into a collection database object, which is a data store with the `table_type` “COLLECTION”. The file format required in the .hdbcollection artifact uses a DDL-style syntax that is equivalent to the syntax of the corresponding SQL statement “CREATE COLLECTION”, but without the leading CREATE command. The so-called “collections” are used to store JSON documents in the SAP HANA Document Store (DocStore).

### Example Artifact Code

The following code shows a simple example of a JSON DocStore “collection” definition for XS advanced HDI:

### Code Syntax

```
/src/CUSTOMERS.hdbcollection

COLLECTION TABLE "CUSTOMERS"
```

### Note

The hdbcollection artifact can only be used to create the JSON collection; it cannot be used to insert JSON documents into the collection.

## Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plug-in configuration should look like the following example:

### Code Syntax

```
.hdiconfig

"hdbscollection" : {
  "plugin_name" : "com.sap.hana.di.collection",
  "plugin_version": "2.0.0.0"
}
```

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.8 Full-Text Indexes (.hdbfulltextindex)

Transform a design-time full-text index resource into a full-text index on a database table.

The full-text index plugin (`hdbfulltextindex`) transforms a design-time full-text index resource into a full-text index on a database table. The file format uses a DDL-style syntax which is equivalent to the corresponding syntax in the SQL command `CREATE FULLTEXT INDEX`, although without the leading “`CREATE`” command.

### ⚠ Restriction

With this release it is not possible to use a Text Analysis configuration object in combination with a synchronous full-text index

Text Analysis CONFIGURATION and MINING CONFIGURATION properties reference container-local Text Analysis configuration objects which need to be deployed using the `hdbtext*` artifacts.

### Example Artifact Code

The following code shows a simple example of a full-text index definition for XS advanced HDI:

### Code Syntax

```
/src/CUSTOMER_NAME_FT1.hdbfulltextindex

FULLTEXT INDEX "com.sap.hana.example::CUSTOMER_NAME_FT1"
ON "com.sap.hana.example::CUSTOMERS" (NAME)
```

## Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plugin configuration should look like the following example:

### Code Syntax

```
.hdiconfig

"hbdbfulltextindex" : {
    "plugin_name" : "com.sap.hana.di.fulltextindex",
    "plugin_version": "2.0.0.0"
}
```

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.9 Functions (.hdbfunction)

Transform a design-time function resource into a function database object.

The function plugin (`hdbfunction`) transforms a design-time function resource into a function database object. The file format uses a DDL-style syntax which is equivalent to the syntax of the corresponding SQL command `CREATE FUNCTION` SQL, without the leading “`CREATE`”. The Function plugin handles scalar and table functions, too.

### Note

Virtual functions are handled by the virtual function plugin.

### Example Artifact Code

The following code shows a simple example of a **scalar** function definition for XS advanced HDI:

### Code Syntax

```
/src/SCALAR_FUNC_ADD_MUL.hdbfunction

FUNCTION "com.sap.hana.example::SCALAR_FUNC_ADD_MUL" (X DOUBLE, Y DOUBLE)
RETURNS RESULT_ADD DOUBLE, RESULT_MUL DOUBLE
LANGUAGE SQLSCRIPT
AS
BEGIN
    RESULT_ADD := :X + :Y;
    RESULT_MUL := :X * :Y;
END
```

The following code shows a simple example of a **table** function definition for XS advanced HDI:

#### Code Syntax

```
/src/TABLE_FUNC_SCALE.hdbfunction

FUNCTION "com.sap.hana.example::TABLE_FUNC_SCALE" (VAL CHAR)
RETURNS TABLE (A INT, B INT)
LANGUAGE SQLSCRIPT
AS
BEGIN
    RETURN SELECT A, :VAL * B AS B FROM MYTABLE;
END
```

#### Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plugin configuration should look like the following example:

#### Code Syntax

```
.hdiconfig

"hdbfunction" : {
    "plugin_name" : "com.sap.hana.di.function",
    "plugin_version": "2.0.0.0"
}
```

#### Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.10 Graph Workspaces (.hdbgraphworkspace)

Transform a design-time graph-workspace resource into a graph-workspace object in the database.

The graph workspace plugin (`hdbgraphworkspace`) transforms a design-time graph workspace resource into a graph workspace object in the database. The file format uses a DDL-style syntax that is equivalent to syntax of the corresponding SQL command `CREATE GRAPH WORKSPACE`, without the leading “`CREATE`”.

#### Example Artifact Code

The following code shows a simple example of a graph workspace definition for XS advanced HDI:

#### Code Syntax

```
/src/my_graph_workspace.hdbgraphworkspace

GRAPH WORKSPACE MY_GRAPH_WORKSPACE
```

```
EDGE TABLE THE_EDGE_TABLE
  SOURCE COLUMN SOURCE TARGET COLUMN TARGET KEY COLUMN EDGE_ID
VERTEX TABLE THE_VERTEX_TABLE
  KEY COLUMN VERTEX_ID
```

## Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plugin configuration should look like the following example:

### Code Syntax

```
.hdiconfig

"hdbsgraphworkspace" : {
  "plugin_name" : "com.sap.hana.di.graphworkspace",
  "plugin_version": "2.0.0.0"
}
```

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.11 Hadoop Map Reduce Jobs (.hdbmrjob or .jar)

Transform a design-time Hadoop map-reduce job resource into a virtual-function-package database object.

### ⚠ Caution

This plug-in for the file types (.hdbmrjob or .jar) has been superseded by the Virtual Package plug-in (.hdbvirtualpackage\*). Use the .hdbvirtualpackage\* plug-ins for design-time Hadoop and SparkSQL JAR files respectively. For more information, see *Related Links*.

The plug-in for virtual-function-package for Hadoop Map Reduce Jobs (hdbmrjob or jar) transforms a design-time Hadoop map reduce job resource into a virtual-function-package database object with adapter type "HADOOP". The plugin supports the following two formats:

- A binary file format which contains a Hadoop JAR file, packaged together with a metadata header
- A plain Hadoop JAR file

## Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plugin configuration should look like the following example:

### Restriction

The `hdbhadoopmrjob` plug-in has been deprecated; use the `hdbvirtualpackagehadoop` plug-in instead. For more information, see *Related Links* below.

### Code Syntax

```
.hdiconfig

"hdbhadoopmrjob" : {
    "plugin_name" : "com.sap.hana.di.virtualfunctionpackage.hadoop",
    "plugin_version": "2.0.0.0"
}

"jar" : {
    "plugin_name" : "com.sap.hana.di.virtualpackage.hadoop",
    "plugin_version": "2.0.0.0"
}
```

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

[Virtual Packages \(.hdbvirtualpackage\) \[page 847\]](#)

## 12.12 Indexes (.hdbindex)

Transform a design-time index resource into an index on a database table.

The index plugin (.hdbindex) transforms a design-time index resource into an index on a database table. The file format uses a DDL-style syntax which is equivalent to the corresponding syntax in the SQL command `CREATE INDEX`, although without the leading “`CREATE`” command.

### Example Artifact Code

The following code shows a simple example of an index definition for XS advanced HDI:

### Code Syntax

```
/src/CUSTOMER_NAME_IDX.hdbindex

INDEX "com.sap.hana.example::CUSTOMER_NAME_IDX"
ON "com.sap.hana.example::CUSTOMERS" (NAME)
```

## Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plugin configuration should look like the following example:

### Code Syntax

```
.hdiconfig

"hdbindex" : {
    "plugin_name" : "com.sap.hana.di.index",
    "plugin_version": "2.0.0.0"
}
```

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.13 Libraries (.hdbllibrary)

Transform a design-time library resource into a library database object.

The library plugin (hdbllibrary) transforms a design-time library resource into a library database object. The file format uses a DDL-style syntax that is equivalent to the corresponding syntax in the SQL command CREATE LIBRARY, although without the leading “CREATE” command.

### **Restriction**

Supported languages are: L.

## Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plugin configuration should look like the following example:

### Code Syntax

```
.hdiconfig

"hdbllibrary" : {
    "plugin_name" : "com.sap.hana.di.library",
    "plugin_version": "2.0.0.0"
}
```

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.14 Logical Schema Definition (.hdblogicalschema)

Transforms a design-time logical-schema definition into a database object.

Synonym and projection-view configurations (as described in the corresponding sections for `.hdbsynonymconfig` and `.hdbprojectionviewconfig` files) provide two ways of defining the schema containing the accessed database object. Either the schema is specified explicitly in the “`schema`” field of the configuration or the configuration contains a “`logical_schema`” field. Specifying a “`logical_schema`” creates a deployment dependency to a logical schema definition that is deployed separately.

Logical schema definitions have the following JSON format.

### Note

A logical schema definition file can contain multiple definitions.

### Code Syntax

```
{  
    "<logical schema 1>" : {  
        "target": {  
            "schema" : "<target schema>" // mandatory, not empty and different  
                                // from the container schema  
        }  
    },  
    "<logical schema 2>" : {  
        "target": {  
            "schema" : "<target schema>" // mandatory, not empty and different  
                                // from the container schema  
        }  
    },  
    "<...>"  
}
```

The “`schema`” field of a target description is mandatory; it cannot be empty and must be different from the container schema.

### Plug-in Configuration

In the configuration file for the HDI container (`.hdiconfig`), the plugin configuration should look like the following example:

### Code Syntax

```
.hdiconfig  
  
    "hdblogicalschema" : {  
        "plugin_name" : "com.sap.hana.di.logicalschemas",
```

```
        "plugin_version": "2.0.0.0"
    }
```

## Related Information

[Projection Views \(.hdbprojectionview\) \[page 802\]](#)  
[Synonyms \(.hdbsynonym and .hdbsynonymconfig\) \[page 818\]](#)

## 12.15 Procedures (.hdbprocedure)

Transform a design-time procedure resource into a procedure database object.

The procedure plugin (`hdbprocedure`) transforms a design-time procedure resource into a procedure database object. The file format uses a DDL-style syntax which is equivalent to the syntax of the corresponding SQL command `CREATE PROCEDURE`, but without the leading “`CREATE`”. Supported languages are: SQLScript, L, and R. AFLLang procedures are handled by the “AFLLang Procedure” plugin.

### i Note

If the procedure is implemented in R, then the container’s object owner (`<container>#00`) needs the privilege `CREATE R SCRIPT`.

### Example Artifact Code

The following code shows a simple example of a procedure definition for XS advanced HDI:

#### Code Syntax

```
/src/SELECT_CUSTOMER.hdbprocedure

PROCEDURE "com.sap.hana.example::SELECT_CUSTOMER" (
    IN ID INTEGER,
    OUT NAME NVARCHAR(1024)
)
LANGUAGE SQLSCRIPT
SQL SECURITY INVOKER
AS
BEGIN
    SELECT NAME INTO NAME FROM "com.sap.hana.example::CUSTOMERS"
    WHERE CUSTID = ID;
END
```

## Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plugin configuration should look like the following example:

### Code Syntax

```
.hdiconfig

"hdbprocedure" : {
    "plugin_name" : "com.sap.hana.di.procedure",
    "plugin_version": "2.0.0.0"
}
```

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.16 Projection Views (.hdbprojectionview)

Transforms a design-time projection-view definition into a database object.

The Projection View plug in transforms a design-time projection view resource into a projection view database object. A projection view configuration resource is required that contains the binding from the projection view to the target database, target schema, and target object; the configuration file is similar to the one required for the configuration of a database synonym.

The complete definition of a projection view is split into the following design-time files:

- `hdbprojectionview`  
A projection view declaration (with an optional default configuration)
- `hdbprojectionviewconfig`  
An explicit configuration of the projection view's target. A projection view configuration file can contain multiple configurations.

### Note

The explicit configuration can be provided at the latest at deployment time and it overrides the optional default configuration. This way, an administrator can map object references according to the deployment context

## Example Artifact Code

The format of the projection view file uses a DDL-style syntax which is equivalent to the syntax of the corresponding SQL command `CREATE PROJECTION VIEW`, without the leading “`CREATE`” command. The

FROM clause of the `PROJECTION VIEW` definition defines the default configuration. The following code shows a simple example of a projection-view definition for XS advanced HDI:

#### Code Syntax

```
/src/customers.hdbprojectionview

PROJECTION VIEW "com.sap.hana.example::CUSTOMERS_VIEW"
AS
SELECT ID, NAME FROM "<database>".<schema>.<object>"
```

To deploy the projection view, you must bind the projection view to an object. To bind the projection view to an object, you must define a projection-view **configuration**, as illustrated in the following example:

#### Code Syntax

```
/src/customers.hdbprojectionviewconfig

{
  "com.sap.hana.example::CUSTOMERS_VIEW" : {
    "target" : {
      "database" : "DATABASE_A",
      "schema"   : "APPLICATION_B", // optional
      "object"   : "CUSTOMERS"
    }
  }
}
```

#### Note

A projection view configuration file can contain multiple configurations.

#### Code Syntax

```
/src/alternate.hdbprojectionviewconfig

{
  "com.sap.hana.example::CUSTOMERS_VIEW" : {
    "target" : {
      "logical_schema" : "ANOTHER_APPLICATION", // not in conjunction with
                                                // database or schema
      "object"       : "CUSTOMERS"
    }
  }
}
```

If the `logical_schema` field of a target description is defined, it is not necessary to provide the fields `database`, `schema`, or `revalidate`. In this case, there is a deployment dependency to a logical schema definition (described in `.hdblogicalschema` files). The logical schema definition contains the schema name that is actually used; it must be different from the name of the container schema.

If the target description does not contain the `logical_schema` field and the `schema` field is omitted, too, then the synonym points to a container-local object. In this case, the referenced object is considered as a deployment dependency unless the `revalidate` field is set to “`false`”. The `revalidate` field is optional and defaults to “`true`” for this case.

If no explicit schema is defined for a target description, the projection view points to a container-local object. In this case, the referenced object is considered as a deployment dependency. If the projection view points to an object inside another schema, to a different container, or to an object inside the same container, which is owned by a different user, then the container's object owner ("<containerName>#OO") must have the required privileges (for example, WITH GRANT OPTION) on the specified target object, for example, SELECT, UPDATE, INSERT.

## Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plugin configuration should look like the following example:

### Code Syntax

```
.hdiconfig

"ldbprojectionview" : {
    "plugin_name" : "com.sap.hana.di.projectionview",
    "plugin_version": "2.0.0.0"
},
"ldbprojectionviewconfig" : {
    "plugin_name" : "com.sap.hana.di.projectionview.config",
    "plugin_version": "2.0.0.0"
}
```

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

[Logical Schema Definition \(.hdblogicalschema\) \[page 800\]](#)

## 12.17 Public Synonym (.hdbpublicsynonym)

Create public synonyms that refer to database objects located in the target schema of the current container.

The Public Synonym plugin can be used to create public synonyms that refer to database objects located in the target schema of the current container. Public synonyms are single database objects; a public synonym with a specific name can only be created from within **one** HDI container.

### i Note

The Public Synonym plugin is intended for use in migration scenarios where an existing application uses public synonyms to access database objects. As a result, the corresponding plugin library (com.sap.hana.di.publicsynonym) is not part of the default libraries for a new HDI container and must be explicitly configured by an administrator.

## Example Artifact Code

### Code Syntax

```
src/a_synonym.hdbpublicsynonym

{
  "<synonym 1>" : {
    "target": {
      "object" : "<the target object 1>"
    }
  },
  "<synonym 2>" : {
    "target": {
      "object" : "<the target object 2>"
    }
  },
  <...>
}
```

A public synonym definition file can contain multiple definitions, just like the normal synonym artifact.

<synonym\_1>, <synonym\_2>, and so on, define the public synonym names to be created; <the target object 1> and <the target object 2> are the corresponding, referenced container-specific run-time objects.

The following example creates a public synonym named com.sap.hana.example::A\_Public\_Synonym in the schema PUBLIC which points to the object com.sap.hana.example::A\_TABLE in the container's schema. The name of the public synonym must follow the normal namespace rules.

### Sample Code

```
{
  "com.sap.hana.example::A_Public_Synonym" : {
    "target": {
      "object" : "com.sap.hana.example::A_TABLE"
    }
  }
}
```

### Note

Users who make use of the \_SYS\_BIC synonym must be assigned to a container-specific role that provides privileges to the addressed target object; the role can be assigned to the user with the container's GRANT\_CONTAINER\_SCHEMA\_ROLES application programming interface (API).

## Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plugin configuration should look like the following example:

### Code Syntax

```
.hdiconfig

"hdbsynonym": {
  "plugin_name" : "com.sap.hana.di.publicsynonym",
```

```
    "plugin_version": "2.0.0.0"
}
```

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.18 Result Cache (.hdbresultcache)

Transform a DDL-based definition of a result cache into the corresponding catalog object.

The Result Cache plugin transforms a design-time DDL-based definition of a result cache into a catalog-level result cache definition. The file format uses a DDL-style syntax which is similar to the corresponding syntax for the [ALTER VIEW | FUNCTION] <object> ADD CACHE statement. The file must use an artificial cache name which uses the prefix \_SYS\_CACHE# followed by the name of the referenced view or function, as illustrated in the following example.

### Example Artifact Code

#### Code Syntax

```
src/a_resultcache.hdbresultcache

RESULT CACHE "_SYS_CACHE#com.sap.hana.example::A_SQL_View"
  ON VIEW "com.sap.hana.example::A_SQL_View"
  WITH RETENTION 30
```

### Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plugin configuration should look like the following example:

#### Code Syntax

```
.hdiconfig

  "hdbresultcache": {
    "plugin_name" : "com.sap.hana.di.resultcache",
    "plugin_version": "2.0.0.0"
  }
```

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.19 Roles (.hdbrole)

Transform a design-time role resource (.hdbrole) into a run-time role object.

The role plug-in transforms a design-time role resource, a file with the .hdbrole extension, into a run-time role. The format of the file must comply with the JSON syntax, and the file content specifies privileges and other roles to be included in the role to be activated.

### Example Artifact Code

The following code shows a simple example of a role definition for XS advanced HDI:

#### Code Syntax

```
/src/myDesignTimeRole.hdbrole

{
  "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" ]
      }
    ]
  }
}
```

```
}
```

The following table lists the properties that can be defined in the role-definition file, and indicates if the property is mandatory.

Table 132: Design-Time Role-Definition Properties

Property	Mandatory	Description
"role"	Yes	The root of the JSON data structure and an object with a single key role
"name"	Yes	The name of the role. The name is used as the name of the run-time role created in the database. Each role definition must have a valid name, which is a non-empty string that contains only permitted characters.
"global_roles"	No	A role can include other global roles; that is, roles created without a schema, for example, "MODELING" or "DATA ADMIN". A valid entry for a global role is any non-empty string
"schema_roles"	No	A role can include other schema-local roles; that is, roles created with a schema. A valid entry for schema-local roles consists of: <ul style="list-style-type: none"><li>• A valid entry for a "schema_reference", which is a unique identifier within the .hbrole file and can be resolved to the name of a real schema by means of the role configuration file (.hbroleconfig).</li><li>• A non-empty list, names, of the relevant roles created in the referenced schema. The schema_reference entry is optional. If omitted, the specified role names will refer to roles created in the schema of the object owner used in the deployment. The resolution of the schema references is based on configuration specified in a corresponding .hbroleconfig file</li></ul> <pre>"schema_roles": [   {     "schema_reference": "RoleSchemaRef1",     "names": ["RoleA", "RoleB"]   } ]</pre>
"system_privileges"	No	A role can include <b>system</b> privileges. The relevant system privileges can be specified using a list. If the list is specified, it must not be empty and must contain valid system privileges
		<pre>"system_privileges": [   "BACKUP ADMIN",   "USER ADMIN" ],</pre>

Property	Mandatory	Description
"schema_privileges"	No	<p>A role can include <b>schema</b> privileges. The relevant schema privileges can be specified using a list. If the list is specified, it must not be empty and must contain valid schema privileges.</p> <pre>"schema_privileges": [   {     "reference": "Ref1",     "privileges": [ "SELECT" ],     "privileges_with_grant_option": [       "UPDATE"     ]   } ],</pre> <p>A valid entry for schema privileges consists of:</p> <ul style="list-style-type: none"> <li>• A valid entry for “reference”, which is a unique identifier within the .hbrole file and can be resolved to a real schema name in the corresponding role-configuration file</li> <li>• A non-empty list, “privileges”, of schema privileges, which are to be provided by the role <b>without</b> grant option</li> <li>• And/or a non-empty list, “privileges <b>with</b> grant options”, of schema privileges, which are to be provided by the role with grant option.</li> </ul> <p><b>i Note</b></p> <p>The “reference” entry is optional. If it is omitted, the specified privileges will refer to the schema of the object owner used in the deployment.</p>

Property	Mandatory	Description
"object_privileges"	No	<p>A role can include privileges on objects activated in the same container. The relevant object privileges can be specified using an "object_privileges" list. If the list of object privileges is specified, it must not be empty and must contain valid entries of object privileges.</p> <pre>"object_privileges": [   {     "name": "Table1",     "type": "TABLE",     "privileges": [ "SELECT" ],     "privileges_with_grant_option":     [ "UPDATE" ]   } ],</pre> <p>A valid entry for object privileges consists of::</p> <ul style="list-style-type: none"> <li>• A valid string for the object "name", or a synonym of the real object</li> <li>• A valid string for "type" of the object, or the "type" of the real object in case of a synonym</li> <li>• A non-empty list ("privileges") of relevant and applicable privileges on this object, which are to be provided by the role <b>without</b> grant option</li> <li>• And/or a non-empty list ("privileges_with_grant_option"), of relevant and applicable privileges, which are to be provided by the role <b>with</b> grant option.</li> </ul> <p>The following object types are supported in the specification of object privileges:</p> <ul style="list-style-type: none"> <li>• Basic SQL types INDEX, FUNCTION, PROCEDURE, SYNONYM, TABLE, TRIGGER, VIEW</li> <li>• Special object types: REMOTE SOURCE, and PSE</li> </ul>

Property	Mandatory	Description
"schema_analytic_privileges"	No	<p>A role can include analytic privileges (also called structured privileges) specified in a list. Only schema-local analytic privileges (created with a schema) are supported. If the list of structured privileges is specified, it must not be empty and must contain valid entries.</p> <pre>"schema_analytic_privileges": [     {         "schema_reference": "APSchemaRef1",         "privileges": ["AP1", "AP2"],         "privileges_with_grant_option": ["AP4", "AP6"]     } ],</pre> <p>A valid entry for schema-local analytic privileges consists of::</p> <ul style="list-style-type: none"> <li>• A valid string for the object “schema_reference” is a unique identifier within the .hdbrole file which can be resolved to a real schema name in the role configuration</li> <li>• A non-empty list (“privileges”) of relevant analytic privileges created in the referenced schema. These privileges are provided by the role <b>without</b> grant option.</li> <li>• And/or a non-empty list (“privileges_with_grant_option”), of relevant analytic privileges, which are to be provided by the role <b>with</b> grant option.</li> </ul> <p><b>i Note</b></p> <p>The schema_reference entry is optional. If omitted, the specified privileges will refer to analytic privileges created in the schema of the object owner used in the deployment. The resolution of any schema reference is based on the configuration specified in a corresponding .hdbroleconfig file.</p>

If required, a role definition can be complemented by a configuration, which must be specified in a role-configuration (.hdbroleconfig) file, whose contents must comply with the JSON format. The role-configuration file is optional and contains the resolution of the schema reference for schema privileges, roles, and analytic privileges contained in the role, as illustrated in the following example:

### Code Syntax

/src/myDesignTimeRole.hdbroleconfig

```
{
  "RoleX": {
    "Ref1": {
      "schema": "Schema1"
    },
    "RoleSchemaRef1": {
      "schema": "Schema2"
    },
    "APSchemaRef1": {
      "schema": "Schema3"
    }
}
```

```
    }
}
```

In the role-configuration file, the root object is identified by the name of the role, “RoleX” in this example. It contains a list of mappings. Each mapping is in turn identified by the reference name, for example, “Ref1”, “Ref2”.

### **Restriction**

You can only specify references of type “schema”.

## Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plug-in configuration should look like the following example:

### **Code Syntax**

```
.hdiconfig

"hdbrole" : {
    "plugin_name" : "com.sap.hana.di.role",
    "plugin_version": "2.0.0.0"
},
"hdbroleconfig" : {
    "plugin_name" : "com.sap.hana.di.role.config",
    "plugin_version": "2.0.0.0"
}
```

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.20 Search Rule Set (.hdbsearchruleset)

Creates search configurations that can be consumed with a built-in database procedure.

The Search Rule Set plugin (`hdbsearchruleset`) creates search configurations that can be consumed with the built-in database procedure `SYS.EXECUTE_SEARCH_RULE_SET`. The format of the design-time artifact that defines the search-rule configuration is based on XML. Search configurations define rules that describe when a row will be returned as a search result.

### **Note**

To use the search configuration, `EXECUTE` privileges on `SYS.EXECUTE_SEARCH_RULE_SET` are required.

## Example Artifact Code

The following code shows a simple example of a sequence definition for XS advanced HDI:

### Code Syntax

```
/SRS.hdbsearchruleset

<?xml version="1.0" encoding="UTF-8"?>
<SearchRuleSet:ruleSet xmlns:SearchRuleSet="http://www.sap.com/nedb/
SearchRuleSet.ecore" scoreSelection="firstRule">
  <attributeView name="p1.p2.p3::aView">
    <keyColumn name="id1"/>
    <keyColumn name="id2"/>
  </attributeView>
  <rule name="Rule 1">
    <column minFuzziness="0.7" name="firstName" weight="0.9">
      <textColumnOptions abbreviationSimilarity="0.9"/>
    </column>
    <column minFuzziness="1.0" name="lastName">
      <textColumnOptions/>
    </column>
    <column minFuzziness="0.8" name="street">
      <stringColumnOptions/>
    </column>
    <column minFuzziness="0.8" name="city">
      <stringColumnOptions/>
    </column>
  </rule>
  <rule name="Rule 1">
    <column minFuzziness="1.0" name="firstName">
      <textColumnOptions/>
    </column>
    <column minFuzziness="1.0" name="lastName">
      <textColumnOptions/>
    </column>
    <column minFuzziness="0.8" name="dateOfBirth">
      <dateColumnOptions maxDateDistance="5"/>
    </column>
  </rule>
</SearchRuleSet:ruleSet>
```

### Tip

For examples showing the content and format of the `.hdbsearchruleset` artifact, see the *SAP HANA Search Developer Guide* in the *Related Links* below.

## Plug-in Configuration

In the configuration file for the HDI container (`.hdiconfig`), the plugin configuration should look like the following example:

### Code Syntax

```
.hdiconfig

"hdbsearchruleset" : {
  "plugin_name" : "com.sap.hana.di.searchruleset",
  "plugin_version": "2.0.0.0"
}
```

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.21 Sequence (.hdbsequence)

Transforms a design-time sequence resource into a sequence database object.

The sequence plugin (`hdbsequence`) transforms a design-time sequence resource into a sequence database object. The file format uses a DDL-style syntax which is equivalent to the syntax of the corresponding SQL command `CREATE SEQUENCE`, but without the leading “`CREATE`”. If the sequence definition contains a `RESET BY` query, then this `RESET BY` query is executed during deployment to set the sequence to its start value. Sequences **without** a `RESET BY` query are also supported.

### Example Artifact Code

The following code shows a simple example of a sequence definition for XS advanced HDI:

#### Code Syntax

```
/src/CUSTOMER_ID.hdbsequence

SEQUENCE "com.sap.hana.example::CUSTOMER_ID"
RESET BY
SELECT IFNULL(MAX(ID), 0) + 1 FROM "com.sap.hana.example::CUSTOMERS"
```

### Plug-in Configuration

In the configuration file for the HDI container (`.hdiconfig`), the plugin configuration should look like the following example:

#### Code Syntax

```
.hdiconfig

"hdbsequence" : {
  "plugin_name" : "com.sap.hana.di.sequence",
  "plugin_version": "2.0.0.0"
}
```

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.22 SQL Views (.hdbview)

Transforms a design-time view resource into an SQL view database object.

The view plugin (`hdbview`) transforms a design-time view resource into an SQL view database object. The file format uses a DDL-style syntax which is equivalent to the syntax of the corresponding SQLScript command `CREATE VIEW`, without the leading “`CREATE`”.

### Example Artifact Code

The following code shows a simple example of an SQL view definition for XS advanced HDI:

#### ⚠ Caution

The use of the asterisk (\*) wildcard inside a view's projection clause will be reported as a warning. This is because the use of the asterisk in the projection causes problems with the column ordering of underlying objects.

#### Code Syntax

```
/src/active_customers.hdbview

VIEW "com.sap.hana.example::ACTIVE_CUSTOMERS"
AS SELECT ID, NAME
FROM "com.sap.hana.example::CUSTOMERS"
WHERE ACTIVE = '1'
```

SQL views also support “forward declarations” for database-level associations, for example, using the “`WITH ASSOCIATIONS`” clause, as illustrated in the following code snippet. The view plugin creates an additional intermediate “validate” artifact as part of the deployment process. This “validate” artifact positions itself between the view artifact, the artifacts which are referenced in the “`WITH ASSOCIATIONS`” clause, and the artifacts which consume the view artifact. The “validate” artifact attempts to validate the forward declared associations and issues a warnings if an association is not valid.

#### Code Syntax

```
/src/view_with_associations.hdbview

VIEW "com.sap.hana.example::CUSTOMERS2"
AS SELECT ID, NAME
FROM "com.sap.hana.example::CUSTOMERS"
WHERE ACTIVE = '1'
WITH ASSOCIATIONS
(
    JOIN OTHER_TABLE
    AS
    TO_OTHER_TABLE
    ON ID = TO_OTHER_TABLE.ID
)
```

## Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plugin configuration should look like the following example:

### Code Syntax

```
.hdiconfig

"hdbview" : {
    "plugin_name" : "com.sap.hana.di.view",
    "plugin_version": "2.0.0.0"
}
```

If the view selects from a synonym which points to an object inside another schema or different container, or to an object inside the same container but which is owned by a different user, then the container's object owner ("<container>#00") must have the required privileges on this target object, for example. SELECT, UPDATE, INSERT. If a view is exposed to other users, the object owner usually needs privileges WITH GRANT OPTION.

With this release, the use of \* inside a view's projection clause will be reported as a warning, because the use of \* is not stable with regard to the column ordering of underlying objects. Views also support forward declarations for database-level associations using the WITH ASSOCIATIONS clause. The view plugin will create an additional intermediate "validate" artifact as part of the deployment. This "validate" artifact will position itself between the view artifact, the artifacts which are referenced in the WITH ASSOCIATIONS clause, and the artifacts which consume the view artifact. The "validate" artifact validates the forward declared associations and will issue a warnings in case an association is not valid.

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.23 Statistics (.hdbstatistics)

Transforms a design-time statistics resource into a statistics object on a database table.

The statistics plugin (hdbstatistics) transforms a design-time statistics resource into a statistics object on a database table. The file format uses a DDL-style syntax which is equivalent to the syntax of the corresponding SQL command CREATE STATISTICS, but without the leading "CREATE".

### ⚠ Restriction

The statistics plugin only supports names statistics.

## Example Artifact Code

The following code shows a simple example of a statistics definition for XS advanced HDI:

### Code Syntax

```
/src/CUSTOMER_NAME_STATISTICS.hdbstatistics  
  
STATISTICS "com.sap.hana.example::CUSTOMER_NAME_STATISTICS"  
ON "com.sap.hana.example::CUSTOMERS" (NAME)
```

## Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plugin configuration should look like the following example:

### Code Syntax

```
.hdiconfig  
  
"hdbstatistics" : {  
    "plugin_name" : "com.sap.hana.di.statistics",  
    "plugin_version": "2.0.0.0"  
}
```

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.24 Structured Privilege (.hdbstructuredprivilege)

Transforms a design-time DDL-based structured privilege resource into a structured privilege object.

The structured privilege plugin (`hdbstructuredprivilege`) transforms a design-time DDL-based structured privilege resource into a structured privilege object in the database. The file format uses a DDL-style syntax which is equivalent to the syntax of the corresponding SQL command `CREATE STRUCTURED PRIVILEGE`, but without the leading “`CREATE`”. The referenced views must be defined using the `WITH STRUCTURED PRIVILEGE CHECK` clause.

## Example Artifact Code

The following code shows a simple example of a structured-privilege definition for XS advanced HDI:

### Code Syntax

```
/src/USER_DATA_VIEW_PRIVILEGE.hdbstructuredprivilege  
  
STRUCTURED PRIVILEGE USER_DATA_VIEW_PRIVILEGE
```

```
FOR SELECT ON USER_DATA_VIEW  
WHERE USER_DATA_VIEW.USER_NAME = CURRENT_USER
```

## Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plugin configuration should look like the following example:

### Code Syntax

```
.hdiconfig  
  
"hdbstructuredprivilege" : {  
    "plugin_name" : "com.sap.hana.di.structuredprivilege",  
    "plugin_version": "2.0.0.0"  
}
```

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.25 Synonyms (.hdbsynonym and .hdbsynonymconfig)

Transforms a design-time synonym definition into a database synonym object.

The complete definition of a synonym is split into the following design-time files:

- **.hdbsynonym**  
A synonym declaration (with an optional default configuration)
- **.hdbsynonymconfig**  
An explicit (but optional) configuration of the synonym's target; the configuration can also be included in the synonym definition (.hdbsynonym).

### Note

The explicit configuration can be provided at the latest at deployment time and it overrides the optional default configuration. This way, an administrator can map object references according to the deployment context.

Consider a procedure with the following SQL query in the body:

```
SELECT * FROM SYS.DUMMY;
```

Since `SYS.DUMMY` is a non-local schema reference (the same applies to `PUBLIC.DUMMY`), the usage is forbidden within the procedure. To perform the `SELECT` on the `SYS.DUMMY` table, the developer must define a synonym, as illustrated in the following examples:

### Example Artifact Code

Synonym definitions and configurations have JSON format, where the synonym definitions use the content of "target" object(s) in the associated synonym configuration as a default configuration. The following code shows a simple example of a synonym definition for XS advanced HDI:

#### Code Syntax

```
/src/synonyms.hdbsynonym

{
  "com.sap.hana.example::DUMMY" : {
    }
}
```

Though the non-local schema references are no longer included, it is still not possible to deploy the procedure since the synonym is not yet bound to any object. To bind the synonym to an object, you must define a synonym configuration, as illustrated in the following example:

#### Note

The following example is not syntactically correct; it provides **all** options and is intended for illustration purposes only.

#### Code Syntax

```
/src/synonyms.hdbsynonymconfig

{
  "com.sap.hana.example::DUMMY" : {
    "target" : {
      "database"   : "DATABASE_A", // optional (cross-database access)
      "schema"     : "SYS",       // optional
      "object"     : "DUMMY",
      "revalidate" : true        // optional (Boolean)
    }
  },
  "com.sap.hana.example::synonym2" : {
    "target": {
      "logical_schema": "<the logical schema>", // not in conjunction with
                                                    // database, schema or
      "revalidate"
      "object" : "<the target object>",
    }
  }
}
```

If the `logical_schema` field of a target description is defined, it is not necessary to provide the fields `database`, `schema`, or `revalidate`. In this case, there is a deployment dependency to a logical schema definition (described in `.hdblogicalschema` files). The logical schema definition contains the schema name that is actually used; it must be different from the name of the container schema.

If the target description does not contain the `logical_schema` field and the `schema` field is omitted, too, then the synonym points to a container-local object. In this case, the referenced object is considered as a

deployment dependency unless the `revalidate` field is set to “`false`”. The `revalidate` field is optional and defaults to “`true`” for this case.

If the `schema` field of a target description is omitted, the synonym points to a container-local object. This means that the referenced object is considered as a deployment dependency unless the `revalidate` field is set to “`false`”. The `revalidate` field is optional and defaults to “`true`” for this case.

If the `schema` field of a target description is defined, then the referenced target object is treated as a run-time object that is not managed by the SAP HANA Deployment Infrastructure (HDI), which also means that it is not considered during dependency management.

### **Restriction**

The `revalidate` field is not allowed if the `schema` field is defined.

If the `database` field of a target description is defined, then the synonym points to an object inside a remote database in a multi-tenant database-container setup.

With these two configuration files, the synonym `com.sap.hana.example::DUMMY` for the table `SYS.DUMMY` is fully defined, and the procedure can be deployed. In this way, you can access non-local schema references and configure the synonyms to match the local deployment database structure without modifying any sources **except** the synonym configurations.

### **Restriction**

If the synonym points to an object inside another schema or a different container or to an object inside the same container which is owned by a different user, then the container’s object owner (`<container>#OO`) needs to be granted the required privileges on this target object, for example, `SELECT`, `UPDATE`, and `INSERT`. If views (that are exposed to other users) or definer-mode procedures are built on top of synonyms, then the object owner needs privileges `WITH GRANT OPTION`.

## **Plug-in Configuration**

In the configuration file for the HDI container (`.hdiconfig`), the plugin configuration should look like the following example:

### **Code Syntax**

```
.hdiconfig

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

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

[Logical Schema Definition \(.hdblogicalschema\) \[page 800\]](#)

## 12.26 \_SYS\_BIC Synonym (.hdbsysbicsynonym)

Create synonyms in the \_SYS\_BIC schema that refer to database objects located in the target schema of the current container.

The \_SYS\_BIC Synonym plug-in can be used to create synonyms in the \_SYS\_BIC schema that refer to database objects located in the target schema of the current container. \_SYS\_BIC synonyms are singleton database objects; a \_SYS\_BIC synonym with a specific name can only be created from within only one HDI container.

### i Note

The \_SYS\_BIC Synonym plug-in is intended for use in migration scenarios where an existing application accesses database objects via \_SYS\_BIC synonyms. Consequently, the corresponding plug-in library (`com.sap.hana.di.sysbicsynonym`) is not part of the default libraries for a new HDI container; `com.sap.hana.di.sysbicsynonym` must be explicitly configured by an administrator.

### Example Artifact Code

#### Code Syntax

```
src/a_sysbic_synonym.hdbsysbicsynonym

{
    "<synonym 1>" : {
        "target": {
            "object" : "<the target object 1>"
        }
    },
    "<synonym 2>" : {
        "target": {
            "object" : "<the target object 2>"
        }
    },
    <...>
}
```

Like the standard synonym artifact, a \_SYS\_BIC synonym definition file can contain multiple definitions.. `<synonym_1>/<synonym_2>/<synonym_#>` define the names of the \_SYS\_BIC synonym to be created; the target object 1/the target object 2/the target object # is the corresponding referenced container-specific run-time object.

The following example creates a public synonym “`com.sap.hana.example::A_SYS_BIC_Synonym`” (in the schema \_SYS\_BIC) which points to the object `com.sap.hana.example::A_CALCVIEW` in the container’s schema. The name of the \_SYS\_BIC synonym must follow the normal name-space rules.

### Sample Code

```
A_SYS_BIC_Synonym

{
    "com.sap.hana.example::A_SYS_BIC_Synonym" : {
        "target": {
            "object" : "com.sap.hana.example::A_CALCVIEW"
        }
    }
}
```

Users using the \_SYS\_BIC synonym must be assigned to a container-specific role that provides privileges to the addressed target object. This role can be assigned to the user via the container's GRANT\_CONTAINER\_SCHEMA\_ROLES API.

### Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plug-in configuration should look like the following example:

### Code Syntax

```
.hdiconfig

"hdbsysbicsynonym": {
    "plugin_name" : "com.sap.hana.di.sysbicsynonym",
    "plugin_version": "2.0.0.0"
}
```

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.27 Tables (.hbtable and .hbdropcreatetable)

Transforms a design-time table resource into a table database object.

The table plug-ins (hbtable and hbdropcreatetable) transforms a design-time table resource into a table database object.

The Table plugin uses a data migration component which transforms an already deployed version of the table into the new structure of the table. This transformation always uses a temporary shadow table into which the existing data is copied to match the new structure. The Drop-Create-Table plugin does not provide any data migration. A changed table definition will drop an already deployed version of the table on deployment and then recreate the table. If the table contains any data, this is not allowed, since the data would be lost.

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

## Note

**Virtual** tables are handled by the `hdbvirtualtable` build plugin; constraints on foreign keys are handled by the `hdbconstraint` build plugin.

## Example Artifact code

The following code shows a simple example of a table definition for XS advanced HDI:

### Code Syntax

```
/src/CUSTOMERS.hdbtable

COLUMN TABLE "com.sap.hana.example::CUSTOMERS" (
    "ID" INTEGER, "NAME" NVARCHAR(256),
    "ACTIVE" TINYINT,
    "COUNTRY" NVARCHAR(256),
    PRIMARY KEY ("ID") )
```

## Example Artifact code with comments

In this release, table and column comments can be defined in-place using the `COMMENT` keyword.

The following code shows an example of a table definition with comments:

### Code Syntax

```
/src/CUSTOMERS.hdbtable

COLUMN TABLE "com.sap.hana.example::CUSTOMERS" (
    "ID" INTEGER          COMMENT 'Customer ID',
    "NAME" NVARCHAR(256)   COMMENT 'Customer Name',
    "ACTIVE" TINYINT        COMMENT 'Currently active?',
    "COUNTRY" NVARCHAR(256) COMMENT 'Customer Country',
    PRIMARY KEY ("ID")
)
COMMENT 'Table with Customer data'
```

## Example Artifact code with associations

In this release, tables support forward declarations for database-level associations using the `WITH ASSOCIATIONS` clause.

The table plugins will create an additional intermediate “validate” artifact as part of the deployment. This “validate” artifact will position itself between the table artifact, the artifacts which are referenced in the `WITH ASSOCIATIONS` clause, and the artifacts which consume the table artifact. The “validate” artifact validates the forward declared associations and issues a warning in case an association is not valid.

The following code shows an example of a table definition with associations:

### Code Syntax

```
/src/CUSTOMERS.hdbtable

COLUMN TABLE "com.sap.hana.example::CUSTOMERS" (
    "ID" INTEGER, "NAME" NVARCHAR(256),
    "ACTIVE" TINYINT,
```

```
"COUNTRY" NVARCHAR(256),  
    PRIMARY KEY ("ID") )  
WITH ASSOCIATIONS  
(  
    JOIN OTHER_TABLE  
    AS  
    TO_OTHER_TABLE  
    ON ID = TO_OTHER_TABLE.ID  
)
```

## Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plugin configuration should look like the following example:

### Code Syntax

```
.hdiconfig  
  
"hdbtable" : {  
    "plugin_name" : "com.sap.hana.di.table",  
    "plugin_version": "2.0.0.0"  
,  
"hdbdropcreatetable" : {  
    "plugin_name" : "com.sap.hana.di.dropcreatetable",  
    "plugin_version": "2.0.0.0"  
}
```

### Restriction

Note the following restrictions:

- Permitted table types are limited to ROW and COLUMN.
- Specifying the table **type** is mandatory.
- A table cannot reference another table. It is not possible to define a table:
  - Using another table (for example, TABLE A LIKE B)
  - Based on a query (for example, TABLE A AS SELECT [...] FROM B)
- The clause USING EXTENDED STORAGE is not supported.
- Text analysis features such as CONFIGURATION and MINING CONFIGURATION properties for a table column are not supported.

### Tip

To use Text Analysis configurations for text columns define a separate fulltext index

- Flexible tables (using the WITH SCHEMA FLEXIBILITY clause) and HISTORY tables are not supported.
- Named constraints are not supported.

## Related Information

[Table Data \(.hdbtabledata\) \[page 825\]](#)

[Table Type \(.hdbtabletype\) \[page 836\]](#)

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.28 Table Data (.hdbtabledata)

Insert data from other files (for example, CSV files) into database tables.

The table data plug-ins (`hdbtabledata` and `csv`) can be used to insert data from other files (for example, CSV files) into database tables which are managed by SAP HANA DI. Since the table data plugin has ownership of the data it manages, any run-time modification is reverted on the next deployment of the corresponding table data artifacts.

### Example Artifact Code

The following code shows a simple (incomplete) example of a table-data design-time definition for XS advanced HDI; the format follows JSON syntax:

#### Code Syntax

```
/src/TABLE.hdbtabledata

{
  "format_version": 1,
  "imports": [
    {
      "column_mappings": {
        "tableCol1": 1,
        "tableCol2": "csvCol4",
        "tableCol3": {
          "type": "constant",
          "value": "Constant"
        },
        "tableCol4": {
          "type": "function",
          "name": "range",
          "parameters": {
            "increment_by": "1",
            "start_with": "1"
          }
        },
        "tableCol5": {
          "type": "function",
          "name": "decodeBase64String",
          "parameters": {
            "column_name": "csvCol2"
          }
        }
      },
      "import_settings": {
        "import_columns": [
          "tableCol1",
          "tableCol2",
          "tableCol3"
        ]
      }
    }
  ]
}
```

```

        ],
        "include_filter" : [
            { "tableCol1" : "de", // ( "de" and "X" )
              "tableCol4" : "X" }
        ],
        "exclude_filter" : [
            { "tableCol1" : "de", // ( "de" and "10" )
              "tableCol3" : "10" }
        ]
    },
    "is_collection_table": true,    //not valid here; for illustration only
    "source_data" : {
        "data_type" : "CSV",
        "file_name" : "com.sap.hana.example.data::data.csv",
        "has_header" : true,
        "no_data_import": false,
        "delete_existing_foreign_data": false,
        "dialect" : "HANA",
        "type_config" : {
            "delimiter" : ","
        }
    },
    "target_table" : "com.sap.hana.example::TABLE"
}
]
}

```

## Imports

The `imports` property defines the following objects:

- `target_table`  
Specifies the name of the database table into which the data should be inserted. The given name must point to a database table which is also managed by SAP HDI, for example, the name cannot point to a database synonym or to a database table in another schema.
- [is\\_collection\\_table \[page 827\]](#)

### ⚠ Restriction

Only for use with imports to JSON collections.

Specifies that the target "table" is a JSON collection in the SAP HANA Documentation Store (DocStore). Restrictions exist for imports to collection tables, for example, the DocStore must be enabled and running.

- [source\\_data \[page 828\]](#)  
Describes the structure of the data file which is used as data source for the import. It contains information about the general type of the data file, its name, and format details.
- [import\\_settings \[page 830\]](#)  
Specifies the target table columns and filters for the data that is inserted into the target table
- [column\\_mappings \[page 832\]](#)  
Connects the target table's columns (specified in "import\_settings" with the data specified in "source\_data"

## is\_collection\_table

You can use the HDI Table Data plug-in (`hdbtabledata`) to import JSON documents stored in a CSV file into a JSON collection created in the SAP HANA Document Store (DocStore).

### ⚠ Caution

Although JSON documents do not have a schema, it is not recommended to use different data types for the same identifier in different JSON documents. This could cause inconsistency in the results returned from queries.

Since the CSV file must only contain **one** column, the mapping should look as follows:

### Sample Code

```
"column_mappings": { "MY_DocStore_COLLECTION": 1 },
```

It is only possible to import JSON data into a "table" of type "COLLECTION". For this reason, in the table-data definition, use the "`is_collection_table`" parameter (default is `false`) to define the target table "type" as illustrated in the following example.

### Sample Code

```
is_collection_table": true,
```

You must also specify the "`data_type`" : (CSV) of the source file itself; the format of the data in the CSV file ("`dialect`" : "HANA\_JSON"); the name of the source CSV file ("`file_name`" : ), and indicate that the CSV file does not contain any header information ("`has_header`" : `false`).

It is only possible to import data into one, single collection from one, single `hdbtabledata` file, and the import operation always imports **all** the content included in the corresponding CSV file. The definition file for your import operation involving JSON data should look something like the following example:

### ⚠ Restriction

It is not allowed to use multiple data sources to fill a collection table, for example, by using the `include_filter` parameter in the "`import_settings`" section of the table-data definition file.

### Sample Code

`hdbtabledata` File for JSON data import

```
{
  "format_version": 1,
  "imports": [
    "column_mappings": {
      "CUSTOMERS": 1
    },
    "is_collection_table": true,
    "source_data": {
      "data_type": "CSV",
      "dialect": "HANA-JSON",
```

```

        "file_name": "sap::myJSONcustomerData.csv",
        "has_header": false
    },
    "target_table": "CUSTOMERS"
}
]
}

```

## source\_data

The data source to be used for the import operation is defined in the `source_data` attribute; it describes the structure of the data file which is used as data source for the import as well as the type of the data file, its file name, and additional format details.

Table 133: Source Data Attributes

Attribute	Description	Mandatory
<code>data_type</code>	<p>The type of the data file. Supported values are: “CSV” and “PROPERTIES”</p> <div style="background-color: #ffffcc; padding: 5px;"> <b>i Note</b>            The data type “PROPERTIES” specifies data stored in a <code>.properties</code> file format. No configuration options are required.         </div>	Yes
<code>file_name</code>	The name of a deployed table data source file; the file has to be deployed via the table data source plugin and follows the normal run-time naming scheme	Yes
<code>has_header</code>	Boolean flag to indicate whether the data file contains a header (default = “false”)	No
<code>no_data_import</code>	Boolean flag to indicate that the data file should not be imported and an undeploy-call does not delete entries according to the key specifications (default = “false”)	No
<code>delete_existing_foreign_data</code>	<p>Boolean flag to indicate that in the deploy phase existing data in the target table will be deleted according to the current key specifications (defined in the <code>include_filters</code>); that is, the keys are used to delete foreign entries and no check is performed for existing matching foreign data.</p> <div style="background-color: #ffffcc; padding: 5px;"> <b>⚠ Caution</b>            If set to “true”, all data in the target table is overwritten even if matches are found in the key reservations. For more information, see <i>Key-Reservation Import Scenario</i> below.         </div>	No
<code>type_config</code>	Object to further configure the data parser to match the data file format	No

Attribute	Description	Mandatory
dialect	<p>Specifies the type of data to import. Possible dialects are: HANA and HANA-JSON. Note that the default configuration can be overwritten by <code>type_config</code>.</p> <p><b>⚠ Restriction</b> HANA-JSON is only for use when importing JSON data into a JSON collection in the SAP HANA Document Store (DocStore).</p>	No

The “`data_type` : “CSV” specifies a source file whose contents are formatted with comma-separated values (CSV). For CSV source files, the settings allowed for the configuration type (`type_config`) attribute are described in the following table:

Table 134: Configuration Type Attributes for the Data Type CSV

CSV Attribute	Description	Default Value
<code>line_terminator</code>	Character to use as line terminator, for example, “\n” or “\r”	auto detect: \n, \r, \r\n
<code>delimiter</code>	Character to use as record delimiter, for example, “,” (comma) or “;” (semi-colon)	,
	<p><b>⚠ Caution</b></p> <p><code>&lt;delimiter&gt;&lt;delimiter&gt;(,.) or (::)</code> is treated as NULL.</p> <p><code>&lt;delimiter&gt;&lt;quote_character&gt;&lt;quote_character&gt;&lt;delimiter&gt;(.,") or (;";)</code> is treated as an empty value.</p> <p><code>&lt;delimiter&gt;&lt;space&gt;&lt;delimiter&gt;(,.) or (::)</code> is treated as a space</p>	
<code>do_quote</code>	Flag to enable quoting	true
<code>quote_character</code>	Character to use as quoting character for records, for example, “” (double quote) for records like “value,1”	”
<code>do_escape</code>	Flag to enable escaping	false
<code>escape_character</code>	Character to use as escaping character, for example, \ (backslash) for escaped records such as “a\`nb” (to escape the “n” character)	\
<code>do_weak_escape</code>	Flag to enable a weak-mode of escaping: only the record delimiter, the quote character, and the escape character are considered to be escaped	true
<code>use_escape_sequences</code>	Flag to enable the standard set of escape characters sequences for example, “\n” “\r” “\t”	true

The `hdbtabledata` file brings the content specified in the corresponding CSV file and takes ownership of the content regarding the key restrictions. For import operations where applications still want to be able to add content manually to the table but also want to ensure that no-one else can interfere with the content, the following scenarios are available:

- Key-Reservation Scenario
- Column-Import Scenario

For more information about importing data using the column-import scenario, see *Column-Import Scenario* in [import\\_settings \[page 830\]](#) below.

### Key-Reservation Import Scenario

The source-data parameters `no_data_import` and `delete_existing_foreign_data` are provided to enable you to implement the so-called Key-Reservation scenario, which enables you to import data manually into the target table and which is split into the following phases:

- The table-building phase:

In this phase, the key reservation is performed by the specifications in the `hdbtabledata` file, but no CSV data is imported into the target table. For this phase, you must set the following parameters:

- `"no_data_import": true`
- `"delete_existing_foreign_data": false`

- The table-population phase:

In this phase, the application adds data to the CVS file it wants to use to populate the target table. From this point, it is not recommended to manually insert content into the target table as the manually inserted data will be deleted on redeployment. For this phase, you must set the following parameters:

- `"no_data_import": false`
- `"delete_existing_foreign_data": true`

## import\_settings

Both the `import_settings` and the `import_columns` attributes are mandatory. `import_settings` specifies the target table columns and defines filters for the data that is inserted into the target table;

### Sample Code

```
"import_settings" : {
  "import_columns" : [
    "tableCol1",
    "tableCol2",
    "tableCol3",
    "tableCol4",
    "tableCol5"
  ],
  "include_filter" : [
    {
      "tableCol1" : "de",    // ( "de" and "X" )
      "tableCol4" : "X"
    },
    // or
    {
      "tableCol1" : "en",    // ( "en" and "X" )
      "tableCol4" : "X"
    }
  ]
},
```

```

    "exclude_filter" : [
      {
        "tableCol1" : "de",    // ( "de" and "10" )
        "tableCol3" : "10"
      }
    ]
}

```

## Column-Import Scenario

The `hdbtabledata` file brings the content specified in the corresponding CSV file and takes ownership of the content regarding the key restrictions. For import operations where applications still want to be able to add content manually to the table but also want to ensure that no-one else can interfere with the content, the following scenarios are available:

- Key-Reservation Scenario
- Column-Import Scenario

The parameters `column_mappings` and `import_columns` are provided to enable you to implement the so-called Import-Column scenario, where importing data using the “key-reservation” scenario is not appropriate. For more information about the key-reservation data-import scenario, see *Key Reservation Scenario* in [source\\_data \[page 828\]](#) above.

In the “import-column” scenario, the target table is extended by adding a column indicating that content is imported (for example, `column_name = IMPORTED (NVARCHAR(1))`). Next, the application can extend its keys, so that the `hdbtabledata` file imports values into the indicated column with “Y”, and the application itself can **add** other values, for example, “`IMPORTED='N'" ("IMPORTED": { "type": "constant", "value": "N" })`”.

In the following example, “`mytable`” has three columns: “`A`”, “`B`”, “`IMPORTED`”

### Sample Code

```

"imports": [
  {
    "target_table": "myTable",
    "source_data": {
      "data_type": "CSV",
      "file_name": "sap.pdms_dss_content::package.csv",
      "has_header": true,
      "type_config": {
        "delimiter": ";"
      }
    },
    "column_mappings": { "IMPORTED": { "type": "constant", "value": "Y" } }
  },
  "include_filter" : {
    "import_settings" : {
      "import_columns": ["A", "B", "IMPORTED"]
    }
  }
},

```

## import\_columns

The `import_columns` list is mandatory; it defines the columns of the target table which are relevant for the data import. More specifically, the `import_columns` attribute enables you to connect the target table's columns (specified in the "import\_settings" with the data specified in the "source\_data" section.

In the example above, the mapping between the target table columns and the source CSV file produces the following import results:

- `"tableCol1" : 1,`  
"tableCol1" in the target table is filled with data taken from the first column of the source CSV file
- `"tableCol2" : "csvCol4",`  
"tableCol2" in the target table is filled with data taken from the column named "csvCol4" in the source CSV file
- `"tableCol3" : { "type" : "constant", "value" : "Constant" }`  
"tableCol3" in the target table is filled with data defined by the value "Constant"

It is also possible to specify filtering rules, for example `include_filter` and `exclude_filter`, which restrict the set of data rows inserted into the target table. If no `include_filter` is defined, then the data imported will collide with other imports into the target table. If both expert and import filters are specified, the exclude filter is evaluated after the include filter. This enables you to filter out values that have a matching include filter.

The "column\_mappings" object in the import description connects the target table's columns specified in the "import\_settings" with the data specified in the "source\_data" section.

### i Note

In one exceptional case it is possible to omit the "column\_mappings" object. In this case, the data file has a header (defined in "source\_data"), the columns in the target table are named equally to the CSV header, and a 1:1 mapping must exist between the CSV columns and the table columns.

In all other cases, it is mandatory to specify column mappings, as illustrated in the following example:

### Sample Code

#### Mapping Columns for Data Import from CSV Files

```
...  
"column_mappings" : {  
    "tableCol1" : 1,  
    "tableCol2" : "csvCol4",  
    "tableCol3" : {  
        "type" : "constant",  
        "value" : "Constant"  
    },  
    "tableCol4" : {  
        "type" : "function",  
        "name" : "range",  
        "parameters" : {  
            "increment_by" : "1",  
            "start_with" : "1"  
        }  
    },  
    "tableCol5" : {  
        "type" : "function",  
        "name" : "decodeBase64",  
    }  
},
```

```

        "parameters" : {
            "column_name" : "csvCol2"
        }
    }
}
...

```

The following tables lists and describes the supported column mappings:

Table 135: Supported Table-Import Column Mappings

Mapping Type	Description	Example
<columnName> - Integer X	Maps the data in column "X" of the data file to the column with name "columnName" of the target table	CSV column 1 is mapped to target table column "tableCol1"
<columnName> - String X	Maps the data in the column identified with header name "X" to the column with <columnName> of the table (requires headers in the data file)	The data in the column specified with header name "csvCol4" is mapped to column "tableCol2"
<columnName> - Function X	Calculates the value to map to the column <columnName> by means of the function "X"	Base16/base64 decodings

The following functions can be used in a column mapping scenario to calculate the value to map to a specified column:

Table 136: Supported Column Mappings Functions

Name	Parameters	Description
constant	value	Maps the column to a constant value
range	start_with increment_by	Uses a constant sequence to fill the column value, starting with the value "start_with"; for every row the value will be increased by "increment_by"
decodeBase16	column_name column_number	Decodes a given value from the data file as a base16 string; the value from the CSV file is determined by column_name or column_number
decodeBase64	char62 (optional) char63 (optional) column_name column_name column_number	Decodes a given value from the data file as a base64 string; the value from the CSV file is determined by column_name or column_number; the optional char62 and char63 parameters can be used to define the encoding characters for the values 62 and 63 (defaults are "+" and "/")
getCurrentSchemaName	-	Returns the name of the container's run-time schema.

Name	Parameters	Description
extractLanguageCodeFromFile-Name	file_name	<p>Extracts the language code from the named file using the parameter <code>file_name</code> (the original file name without suffix and language code). The following file-name formats are supported:</p> <ul style="list-style-type: none"> <li>• <code>&lt;file_name&gt;_&lt;languageCode&gt;</code></li> <li>• <code>&lt;file_name&gt;</code></li> </ul> <p>No specified language code (corresponds to the original language)</p>

## Plug-in Configuration

In the configuration file for the HDI container (`.hdiconfig`), the plugin configuration should look like the following example:

### Code Syntax

```
.hdiconfig

"hdbtabledata" : {
  "plugin_name" : "com.sap.hana.di.tabledata",
  "plugin_version": "2.0.0.0"
},
"csv" : {
  "plugin_name" : "com.sap.hana.di.tabledata.source",
  "plugin_version": "2.0.0.0"
}
```

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.29 Table Data Properties (.properties)

Insert data from files in the “`.properties`” file format into database tables which are managed by SAP HANA HDI.

The Table Data Properties plugin can be used to insert data from files in the “`.properties`” file format into database tables which are managed by SAP HANA HDI, for example, to manage translated texts which must be available at the database layer. The information about the target table, additional constant values, and column mappings are provided via a special “`!tabledata`” header inside the file in order to make the file self-contained and to simplify the translation process for language `.properties` files

The special `extractLanguageCodeFromFile` function can be used to automatically extract language-code information from the file's name. For example, for “`OBJECT_en_US.properties`” the language code “`en_US`” gets extracted. The file encoding is UTF-8, and not ISO-8859-1/Latin-1. The translation process for HDI / XS advanced artifacts can also handle `.properties` files as UTF-8 encoded files. In addition, the translation process copies the `!tabledata` header from the original `.properties` file into the translated `.properties` files.

### **Restriction**

A `.properties` file does not support `\u` escape sequence.

The special `target_table` value “`#BIMC_DESCRIPTIONS`” must be used when translated texts should be deployed for calculation views, because these translations are stored inside the database-internal table `_SYS_BI.BIMC_DESCRIPTIONS`.

### **Note**

The Table Data Properties plugin can also be used to model `.tags` files.

## **Example Artifact Code**

The following code shows a simple example of a properties definition for XS advanced HDI:

### **Code Syntax**

```
/src/OBJECT_en_US.properties

#!tabledata
#{
# "target_table" : "TEXT_TABLE",
# "column_mappings" : {
#   "OBJECT" : { "type" : "constant",
#                 "value" : "name.space::VIEW" },
#   "LANGUAGE" : { "type" : "function",
#                  "name" : "extractLanguageCodeFromFile",
#                  "parameters": { "file_name": "OBJECT"} },
#   "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>
#
# XCOL, 120
KEY_1=First Key
# XCOL, 120
KEY_2=Second Key
# XCOL, 120
PARAMETER_1=Parameter One
# XCOL, 120
CUSTOMERS_HIERARCHY=CUSTOMERS Hierarchy
```

## Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plugin configuration should look like the following example:

### Code Syntax

```
.hdiconfig

"properties" : {
    "plugin_name" : "com.sap.hana.di.tabledata.properties",
    "plugin_version": "2.0.0.0"
},
"tags" : {
    "plugin_name" : "com.sap.hana.di.tabledata.properties",
    "plugin_version": "2.0.0.0"
}
```

## Related Information

[Table Data \(.hbtabledata\) \[page 825\]](#)

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.30 Table Type (.hbtabletype)

Transforms a design-time table type resource into a table type database object.

The table type plugin (hbtabletype) transforms a design-time table type resource into a table type database object, for example, for use by signatures of SQL procedures. The file format uses a DDL-style syntax which is equivalent to the syntax of the corresponding SQL command CREATE TYPE AS TABLE, but without the leading “CREATE”.

### Example Artifact Code

The following code shows a simple example of a table **type** definition for XS advanced HDI:

### Code Syntax

```
/src/TT_PUBLISHERS.hbtabletype

TYPE "com.sap.hana.example::TT_PUBLISHERS" AS TABLE (
    "PUBLISHER" INTEGER,
    "NAME" NVARCHAR(50),
    "PRICE" DECIMAL,
    "COUNT" INTEGER
)
```

## Plug-in Configuration

In the configuration file for the HDI container (`.hdiconfig`), the plugin configuration should look like the following example:

### Code Syntax

```
.hdiconfig

"hdbtablename" : {
  "plugin_name" : "com.sap.hana.di.tablename",
  "plugin_version": "2.0.0.0"
}
```

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.31 Text Analysis Configuration (`.hdbtextconfig`)

The SAP HANA Text Analysis Configuration artifact (`.hdbtextconfig`) enables you to customize the features and options to be used for text analysis and to incorporate custom text-analysis dictionaries and extraction rule-sets.

### ➔ Tip

For examples showing the content and format of the `.hdbtextconfig` artifact, see the *SAP HANA Text Analysis Developer Guide* in the *Related Links* below.

## Plug-in Configuration

In the configuration file for the HDI container (`.hdiconfig`), the plugin configuration should look like the following example:

### Code Syntax

```
.hdiconfig

"hdbtextconfig" : {
  "plugin_name" : "com.sap.hana.di.textconfig",
  "plugin_version": "2.0.0.0"
}
```

## Related Information

[Text Analysis Dictionaries \(.hdbtextdict\) \[page 838\]](#)

[Text Analysis Extraction Rules \(.hdbtextrule\) \[page 838\]](#)

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.32 Text Analysis Dictionaries (.hdbtextdict)

The SAP HANA Text Analysis Dictionaries artifact (.hdbtextdict) enables you to specify custom entity types and entity names for use with text analysis.

### Tip

For examples showing the content and format of the .hdbtextdict artifact, see the *SAP HANA Text Analysis Developer Guide* in the *Related Links* below.

### Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plugin configuration should look like the following example:

#### Code Syntax

```
.hdiconfig

"hdbtextdict" : {
    "plugin_name" : "com.sap.hana.di.textdictionary",
    "plugin_version": "2.0.0.0"
}
```

## Related Information

[Text Analysis Extraction Rules \(.hdbtextrule\) \[page 838\]](#)

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.33 Text Analysis Extraction Rules (.hdbtextrule)

The SAP HANA Text Analysis Extraction Rules artifact (.hdbtextrule) is used to specify rules (patterns) for extracting complex entities and relationships using text analysis.

## ➔ Tip

For examples showing the content and format of the `.hdbtextrule` artifact, see the *SAP HANA Text Analysis Developer Guide* in the *Related Links* below.

## Plug-in Configuration

In the configuration file for the HDI container (`.hdiconfig`), the plugin configuration should look like the following example:

### Code Syntax

```
.hdiconfig

"hdbtextrule" : {
    "plugin_name" : "com.sap.hana.di.textrule",
    "plugin_version": "2.0.0.0"
}
```

## Related Information

[Text Analysis Extraction Rules Includes \(.hdbtextinclude\) \[page 839\]](#)

[Text Analysis Extraction Rules Lexicon \(.hdbtextlexicon\) \[page 840\]](#)

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.34 Text Analysis Extraction Rules Includes (.hdbtextinclude)

Define rule definitions to be used in one or more top-level text analysis extraction rule sets.

The SAP HANA Text Analysis Extraction Rules **Include** artifact (`.hdbtextinclude`) is used to define rule definitions to be used in one or more top-level text analysis extraction rule sets (defined in the `.hdbtextrule` artifact) using the `#include` directive.

## ➔ Tip

For examples showing the content and format of the `.hdbtextinclude` artifact, see the *SAP HANA Text Analysis Developer Guide* in the *Related Links* below.

## Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plugin configuration should look like the following example:

### Code Syntax

```
.hdiconfig

"hbdbtextinclude" : {
    "plugin_name" : "com.sap.hana.di.textrule.include",
    "plugin_version": "2.0.0.0"
}
```

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.35 Text Analysis Extraction Rules Lexicon (.hbdbtextlexicon)

The SAP HANA Text Analysis Extraction Rules **Lexicon** artifact (.hbdbtextlexicon) is used to define lists of words for use in one or more top-level sets of text-analysis rules using the "#lexicon" directive.

### Tip

For examples showing the content and format of the .hbdbtextlexicon artifact, see the *SAP HANA Text Analysis Developer Guide* in the *Related Links* below.

## Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plugin configuration should look like the following example:

### Code Syntax

```
.hdiconfig

"hbdbtextlexicon" : {
    "plugin_name" : "com.sap.hana.di.textrule.lexicon",
    "plugin_version": "2.0.0.0"
}
```

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

[Text Mining Configurations \(.hdbtextminingconfig\) \[page 841\]](#)

## 12.36 Text Mining Configurations (.hdbtextminingconfig)

Customize the features and options used for text mining.

The SAP HANA Text Mining Configuration artifact (.hdbtextminingconfig) is used to customize the features and options used for text mining.

➔ Tip

For examples showing the content and format of the .hdbtextminingconfig artifact, see the *SAP HANA Text Mining Developer Guide* in the *Related Links* below.

### Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plugin configuration should look like the following example:

Code Syntax

```
.hdiconfig

  "hdbtextminingconfig" : {
    "plugin_name" : "com.sap.hana.di.textminingconfig",
    "plugin_version": "2.0.0.0"
  }
```

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.37 Triggers (.hdbtrigger)

Transforms a design-time trigger resource into a trigger on a database table.

The trigger plugin (hdbtrigger) transforms a design-time trigger resource into a trigger on a database table. The file format uses a DDL-style syntax which is equivalent to the syntax of the corresponding SQL command CREATE TRIGGER, but without the leading CREATE.

## Example Artifact Code

The following code shows a simple example of a trigger definition for XS advanced HDI:

### Code Syntax

```
/src/TRIGGER_A.hdbtrigger

TRIGGER "com.sap.hana.example::TRIGGER_A"
AFTER DELETE ON "com.sap.hana.example::CUSTOMERS"
FOR EACH STATEMENT
BEGIN
    DECLARE THE_COUNT INT;
    SELECT COUNT(*) INTO THE_COUNT FROM "com.sap.hana.example::CUSTOMERS";
    INSERT INTO "com.sap.hana.example::CUSTOMERS_COUNT" VALUES (THE_COUNT);
END
```

## Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plugin configuration should look like the following example:

### Code Syntax

```
.hdiconfig

"hdbtrigger" : {
    "plugin_name" : "com.sap.hana.di.trigger",
    "plugin_version": "2.0.0.0"
}
```

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.38 Virtual Functions (.hdbvirtualfunction)

Transform a design-time virtual function resource into a virtual function database object.

The virtual function plug-ins (`hdbvirtualfunction` and `hdbvirtualfunctionconfig`) transform a design-time virtual function resource into a virtual function database object. The target database where the virtual function points to needs to be available via a database remote source.

The file format uses a DDL-style syntax which is equivalent to the syntax of the corresponding SQL command `CREATE VIRTUAL FUNCTION`, without the leading “`CREATE`”. The `AT` part of the `VIRTUAL FUNCTION` definition defines the default configuration for the remote source. The container’s object owner (`<container>#OO`) must have the privilege `CREATE VIRTUAL FUNCTION ON REMOTE SOURCE` on the remote source.

Since, in most cases, the remote source is not known during development but depends on deployment decisions, the complete definition of a virtual function is split into two design-time files: a virtual function file (with a default configuration) and an explicit virtual function configuration that contains the binding from virtual function to remote source. The explicit configuration can be provided, at the latest, at deployment time, overriding the optional default configuration. In this way, an administrator can map object references according to the deployment context.

### Example Artifact Code

The following code shows a simple example of a virtual function definition for XS advanced HDI:

#### Code Syntax

```
/src/remote_function.hdbvirtualfunction

VIRTUAL FUNCTION "com.sap.hana.example::REMOTE_FUNCTION"()
RETURNS TABLE ( WORD NVARCHAR(60), COUNT INTEGER)
PACKAGE "com.sap.hana.example::WORD_COUNT"
CONFIGURATION '{}'
AT REMOTE_SOURCE
```

#### Code Syntax

```
/src/remote_function.hdbvirtualfunctionconfig

{
    "com.sap.hana.example::REMOTE_FUNCTION" : {
        "target" : {
            "remote" : "REMOTE_SYSTEM_A"
        }
    }
}
```

### Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plugin configuration should look like the following example:

#### Code Syntax

```
.hdiconfig

"hdbvirtualfunction" : {
    "plugin_name" : "com.sap.hana.di.virtualfunction",
    "plugin_version": "2.0.0.0"
}
"hdbvirtualfunctionconfig" : {
    "plugin_name" : "com.sap.hana.di.virtualfunction.config",
    "plugin_version": "2.0.0.0"
}
```

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.39 Virtual Procedures (.hdbvirtualprocedure)

Transform a design-time virtual procedure resource into a virtual procedure database object.

The Virtual Procedure plugin transforms a design-time virtual procedure resource into a virtual procedure database object. The target database where the virtual procedure points to needs to be available via a database remote source.

Since, in most cases, the remote source is not known during development but depends on deployment decisions, the complete definition of a virtual procedure is split into two design-time files: a virtual procedure file (with an optional default configuration) and an explicit virtual procedure configuration that contains the binding from virtual procedure to remote source. The explicit configuration can be provided at latest at deployment time, overriding the optional default configuration. In this way, an administrator can map object references according to the deployment context.

### Example Artifact Code

The file format uses a DDL-style syntax which is equivalent to the corresponding CREATE VIRTUAL PROCEDURE SQL syntax, **without** the leading CREATE. The AT part of the VIRTUAL PROCEDURE definition defines the default configuration for the remote source, as illustrated in the following example:

#### Code Syntax

```
/src/remote_procedure.hdbvirtualprocedure

VIRTUAL PROCEDURE "com.sap.hana.example::REMOTE_PROCEDURE"
(
    IN INT PARAM AS_IN INT,
    OUT OUT_TABLE TABLE( INT_COLUMN INT, NVARCHAR_COLUMN NVARCHAR(2000))
)
CONFIGURATION '{}'
AT REMOTE_SOURCE
```

#### Note

The container's object owner (<container>#00) must have the privilege CREATE VIRTUAL PROCEDURE ON REMOTE SOURCE on the remote source.

#### Code Syntax

```
/cfg/remote_procedure.hdbvirtualprocedureconfig

{
    "com.sap.hana.example::REMOTE_PROCEDURE" : {
        "target" : {
            "remote" : "REMOTE_SYSTEM_A"
        }
    }
}
```

```
}
```

## Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plugin configuration for the virtual procedure should look like the following example:

### Code Syntax

```
.hdiconfig

"hd़virtualprocedure" : {
    "plugin_name" : "com.sap.hana.di.virtualprocedure",
    "plugin_version": "2.0.0.0"
},
"hd़projectionviewconfig" : {
    "plugin_name" : "com.sap.hana.di.virtualprocedure.config",
    "plugin_version": "2.0.0.0"
}
```

## 12.40 Virtual Tables (.hd़virtualtable)

Transform a design-time virtual table resource into a virtual table database object.

The virtual table plug-ins (hd़virtualtable and hd़virtualtableconfig) transform a design-time virtual table resource into a virtual table database object. The target database to which the virtual table points must be available by means of a database remote source. The file format uses a DDL-style syntax which is equivalent to the corresponding syntax in the SQL command CREATE VIRTUAL TABLE, although without the leading “CREATE” command.

In most cases the remote source is not known during development but depends on deployment decisions. Consequently, the complete definition of a virtual table is split into two design-time files: a virtual table file (with a default configuration) and an explicit virtual table configuration that contains the binding from virtual table to remote source, target database, target schema, and target object. The explicit configuration can be provided at the latest at deployment time, overriding the optional default configuration. In this way, an administrator can map object references according to the deployment context.

### i Note

The container's object owner (“<container>#OO”) must have the “CREATE VIRTUAL TABLE” privilege on the remote source, for example: “CREATE VIRTUAL TABLE ON REMOTE SOURCE”.

## Example Artifact Code

The following code shows a simple example of a virtual table definition for XS advanced HDI:

### Code Syntax

```
/src/remote_customers.hdbvirtualtable

VIRTUAL TABLE "com.sap.hana.example::REMOTE_CUSTOMERS"
AT REMOTE.CUSTOMERS
```

### Code Syntax

```
/cfg/remote_customers.hdbvirtualtableconfig

{
  "com.sap.hana.example::REMOTE_CUSTOMERS" : {
    "target" : {
      "remote"   : "REMOTE_SYSTEM_A",
      "database" : "DATABASE_B",
      "schema"   : "APPLICATION_C",
      "object"   : "CUSTOMERS"
    }
  }
}
```

## Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plugin configuration should look like the following example:

### Code Syntax

```
.hdiconfig

"hdbvirtualtable" : {
  "plugin_name" : "com.sap.hana.di.virtualtable",
  "plugin_version": "2.0.0.0"
},
"hdbvirtualtableconfig" : {
  "plugin_name" : "com.sap.hana.di.virtualtable.config",
  "plugin_version": "2.0.0.0"
}
```

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

## 12.41 Virtual Packages (.hdbvirtualpackage )

Transform a design-time Hadoop Map Reduce Job or SparkSQL resource into a virtual package database object.

The Virtual Package plug-in transforms a design-time Hadoop Map Reduce Job or a SparkSQL resource into a virtual package database object with adapter type “hadoop” or “sparksql” respectively. The Virtual Package plug-in supports the following two formats:

- Plain Hadoop JAR files  
Plug-in: hdbvirtualpackagehadoop
- Plain SparkSQL files  
Plug-in: hdbvirtualpackagesparksql

### Plug-in Configuration

In the configuration file for the HDI container (.hdiconfig), the plugin configuration for virtual package resources should look like the following example:

#### Code Syntax

```
.hdiconfig

"hdbvirtualpackagehadoop" : {
    "plugin_name" : "com.sap.hana.di.virtualpackage.hadoop",
    "plugin_version": "2.0.0.0"
},
"hdbvirtualpackagesparksql" : {
    "plugin_name" : "com.sap.hana.di.virtualpackage.sparksql",
    "plugin_version": "2.0.0.0"
}
```

### Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

# 13 The XS Command-Line Interface Reference

A list of all the categories and areas covered by the `xs` command-line interface (CLI).

The XS CLI enables you to maintain not only the applications that are deployed to the XS advanced run-time environment, but also the run-time environment itself, and the users who access and use it. The XS advanced command-line client is included not only in the full XS advanced run-time instance installed on the server but in a separate "client" package for installation on a remote machine.

## ➔ Tip

The XS advanced command-line client is available for download on SAP Service Marketplace at the following location for those people with the required S-User ID:

- SAP HANA PLATFORM EDIT. 1.0  
► [SUPPORT PACKAGES AND PATCHES](#) ► [ENTRY BY COMPONENT](#) ► [XS ADVANCED RUNTIME](#) ► [XS RUNTIME 1](#) ▶
- SAP HANA PLATFORM EDITION 2.0  
► [SUPPORT PACKAGES AND PATCHES](#) ► [XS RUNTIME 1](#) ▶

## Usage

```
xs <command> [<ARGUMENTS>] [<OPTIONS>]
```

To display information about a specific `xs` command:

```
xs help <command>
```

To display information about all `xs` commands:

```
xs help <-a>
```

Table 137: XS Command Overview

Command Category	Description
<a href="#">XS CLI: Logon and Setup [page 850]</a>	User logon, viewing user-organization (and space) targets, and setting API URLs
<a href="#">XS CLI: Application Management [page 852]</a>	Maintain SAP HANA XS applications: listing, deploying, starting, stopping, staging, [...]

Command Category	Description
<a href="#">XS CLI: Services Management [page 874]</a>	Maintain SAP HANA XS services: listing, creating, deleting, binding, updating, [...]
<a href="#">XS CLI: Organizations [page 890]</a>	Maintain user organizations: create, list, rename, delete, [...]
<a href="#">XS CLI: Spaces [page 892]</a>	Manage user spaces: create, list, rename, delete, [...]
<a href="#">XS CLI: Domains [page 896]</a>	Manage domains
<a href="#">XS CLI: Certificates [page 899]</a>	Manage certificates
<a href="#">XS CLI: Routes [page 901]</a>	Maintain application routes: create, list, map, unmap, delete, [...]
<a href="#">XS CLI: Buildpacks [page 904]</a>	Maintain application build-packs: create, list, update, rename, delete, [...]
<a href="#">XS CLI: Run-Time Environments [page 908]</a>	Maintain XS run-times: create, list, display information, search, update, delete, [...]
<a href="#">XS CLI: Tasks [page 915]</a>	Maintain XS application-related tasks
<a href="#">XS CLI: User Administration [page 917]</a>	Maintain SAP HANA users: create, list, purge, delete, set/unset organizations and spaces, [...]
<a href="#">XS CLI: Administration [page 929]</a>	Maintain XS application traces and backups
<a href="#">XS CLI: Configuration [page 931]</a>	Set and maintain environment variables
<a href="#">XS CLI: Blob Store [page 934]</a>	Manage and maintain the contents of the blob store
<a href="#">XS CLI: Advanced [page 936]</a>	Perform advanced administration tasks in SAP HANA XS advanced model.
<a href="#">XS CLI: Other Commands [page 937]</a>	Display information about the SAP HANA XS version, CLI, and system
<a href="#">XS CLI: Plug-ins [page 939]</a>	Additional commands as plug-ins: install product archives and deploy multi-target applications (MTA)

## 13.1 XS CLI: Logon and Setup

Commands to maintain user logon, view user-organization (and space) targets, and set API URLs.

Table 138: XS Logon and Setup Commands Overview

Command	Alias	Description
<code>login</code>	<code>l</code>	Log user in to SAP HANA XS advanced
<code>logout</code>		Log user out of SAP HANA XS advanced
<code>target</code>	<code>t</code>	Set or view the target user organization or space
<code>api</code>		Set or view the URL for the target application programming interface

### login

Log user on to SAP HANA XS advanced model.

#### Usage

```
xs login [-a <API_URL>] [-u <USERNAME>] [-p <PASSWORD>] [-o <ORG>] [-s <SPACE>]
```

#### Options

Table 139: Command Options Overview

Option	Description
<code>-a &lt;API_URL&gt;</code>	API endpoint to connect to (for example, <code>https://api.acme.com</code> )
<code>-u &lt;USERNAME&gt;</code>	Name of the user to log on
<code>-p &lt;PASSWORD&gt;</code>	Password for user specified in <code>&lt;USERNAME&gt;</code> .
<code>--stdin</code>	Get the password from <code>stdin</code> pipe
<code>-o &lt;ORGANIZATION&gt;</code>	Name of the organization to which <code>&lt;USERNAME&gt;</code> belongs
<code>-s &lt;SPACE&gt;</code>	Name of the organizational space to which <code>&lt;USERNAME&gt;</code> belongs
<code>--sso</code>	Provide a URL to obtain a one-time password to log <code>&lt;USERNAME&gt;</code> on
<code>--skip-ssl-validation</code>	Skip SSL validation during the login operation

#### ⚠ Caution

It is strongly recommended not to use this option in a productive environment.

Option	Description
--cacert	The path to the SSL certificate file
--timeout <>	Optional timeout in seconds for failed logon attempts, for example, due to a failed connection

## Examples

```
xs login -u name@example.com -p "my password"
```

### ➔ Tip

If the password contains a space, the password must be enclosed in quotation marks ("") as illustrated in the example above.

## logout

Log user out of SAP HANA XS advanced model.

### Usage

```
xs logout
```

## target

Set or view the target user organization or space.

### Usage

```
xs target [-o <ORG>] [-s <SPACE>]
```

### Options

Table 140: XS CLI: Target Command Options

Option	Description
-o <ORGANIZATION>	Name of the target user organization to log the user on to
-s <SPACE>	Name of the target user space to log the user on to

## api

Set or view the URL that defines the target endpoint for a application programming interface.

## Usage

```
xs api [<API_URL>]
```

## Arguments

Table 141: XS CLI: API Command Arguments

Argument	Description
<API_URL>	API endpoint (for example, <code>https://api.acme.com</code> )

## Options

Table 142: Command Options Overview

Option	Description
--skip-ssl-validation	Skip SSL validation during logon operation   <b>Caution</b> It is strongly recommended not to use this option in a productive environment.
--cacert	The path to the SSL certificate file

## Related Information

[XS CLI: Logon and Setup \[page 850\]](#)

[The XS Command-Line Interface Reference \[page 848\]](#)

## 13.2 XS CLI: Application Management

Commands to maintain SAP HANA XS applications, for example: list, deploy, start, stop, stage.

Table 143: XS CLI: Application Management

Command	Alias	Description
<code>apps</code>	a	List all applications available in the target user space
<code>app</code>		Display the status of an application and any additional information
<code>push</code>	p	Deploy a new application or synchronize changes to an existing application

Command	Alias	Description
<code>scale</code>	s	Change or view the instance count of an application, disk-space limit, and memory limit
<code>delete</code>	d	Delete an application
<code>delete-app-instances</code>		Delete one or more instances of an application
<code>wait-for-apps</code>		Wait until application instances are running
<code>rename</code>		Rename an application
<code>start</code>	st	Start an application
<code>stop</code>	sp	Stop an application
<code>restart</code>	rs	Restart an application
<code>restage</code>	rg	Restage an application
<code>events</code>		Show all recent application-related events
<code>files</code>	f	Print a list of files in a directory or the contents of a file
<code>logs</code>		Tail an application log file or list any recent application logs
<code>set-logging-level</code>	sll	Set the logging level for an application
<code>unset-logging-level</code>	ull	Reset the logging level for the given component to its default
<code>list-logging-levels</code>	lll	List the logging levels that have been manually configured for an application
<code>env</code>	e	Show all environment variables set for an application
<code>set-env</code>	se	Set an environment variable for an application
<code>unset-env</code>		Remove an environment variable for an application
<code>pinned-hosts</code>		List current host pinnings for all applications in the target space
<code>pin-hosts</code>		Pin an application to one or more execution-agent hosts
<code>enable-debugging</code>		Switch on debugging mode for an XS advanced application

Command	Alias	Description
<code>disable-debugging</code>		Switch off debugging mode for an XS advanced application
<code>debugging-info</code>		Display details of the set debugging mode for an XS advanced application
<code>java</code>		Obtain a heap-dump or thread-dump from a running Java application
<code>nodejs</code>		Obtain a heap-dump or thread-dump from a running Node.js application

## apps

List all applications available in the target user space.

### Usage

```
xs apps
```

#### ➔ Tip

You can use the alias `a` in place of the `apps` command.

## app

Display the status of an application and any additional information.

### Usage

```
xs app <APPNAME>
```

### Arguments

Table 144: Command Arguments Overview

Argument	Description
<code>APPNAME</code>	The name of the application whose details you want to display

## Options

Table 145: Command Options Overview

Option	Description
--urls	Show only the URL for the specified application; all other output for the application is suppressed.
--guid	Retrieve and display the given application's Globally Unique Identifier (GUID); all other output for the application is suppressed.

## push

Deploy a new application or synchronize changes to an existing application.

### Usage

Deploy a single application with (or without) a deployment manifest:

```
xs push <APP> [-b <BUILDPACK_NAME>] [-c <COMMAND>] [-d <DOMAIN>]
[-f <MANIFEST_PATH>] [-i <NUM_INSTANCES>] [-k <DISK>] [-m <MEMORY>]
[-n <HOST>] [-p <PATH>] [-s <STACK>] [-t <TIMEOUT>]
[--no-hostname] [--no-manifest] [--no-route] [--no-start]
```

Deploy a multi-target application using the deployment configuration defined in a deployment manifest:

```
xs push [-f <MANIFEST_PATH>]
```

### Tip

You can use the alias `p` in place of the `push` command.

## Arguments

Table 146: Command Arguments Overview

Argument	Description
APP	The name of the application to deploy

## Options

Table 147: Command Options Overview

Option	Description
-b <BUILDPACK_NAME>	The name of a custom build-pack (for example, "my-buildpack") or the URL to a GIT resource
-c <COMMAND>	Startup command; set to null to reset to default start command

Option	Description
-d <DOMAIN>	The name of the target deployment domain (for example, acme.com)
-f <MANIFEST_PATH>	The absolute path to the application's deployment manifest.
-i <NUM_INSTANCES>	Number of instances
-k <DISK>	Disk space limit used for the deployment operation (for example, 256M, 1024M, 1G)
-m <MEMORY>	Memory limit for the deployment operation (for example, 256M, 1024M, 1G)
-n <HOST>	Name of the target host for the deployment operation (for example, "my-subdomain")
-p <PATH>	Path of application directory or Zip file containing the deployable archive
-s <STACK>	Stack to use for the deployment <div style="background-color: #fdf5e6; padding: 10px;"> <b>➔ Tip</b>            A stack is a pre-built file system, including an operating system, that can run applications.         </div>
-t <TIMEOUT>	Start timeout in seconds
--port <PORT>	Port number to use for port-based (rather than domain-based) routing
--no-hostname	Map the root domain to this application
--no-manifest	Ignore the contents of the deployment manifest file
--no-route	Do not map a route to this application
--no-start	Do not start an application after deployment
--random-route	Create a random route for this application
--wait-indefinitely	When starting the application after deployment, wait until all instances have a <b>finished</b> state

## scale

Change or view the application-instance count, as well as its disk-space and memory limit.

### Usage

```
xs scale APP [-i <INSTANCES>] [-k <DISK>] [-m <MEMORY>] [-f] [-w]
```

## Arguments

Table 148: Command Arguments Overview

Argument	Description
APP	The name of the application to scale

## Options

Table 149: Command Options Overview

Option	Description
-i <NUM_INSTANCES>	Number of instances
-k <DISK>	Disk space limit used for the scaling operation (for example, 256M, 1024M, 1G)
-m <MEMORY>	Memory limit for the scaling operation (for example, 256M, 1024M, 1G)
-f	Force restart of scaled application without prompt
-w	Wait for asynchronous jobs to finish

## delete

Delete an application.

### Usage

```
xs delete <APP> [-f -r]
```

#### Tip

You can use the alias `d` in place of the `delete` command.

## Arguments

Table 150: Command Arguments Overview

Argument	Description
APP	The name of the application to delete

## Options

Table 151: Command Options Overview

Option	Description
-f	Force delete of application without prompt

Option	Description
-r	Delete any mapped routes

## delete-app-instances

Delete instances of an application.

### Usage

Delete **stopped** or **crashed** instances of an application:

```
xs delete-app-instances <APP> [--stopped] [--crashed]
```

Delete specific instances of an application:

```
xs delete-app-instances <APP> -d <DROPLET> -i <INSTANCE>
```

### Arguments

Table 152: Command Arguments Overview

Argument	Description
APP	The name of the application whose instances you want to delete

### Options

Table 153: Command Options Overview

Option	Description
--stopped	Delete all instances of the specified application, which have the status <b>stopped</b>
--crashed	
--droplet-index <DROPLET>	Specify the droplet instance you want to delete. To display a list of all available droplets, use the command <code>xs app &lt;myApp&gt;</code> .
-d <DROPLET>	
-f	Force delete of the specified application instance without asking for confirmation
--instance-index <INSTANCE>	Specify the application instance you want to delete. To display a list of all available application instances, use the command <code>xs app &lt;myApp&gt;</code> .
-i <INSTANCE>	

## Examples

To display a list of application instances, along with a corresponding (droplet- or instance-) index number, use the command `xs app <myApp>`, as illustrated in the following example:

### Output Code

```
xs app di-core
Showing status and information about "di-core"
  name:          di-core
  requested state:  STARTED
  instances:      1
  memory:        512 MB
  disk:          <not specified>
  buildpack:     <default>
  urls:          https://host.acme.com:53030

Instances of droplet 1 created at May 24, 2016 10:41:30 AM
index   created           state    host      internal port  os
user

-----
2       May 30, 2016 8:07:47 AM  STOPPED  host.acme.com  50016
sapk35xsa
3       May 31, 2016 2:40:37 PM  CRASHED  host.acme.com  50028
sapk35xsa
4       May 31, 2016 4:23:04 PM  STOPPED  host.acme.com  50012
sapk35xsa

Instances of droplet 3 created at May 31, 2016 4:23:28 PM
index   created           state    host      internal port  os
user

-----
0       May 31, 2016 4:41:42 PM  RUNNING  host.acme.com  50022
sapk35xsa
```

## wait-for-apps

Wait until app instances are running.

### Usage

Wait until apps are accessible; that is, at least one instance of the application running. To wait until **all** application instances are running, add the `--all-instances` option.

```
xs wait-for-apps [--apps<APPS>] [--space <SPACE>] [--all-instances] [--timeout <SECONDS>]
```

### Options

Table 154: Command Options Overview

Option	Description
<code>--apps&lt;APPS&gt;</code>	The names of the applications to wait for, given as comma-separated list

Option	Description
--space <SPACE>	Wait for all applications running in the the specified space in the current target organization
--all-instances	Wait until <b>all</b> instances are running (rather than just at least one application instance)
--timeout <SECONDS>	( <b>optional</b> ) the maximum amount of time in seconds to wait until the application instances are running

## rename

Rename an application.

### Usage

```
xs rename <APP> <NEW_NAME>
```

#### ➔ Tip

You can use the alias `d` in place of the `delete` command.

### Arguments

Table 155: Command Arguments Overview

Argument	Description
APP	The current name of the application whose name you want to change
NEW_NAME	The new name of the application

## start

Start an application.

### Usage

```
xs start <APP>
```

#### ➔ Tip

You can use the alias `st` in place of the `start` command.

## Arguments

Table 156: Command Arguments Overview

Argument	Description
APP	The name of the application which you want to start

## Options

Table 157: Command Options Overview

Option	Description
--wait-indefinitely	When starting an application, wait until all instances have a <b>finished</b> state.

## stop

Stop an application.

### Usage

```
xs stop <APP> [OPTIONS]
```

```
xs stop -s <SPACE>
```

#### Tip

You can use the alias `sp` in place of the `stop` command.

## Arguments

Table 158: Command Arguments Overview

Argument	Description
<APP>	The name of the application which you want to stop

## Options

Table 159: Command Options Overview

Option	Description
-s <SPACE>	Stop <b>all</b> application that are running in the specified space.
--wait-indefinitely	When stopping an application, wait until all instances have a <b>finished</b> state.

## restart

Restart an application.

### Usage

```
xs restart <APP>
```

#### ➔ Tip

You can use the alias `rs` in place of the `restart` command.

### Arguments

Table 160: Command Arguments Overview

Argument	Description
APP	The name of the application which you want to restart

### Options

Table 161: Command Options Overview

Option	Description
--wait-indefinitely	When restarting an application, wait until all instances have a <b>finished</b> state.

## restage

Restage an application.

### Usage

```
xs restage <APP>
```

#### ➔ Tip

You can use the alias `rg` in place of the `restage` command.

### Arguments

Table 162: Command Arguments Overview

Argument	Description
APP	The name of the application which you want to restage

## events

Show all recent application-related events.

### Usage

```
xs events <APP>
```

### Arguments

Table 163: Command Arguments Overview

Argument	Description
APP	The name of the application whose event-details you want to display

### Options

Table 164: Command Options Overview

Option	Description
--all	Show events for all users, not just the current user

## files

Print a list of files in a directory or the contents of a file.

### Usage

List, show, or download files of an application instance:

```
xs files <APP> [<PATH>] [-d <DIRECTORY>] [-i <INSTANCE>] [--droplet-index <DROPLET>] [-r]
```

List, show, or download the log files of an application instance:

```
xs files <APP> --logs [<PATH>] [-d <DIRECTORY>] [-i <INSTANCE>] [--droplet-index <DROPLET>] [-r]
```

List, show, or download the application droplet files:

```
xs files <APP> --droplet-files [<PATH>] [-d <DIRECTORY>] [--droplet-index <DROPLET>] [-r]
```

### Tip

You can use the alias `f` in place of the `files` command.

## Arguments

Table 165: Command Arguments Overview

Argument	Description
<APP>	The name of the application whose file-details you want to display
<PATH>	The path to show information about

## Options

Table 166: Command Options Overview

Option	Description
-i <INSTANCE>	Show files for this instance; the default setting is the first running instance. To list the instances available, use the xs app <APP>
-r	Show files recursively
-d <DIRECTORY>	Download the listed files to the specified directory instead of just showing them (implies -r)
--droplet-index	Show files for this droplet. To list the droplets available, use the xs app <APP>. Default is the latest droplet.
--droplet-files	Show the files of the droplet instead of an instance.
--logs	Show the log files of an instance

## logs

Tail an application log file or list any recent application logs.

## Usage

```
xs logs <APP>
```

## Arguments

Table 167: Command Arguments Overview

Argument	Description
<APP>	The name of the application whose log-file details you want to display

## Options

Table 168: Command Options Overview

Option	Description
--recent <INSTANCE>	Dump recent logs instead of tailing
--all	Dump complete logs instead of tailing
--last <NUM>	Show the last NUM lines of log for this instance.
--source <SOURCES>	Filter the output by the log sources which must be provided as a comma-separated list (for example: API,APP,STG,RTR).
--type <TYPES>	Filter the output by the log types, which must be provided as a comma-separated list (for example: OUT,ERR,TRC,LOG,SYS,ACC).
--instance <INSTANCES>	Filter the output by the instance indices specified as a comma-separated list (for example, 0,1).
--droplet <DROPLET>	Show the instances associated with the specified droplet. If not set, the latest droplet is used.

## set-logging-level

Set the logging level for an application.

### Usage

```
xs set-logging-level <APP> <COMPONENT> <LEVEL>
```

#### ➔ Tip

You can use the alias `sll` in place of the `set-logging-level` command.

### Arguments

Table 169: Command Arguments Overview

Argument	Description
<APP>	The name of the application whose logging level you want to set
<COMPONENT>	The name of the application component to log

Argument	Description
<LEVEL>	<p>The logging level to set, for example: "INFO", "WARNING", "ERROR", "DEBUG", ...</p> <p><b>⚠ Caution</b></p> <p>Logging levels are application-specific and case sensitive; they can be lower-case characters (for example, "debug") or upper-case (for example, "DEBUG"). An error occurs, if you set a logging level incorrectly, for example, using "debug", where the application defines the logging level "DEBUG".</p>

## unset-logging-level

Reset the logging level for the given component to its default.

### Usage

```
xs unset-logging-level <APP> <COMPONENT>
```

#### ➔ Tip

You can use the alias `ull` in place of the `unset-logging-level` command.

### Arguments

Table 170: Command Arguments Overview

Argument	Description
<APP>	The name of the application whose logging level you want to change
<COMPONENT>	The name of the application component whose logging level you want to change

## list-logging-levels

List the logging levels that have been manually configured for an application.

### Usage

```
xs list-logging-levels <APP>
```

#### ➔ Tip

You can use the alias `lli` in place of the `list-logging-levels` command.

## Arguments

Table 171: Command Arguments Overview

Argument	Description
<APP>	The name of the application whose manually configured logging levels you want to list

## env

Show all environment variables set for an application.

### Usage

```
xs env <APP>
```

#### ➔ Tip

You can use the alias `e` in place of the `env` command.

The `xs env` command retrieves details of the application's connection to the SAP HANA database; the retrieved information can be added to the application's `default-services.json` file, if you want to automate the connection process.

## Arguments

Table 172: Command Arguments Overview

Argument	Description
<APP>	The name of the application whose environment variables you want to display

## set-env

Set an environment variables for an application.

### Usage

```
xs set-env <APP> <NAME> <VALUE>
```

#### ➔ Tip

You can use the alias `se` in place of the `set-env` command.

If you use a simple comma-separated list or a JSON structure to define `<VALUE>`, the rules that apply for escaping quotes or spaces included in the string differ according to the command shell you are using (Windows or Unix-like), as illustrated in the following examples

- **<VALUE>** as a comma-separated list:

To use a comma-separated list to define **<VALUE>** in a Unix shell, enclose the complete list in single quotes (' [] '):

```
xs set-env myApp MyVariable '["myhost1", "myhost2"]'
```

To use a comma-separated list to define **<VALUE>** in a Windows shell, escape each double quote character ("") with the backslash (\) character:

```
xs set-env myApp MyVariable [\"myhost1\", \"myhost2\"]
```

- **<VALUE>** as a JSON structure:

In a Unix shell, if you define **<VALUE>** in a the JSON structure that contains quotes ({} "") or a space (), you must enclose the complete JSON structure in single quotes (' {} '), as illustrated in the following example:

```
xs set-env myApp MyVariable '{"user":"myuser", "password":"sdfg2345"}'
```

If youIn specify a Windows shell, if the JSON structure used to define **<VALUE>** contains quotes ({} "") If you), you must “escape” the quotes used to specify each values in the list. If the JSON structure also contains a space, you must escape the whole JSON structure with additional double quotes ({} ""), as illustrated in the following example:

```
xs set-env myApp MyVariable "{\"user\":\"myuser\", \"password\":\"sdfg2345\"}”
```

The following example for a Windows shell shows how to escape quotes in a JSON structure that **does not** contain any space characters:

```
xs set-env myApp MyVariable {\"user\": \"myuser\", \"password\": \"sdfg2345\"}
```

## Arguments

Table 173: Command Arguments Overview

Argument	Description
<b>&lt;APP&gt;</b>	The name of the application whose environment variables you want to display
<b>&lt;NAME&gt;</b>	The environment variable to set
<b>&lt;VALUE&gt;</b>	<p>The new value for the environment variable. VALUE is typically a simple string that specifies the</p> <p><b>i Note</b></p> <p><b>&lt;VALUE&gt;</b> as a comma-separated list or a JSON structure, you must ensure that characters are correctly “escaped” in the command line argument. Escape rules differ according to the command shell you are using, for example, Windows, Unix, as illustrated in the examples above.</p>

## unset-env

Remove one or more environment variable settings for an application.

### Usage

```
xs unset-env <APP> <NAME>
```

### Arguments

Table 174: Command Arguments Overview

Argument	Description
<APP>	The name of the application whose environment variable you want to remove
<NAME>	The environment variable to remove

## pinned-hosts

List current host pinnings for all applications in the target space. In strict mode, an application can only run on the host to which it is pinned; in relaxed mode can also run on other hosts, if the host to which it is pinned is not available at run time.

### Usage

```
xs pinned-hosts
```

## pin-hosts

Pin an application to one or more execution-agent hosts.

### Usage

```
xs pin-hosts <APP> [--hosts <HOSTS>] [--eas <EXECUTION_AGENTS>] [--strict | --relaxed]
```

### Arguments

Table 175: Command Arguments Overview

Argument	Description
<APP>	The name of the application whose environment variable you want to remove

## Options

Table 176: Command Options Overview

Option	Description
--hosts <HOSTS>	A comma-separated list of host names providing execution agents
--eas <EXECUTION_AGENTS>	A comma-separated list of execution agent indices. Use the command <code>xs system-info</code> to display a list of execution-agent indices
--strict	Use <b>strict</b> mode ( <b>default</b> ); if no pinned host is available, the selected application cannot start
--relaxed	Use <b>relaxed</b> pinning mode; if no pinned host is available, other hosts may run the application

## enable-debugging

Enable debugging for an XS advanced application.

### Usage

```
xs enable-debugging <APP> [--pid <PID>] [--port <PORT>]
```

If no `<PORT>` or `<PID>` is provided, the command returns the following response:

#### Output Code

```
Enabling debug mode for app '<APP>'...
Debugging mode is enabled.
Waiting for debugger connection on: localhost:<PORT>
```

#### i Note

The indicated port is always opened on the **client** host - not the remote host.

## Arguments

Table 177: Command Arguments Overview

Argument	Description
<APP>	The name of the application for which you want to enable debugging

## Options

Table 178: Command Options Overview

Option	Description
--pid <PID>	Optional ID of the application process (PID) for which you want to enable debugging
--port <PORT>	The number of an optional PORT that should be opened locally for the connection to the debug session
--startup	Start the application in debugging mode.
--websocket	Use Web socket for the connection.  <b>Note</b> This is an alternative method to establish the connection; there is no difference to the default method in terms of functionality.

## disable-debugging

Disable debugging for an XS advanced application.

### Usage

```
xs disable-debugging <APP> [--pid <PID>]
```

### Arguments

Table 179: Command Arguments Overview

Argument	Description
<APP>	The name of the application for which you want to disable debugging

## Options

Table 180: Command Options Overview

Option	Description
--pid <PID>	Optional ID of the application process (PID) for which you want to disable debugging

## debugging-info

Display detailed information about the debugging mode set for an XS advanced application.

## Usage

```
xs debugging-info <APP> [--pid <PID>]
```

## Arguments

Table 181: Command Arguments Overview

Argument	Description
<APP>	The name of the application for which you want to display debugging details

## Options

Table 182: Command Options Overview

Option	Description
--pid <PID>	Optional ID of the application process (PID) for which you want to display debugging details

# java

Obtain a heap-dump or thread-dump from a running Java application.

## Usage

```
xs java [heap-dump|thread-dump] <APP> [-o <PATH>] [-i <INSTANCE>] [--pid <PID>] [--std-out]
```

## Arguments

Table 183: Command Arguments Overview

Argument	Description
heap-dump, thread-dump	The operation to perform on the selected Java application
<APP>	The name of the Java application for which you want to display heap- or thread-dump information

## Options

Table 184: Command Options Overview

Option	Description
-o <PATH> --output-directory <PATH>	The directory to store the generated dump file. By default, this is the current working directory.

Option	Description
<code>-i &lt;INSTANCE&gt; --instance-index &lt;INSTANCE&gt;</code>	<p>Trigger the dump operation for this application instance.</p> <p><b>Tip</b></p> <p>Use the command <code>xs app &lt;APPname&gt;</code> to display a list of all available instances. By default, the dump is triggered for the first running instance found.</p>
<code>--pid &lt;PID&gt;</code>	Optional ID of the application process (PID) for which you want to trigger the dump operation
<code>--std-out</code>	Print the dump file to standard out

## nodejs

Obtain a heap-dump or thread-dump from a running Node.js application.

### Usage

```
xs nodejs [heap-dump|thread-dump] <APP> [-o <PATH>] [-i <INSTANCE>] [--pid <PID>] [--std-out]
```

### Arguments

Table 185: Command Arguments Overview

Argument	Description
<code>heap-dump, thread-dump</code>	The operation to perform on the selected Node.js application
<code>&lt;APP&gt;</code>	The name of the Node.js application for which you want to display heap- or thread-dump information

### Options

Table 186: Command Options Overview

Option	Description
<code>o &lt;PATH&gt; --output-directory &lt;PATH&gt;</code>	The directory to store the generated dump file. By default, this is the current working directory.

Option	Description
<code>-i &lt;INSTANCE&gt; --instance-index &lt;INSTANCE&gt;</code>	<p>Trigger the dump operation for this application instance.</p> <p><b>➔ Tip</b></p> <p>Use the command <code>xs app &lt;APPname&gt;</code> to display a list of all available instances. By default, the dump is triggered for the first running instance found.</p>
<code>--pid &lt;PID&gt;</code>	Optional ID of the application process (PID) for which you want to trigger the dump operation
<code>--std-out</code>	Print the dump file to standard out

## Related Information

[XS CLI: Application Management \[page 852\]](#)

[The XS Command-Line Interface Reference \[page 848\]](#)

## 13.3 XS CLI: Services Management

Maintain SAP HANA XS services, for example: listing, creating, deleting, binding, updating.

Table 187: XS CLI: Services Command Overview

Command	Alias	Description
<code>marketplace</code>	m	List available services in the marketplace
<code>service</code>		Display details of a service instance
<code>services</code>	s	List all services in the target space
<code>managed-services</code>		List all instances of a managed service in the target space
<code>managed-service</code>		Display details of a selected instance of a managed service in the target space
<code>create-service</code>	cs	Create a service instance
<code>update-service</code>		Update a service instance
<code>delete-service</code>	ds	Delete a service instance

Command	Alias	Description
<code>rename-service</code>		Rename a service instance
<code>bind-service</code>	bs	Bind a service instance to an application
<code>unbind-service</code>	us	Remove the binding between a service instance and an application
<code>service-keys</code>	sk	List all service keys for a specified service instance.
<code>service-key</code>	sk	Displays a service key for a specified service instance.
<code>create-service-key</code>	csk	Create a service key for a specified service instance.
<code>delete-service-key</code>	dsk	Delete a service key for a specified service instance.
<code>service-brokers</code>		List all available service brokers
<code>create-service-broker</code>		Create a service broker
<code>delete-service-broker</code>		Delete a service broker
<code>update-service-broker</code>		Update a service broker
<code>rename-service-broker</code>		Rename a service broker
<code>create-user-provided-service</code>	cups	Create a user-provided service instance and make it available for use by applications.
<code>update-user-provided-service</code>	uups	Update the name-values pairs used to define a user-provided service instance
<code>register-service-url</code>		Register a Uniform Resource Locator (URL) for a named service.
<code>unregister-service-url</code>		Unregister a service URL

## marketplace

List all services currently available in the service marketplace.

### Usage

```
xs m[arketplace] [-s <SERVICE_NAME>]
```

## ➔ Tip

Use the `marketplace` command to display a list of the available service **plans** provided by the registered service brokers. Use the `services` command to display a list of service **instances**, for example, those created with the commands `create-service` or `create-user-provided-service`.

To display a list of the service plans available with the `hana` service:

```
xs marketplace -s hana
Getting services from marketplace...
Getting plans for service "hana"...
service plan      description          free/paid
-----
hdi-shared        HDI container on a HANA database    free
sbss             User with permissions to use SBSS    free
schema           Schema on a HANA database            free
securestore      User with permissions to use secure store  free
```

## ➔ Tip

SBSS stands for "Service Broker Security Support".

## Command Options

Table 188: Command Options Overview

Option	Description
<code>-s &lt;SERVICE_NAME&gt;</code>	List the service plans available for the service specified in <code>&lt;SERVICE_NAME&gt;</code>

## service

Display details of a selected instance of a service in the target space.

### Usage

```
xs service <SERVICE_INSTANCE> [--guid]
```

The following example shows how display information about the a specified service instance:

```
xs service fp-db
Getting service "fp-db" in org "myorg" / space "SAP" as XSA_DEV...
Service instance: fp-db
Service: hana
Bound apps: fileprocessor-db, fileprocessor-master, fileprocessor-worker
Tags:
Plan: hdi-shared
Description: HDI container on a HANA database
```

You can use the `--guid` option to supress all information about the service instance **except** the service instance's globally unique identifier (GUID):

```
xs service fp-db --guid
```

c41e81f3-077c-440e-99b2-83004c0fa8c9

### ➔ Tip

To find out the name of a specific service instance, use the `xs services` command to display a list of all the services currently running in the target space.

## Arguments

Table 189: Command Arguments Overview

Argument	Description
<code>&lt;SERVICE_INSTANCE&gt;</code>	The name of the service instance, whose details you want to display  <b>i Note</b> Only alpha-numeric characters (aA-zZ, 0-9), hyphens (-), and underscores (_) are allowed.

## Options

Table 190: Command Options Overview

Option	Description
<code>--guid</code>	Display <b>only</b> the globally unique identifier (GUID) of the specified service instance; with this option, all other information about the service is suppressed.

## services

List available services in the target organizational space. You can use the `xs target` command to set or display the target organization or space.

### Usage

```
xs s[ervices]
```

### ➔ Tip

Use the `services` command to display a list of service **instances**, for example, those created with the commands `create-service` or `create-user-provided-service`. Use the `marketplace` command to display a list of the available service **plans** provided by the registered service brokers.

## managed-services

Display a list of all instances of a managed service in the target space.

## Usage

```
xs managed-services <SERVICE_INSTANCE>
```

## Arguments

Table 191: Command Arguments Overview

Argument	Description
<SERVICE_INSTANCE>	The name of the managed service, whose instances you want to display in a list  <b>i Note</b> Only alpha-numeric characters (aA-zZ, 0-9), hyphens (-), and underscores (_) are allowed.

## managed-service

Display details of a selected instance of a managed service in the target space.

## Usage

```
xs managed-service <SERVICE_INSTANCE> <TENANT>
```

## Arguments

Table 192: Command Arguments Overview

Argument	Description
<SERVICE_INSTANCE>	The name of the managed service, whose instances you want to display in a list  <b>i Note</b> Only alpha-numeric characters (aA-zZ, 0-9), hyphens (-), and underscores (_) are allowed.
<TENANT>	The tenant ID for the managed-service instance whose details you want to display

## create-service

Create a service instance.

## Usage

```
xs create-service <SERVICE> <PLAN> <SERVICE_INSTANCE> [-c <PARAMETERS>] [-t <TAGS>]
```

## ➔ Tip

Use the command `xs create-user-provided-service` to make user-provided services available to XS applications.

## Arguments

Table 193: Command Arguments Overview

Argument	Description
<code>&lt;SERVICE&gt;</code>	Name of the service to create
<code>&lt;PLAN&gt;</code>	Name of the service plan
<code>&lt;SERVICE_INSTANCE&gt;</code>	Name of your service instance  <b>i Note</b> Only alpha-numeric characters (aA-zZ, 0-9), hyphens (-), and underscores (_) are allowed.
<code>&lt;TAGS&gt;</code>	One or more user-defined strings that can be used by an application to identify a service instance.

## Options

Table 194: Command Options Overview

Option	Description
<code>-c &lt;PARAMETERS&gt;</code>	Provide the name of a JSON file containing service-specific parameters or provide any parameters inline as a JSON-compliant string
<code>-t &lt;TAGS&gt;</code>	Specify one or more user-defined tags to associate with the new service instance; multiple tags must be separated by commas.

## update-service

Update a service instance.

### Usage

```
xs update-service <SERVICE_INSTANCE> [-p <NEW_PLAN>] [-c <PARAMETERS_AS_JSON>] [-t <TAGS>]
```

## Arguments

Table 195: Command Arguments Overview

Argument	Description
<SERVICE_INSTANCE>	Name of your service instance  <b>i Note</b> Only alpha-numeric characters (aA-zZ, 0-9), hyphens (-), and underscores (_) are allowed.

## Options

Table 196: Command Options Overview

Option	Description
-p <NEW_PLAN>	The name of the new service plan to use after the update
-c <PARAMETERS_AS_JSON>	The name of a JSON file containing service-specific parameters or provide any parameters inline as a JSON-compliant string
-t <TAGS>	One or more user-defined tags to associate with the service instance that you want to update; multiple tags must be separated by commas.

## delete-service

Delete a service instance.

### Usage

```
xs delete-service <SERVICE_INSTANCE> [-f] [--purge]
```

## Arguments

Table 197: Command Arguments Overview

Argument	Description
<SERVICE_INSTANCE>	Name of the service instance to delete

## Options

Table 198: Command Options Overview

Option	Description
-f	Force deletion (no confirmation is required)
--purge	Force deletion even in the event of service broker errors

## rename-service

Rename a service instance.

### Usage

```
xs rename-service <SERVICE_INSTANCE> <NEW_SERVICE_INSTANCE>
```

### Arguments

Table 199: Command Arguments Overview

Argument	Description
<SERVICE_INSTANCE>	The current name of the service instance which you want to rename
<NEW_SERVICE_INSTANCE>	The <b>new</b> name of the service instance

## bind-service

Bind a service instance to an application.

### Usage

```
xs bind-service <APP> <SERVICE_INSTANCE> [-c <PARAMETERS>]
```

#### ➔ Tip

You can use the alias `bs` in place of the command `bind-service`.

### Arguments

Table 200: Command Arguments Overview

Argument	Description
<APP>	The name of the application to which you want to bind the service instance
<SERVICE_INSTANCE>	The name of the service instance to bind to the application named in APP

### Options

Table 201: Command Options Overview

Option	Description
-c <PARAMETERS>	Provide the name of a JSON file containing service-specific parameters or provide any parameters inline as a JSON-compliant string

## unbind-service

Remove the binding between a service instance and an application.

### Usage

```
xs unbind-service <APP> <SERVICE_INSTANCE> [-c <PARAMETERS>]
```



You can use the alias `us` in place of the command `unbind-service`.

### Arguments

Table 202: Command Arguments Overview

Argument	Description
<code>&lt;APP&gt;</code>	The name of the application from which you want to unbind the service instance
<code>&lt;SERVICE_INSTANCE&gt;</code>	The name of the service instance to unbind from the application named in APP

### Options

Table 203: Command Options Overview

Option	Description
<code>--purge</code>	Force unbinding operation even in the event of service-broker errors

## service-keys

List all service keys for a specified service instance.

### Usage

```
xs service-keys [<SERVICE_INSTANCE>]
```



You can use the alias `sks` in place of the command `service-keys`.

### Arguments

Table 204: Command Arguments Overview

Argument	Description
<code>&lt;SERVICE_INSTANCE&gt;</code>	The name of the service instance whose service keys you want to display

## service-key

Displays a service key for a specified service instance.

### Usage

```
xs service-keys <SERVICE_INSTANCE> <SERVICE_KEY> [--guid]
```

#### ➔ Tip

You can use the alias `sk` in place of the command `service-key`.

### Arguments

Table 205: Command Arguments Overview

Argument	Description
<code>&lt;SERVICE_INSTANCE&gt;</code>	The name of the service instance whose service keys you want to display
<code>&lt;SERVICE_KEY&gt;</code>	The name of the service key

### Options

Table 206: Command Options Overview

Option	Description
<code>--guid</code>	Display the GUID of the service key only

## create-service-key

Create a service key for a specified service instance.

### Usage

```
xs create-service-key <SERVICE_INSTANCE> <SERVICE_KEY> [-c <PARAMETERS_AS_JSON>]
```

#### ➔ Tip

You can use the alias `csk` in place of the command `create-service-key`.

### Arguments

Table 207: Command Arguments Overview

Argument	Description
<code>&lt;SERVICE_INSTANCE&gt;</code>	The name of the service instance for which you want to create a key
<code>&lt;SERVICE_KEY&gt;</code>	The name of the service key

## Options

Table 208: Command Options Overview

Option	Description
-c <PARAMETERS_AS_JSON>	Provide the name of a JSON file containing service-specific parameters or provide any parameters inline as a JSON-compliant string

## delete-service-key

Delete a service key for a specified service instance.

### Usage

```
xs delete-service-key <SERVICE_INSTANCE> <SERVICE_KEY> [-f]
```

#### ➔ Tip

You can use the alias `dsk` in place of the command `delete-service-key`.

## Arguments

Table 209: Command Arguments Overview

Argument	Description
<SERVICE_INSTANCE>	The name of the service instance whose key you want to delete
<SERVICE_KEY>	The name of the service key to delete

## Options

Table 210: Command Options Overview

Option	Description
-f	Force deletion without confirmation
--purge	Force deletion even in the event of service-broker errors

## service-brokers

List all available service brokers.

### Usage

```
xs service-brokers
```

## create-service-broker

Create a service broker.

### Usage

```
xs create-service-broker <SERVICE_BROKER> [-u <USERNAME>] [-p <PASSWORD>] [-s <SERVICE_URL>]  
xs create-service-broker <SERVICE_BROKER> <USERNAME> <PASSWORD> <SERVICE_URL>
```

### Arguments

Table 211: Command Arguments Overview

Argument	Description
<SERVICE_BROKER>	The name of the service broker to create
<USERNAME>	The name of the user required to create the new service-broker instance
<PASSWORD>	The password for the user specified in USERNAME
<SERVICE_URL>	The URL to connect to the new service-broker instance

### Options

Table 212: Command Options Overview

Option	Description
-u <USERNAME>	The name of the user required to create the new service-broker instance
-p <PASSWORD>	The password for the user specified in <USERNAME>
-s <SERVICE_URL>	The URL to connect to the new service-broker instance

## update-service-broker

Update a service broker.

### Usage

#### Code Syntax

```
xs update-service-broker <SERVICE_BROKER> <USERNAME> <PASSWORD> <URL>
```

## Arguments

Table 213: Command Arguments Overview

Argument	Description
<SERVICE_BROKER>	The name of the service broker to update
<USERNAME>	The name of the user required to update the new service-broker instance
<PASSWORD>	The password for the user specified in USERNAME
<URL>	The URL to connect to the updated service-broker instance

## delete-service-broker

Delete a service broker.

### Usage

```
xs delete-service-broker <SERVICE_BROKER> <USERNAME> <PASSWORD> <URL>
```

## Arguments

Table 214: Command Arguments Overview

Argument	Description
<SERVICE_INSTANCE>	Name of the service-broker to delete

### Options

Table 215: Command Options Overview

Option	Description
-f	Force deletion (no confirmation is required)

## rename-service-broker

Rename a service broker.

### Usage

```
xs rename-service-broker <SERVICE_BROKER> <USERNAME> <PASSWORD> <URL>
```

## Arguments

Table 216: Command Arguments Overview

Argument	Description
<SERVICE_BROKER>	The existing name of the service broker whose name you want to change
<NEW_SERVICE_BROKER>	The new name of the service broker

## create-user-provided-service

Create a user-provided service instance and make it available for use by applications.

### Usage

```
xs create-user-provided-service <SERVICE_INSTANCE> [-p <PARAMETERS>] [-l <SYSLOG_DRAIN_URL>]
```

#### ➔ Tip

You can use the alias `cups` in place of the command `create-user-provided-service`.

Pass comma separated parameter names to enable interactive mode:

```
xs create-user-provided-service <SERVICE_INSTANCE> "comma, separated, parameter, names"
```

Pass parameters as JSON to update a service non-interactively:

```
xs create-user-provided-service <SERVICE_INSTANCE> "{\"name\":\"value\", \"name\":\"value\"}"
```

## Arguments

Table 217: Command Arguments Overview

Argument	Description
<SERVICE_INSTANCE>	The name of the service instance to create

## Options

Table 218: Command Options Overview

Option	Description
-p	A list of parameters to be updated; you can supply the parameters in the following ways: <ul style="list-style-type: none"><li>• Comma-separated values in quotes, for example: <code>-p "parm1, parm2, parm3"</code></li><li>• A JSON-compliant string, for example, <code>"{ \"name\":\"value\", \"name\":\"value\" }"</code></li></ul>

Option	Description
-l	The URL to connect the Syslog Drain

## Examples

```
xs create-user-provided-service my-db-service -p "host, port, dbname, username, password"
```

```
xs create-user-provided-service my-db-service -p "{\"username\":\"admin\",\"password\":\"pa55woRD\"}"
```

```
xs create-user-provided-service my-drain-service -l syslog://example.com
```

## update-user-provided-service

Update the name-values pairs used to define a user-provided service instance.

### Usage

```
xs update-user-provided-service <SERVICE_INSTANCE> [-p <PARAMETERS>] [-l <SYSLOG_DRAIN_URL>]
```

#### Tip

You can use the alias `uups` in place of the `update-user-provided-service` command.

Pass comma separated parameter names to enable interactive mode:

```
xs update-user-provided-service <SERVICE_INSTANCE> "comma, separated, parameter, names"
```

Pass parameters as JSON to update a service non-interactively:

```
xs update-user-provided-service <SERVICE_INSTANCE> "{\"name\":\"value\", \"name\":\"value\"}"
```

## Arguments

Table 219: Command Arguments Overview

Argument	Description
<SERVICE_INSTANCE>	The name of the service instance to create

## Options

Table 220: Command Options Overview

Option	Description
-p	A list of parameters to be updated; you can supply the parameters in the following ways: <ul style="list-style-type: none"><li>• Comma-separated values in quotes, for example: -p "parm1, parm2, parm3"</li><li>• A JSON-compliant string, for example,     "{"name": "value", "name": "value"}"</li></ul>
-l	The URL to connect the Syslog Drain

## Examples

```
xs update-user-provided-service my-db-service -p "host, port, dbname, username, password"
```

```
xs update-user-provided-service my-db-service -p {"username": "admin", "password": "pa55woRD"}
```

```
xs update-user-provided-service my-drain-service -l syslog://example.com
```

## register-service-url

Register a Uniform Resource Locator (URL) for a named service.

### Usage

```
xs register-service-url <SERVICE_NAME> <URL>
```

### Arguments

Table 221: Command Arguments Overview

Argument	Description
<SERVICE_NAME>	Name of the service for which you want to register a URL
<URL>	The URL to register for the service specified in <SERVICE_NAME>

## unregister-service-url

Unregister the URL assigned to a service.

### Usage

```
xs unregister-service-url <SERVICE_NAME>
```

## Arguments

Table 222: Command Arguments Overview

Argument	Description
<SERVICE_NAME>	Name of the service to remove from the registry

## Related Information

[XS CLI: Services Management \[page 874\]](#)

[The XS Command-Line Interface Reference \[page 848\]](#)

## 13.4 XS CLI: Organizations

Maintain user organizations, for example: create, list, rename, delete.

Organizations enable developers to collaborate by sharing resources, services, and applications. Access to the shared resources, services, and applications is controlled by roles, for example, “Org Manager” or “Org Auditor”; the role defines the scope of the permissions assigned to the named user in the organization. For example, an Org Manager can create and manage users, create, modify, or delete organizational spaces, and add domains to the organization.

Table 223: XS Organizations Commands Overview

Command	Alias	Description
<code>orgs</code>	o	List all defined organizations
<code>create-org</code>	co	Create a new organization
<code>delete-org</code>		Delete an existing organization
<code>rename-org</code>		Rename an existing organization

### orgs

List all defined organizations.

#### Usage

```
xs orgs
```

## ➔ Tip

You can use the alias `o` in place of the `orgs` command.

## create-org

Create a new organization.

### Usage

```
xs create-org <ORG>
```

## ➔ Tip

You can use the alias `co` in place of the `create-org` command.

### Arguments

Table 224: Command Arguments Overview

Argument	Description
<code>&lt;ORG&gt;</code>	Name of the new organization to create

## delete-org

Delete an existing organization.

### Usage

```
xs delete-org <ORG> [-f]
```

### Arguments

Table 225: Command Arguments Overview

Argument	Description
<code>&lt;ORG&gt;</code>	Name of the organization to delete

### Options

Table 226: Command Options Overview

Option	Description
<code>-f</code>	Force deletion of the organization without any confirmation prompt

Option	Description
--quiet	Do not display an error message if the specified organization does not exist

## rename-org

Rename an existing organization.

### Usage

```
xs rename-org <ORG> <NEW_ORG>
```

### Arguments

Table 227: Command Arguments Overview

Argument	Description
<ORG>	Current name of the organization to rename
<NEW_ORG>	New name of the organization

## Related Information

[XS CLI: Organizations \[page 890\]](#)

[The XS Command-Line Interface Reference \[page 848\]](#)

## 13.5 XS CLI: Spaces

Maintain and manage user spaces in an organization, for example: create, list, rename, delete.

In an organization, spaces enable users to access shared resources that can be used to develop, deploy, and maintain applications. Access to the resources is controlled by roles, for example, "Space Manager", "Space Developer" or "Space Auditor"; the role defines the scope of the permissions assigned to the named user in the organizational space. For example, a Space Developer can deploy and start an application or rename an organizational space.

Table 228: XS Spaces Commands Overview

Command	Description
<a href="#">spaces</a>	List all defined spaces

Command	Description
<code>space</code>	Show details of a particular space
<code>create-space</code>	Create a new space
<code>delete-space</code>	Delete an existing space
<code>rename-space</code>	Rename an existing space
<code>update-space</code>	Update the settings of an existing space

## spaces

List all defined spaces.

### Usage

```
xs spaces
```

## space

Display details of a particular space.

### Usage

```
xs space
```

### Options

Table 229: Command Options Overview

Option	Description
<code>--guid</code>	Retrieve and display the group ID (GUID) for the specified space. All other output for the space is suppressed.

## create-space

Create a new space.

### Usage

```
xs create-space <SPACE> [-o <ORG>]
```

Set a specific user for task execution in this space:

```
xs create-space <SPACE> [-o <ORG>] -u <USERNAME>
```

## Arguments

Table 230: Command Arguments Overview

Argument	Description
<SPACE>	Name of the new space to create

## Options

Table 231: Command Options Overview

Option	Description
-o <ORG>	Name to the organization to which you want to add the newly created space
-u <USERNAME>	The name of the user used to perform tasks in this space

## delete-space

Delete an existing space.

### Usage

```
xs delete-space <SPACE> [-f]
```

## Arguments

Table 232: Command Arguments Overview

Argument	Description
<SPACE>	Name of the space to delete

## Options

Table 233: Command Options Overview

Option	Description
-f	
--quiet	Do not display an error message if the specified space does not exist

## rename-space

Rename an existing space.

## Usage

```
xs rename-space <SPACE> <NEW_SPACE>
```

## Arguments

Table 234: Command Arguments Overview

Argument	Description
<SPACE>	Current name of the space you want to rename
<NEW_SPACE>	New name for the space

## update-space

Update the settings of an existing space. Force deletion of the space without any confirmation prompt

### Usage

Update execution user for a space:

```
xs update-space <SPACE> -u <USERNAME>
```

or

```
xs update-space <SPACE> --unset-user
```

Update default host pinning settings for a space:

```
xs update-space <SPACE> [--hosts <HOSTS> Force deletion of the space without any  
confirmation] [--eas <EXECUTION_AGENTS>]
```

or

```
xs update-space <SPACE> --unset-pinnings
```

### Tip

You can supply the names of the execution agents in a comma-separated list: either as host names (–host) or as indices (–eas). For more information about the –eas option, see the `xs system-info` command.

## Arguments

Table 235: Command Arguments Overview

Argument	Description
<SPACE>	The name of the space whose settings you want to modify

## Options

Table 236: Command Options Overview

Option	Description
<code>-u &lt;USERNAME&gt;</code>	The name of the user used to run applications in this space
<code>--unset-user</code>	Reset the execution user setting
<code>--hosts &lt;USERNAME&gt;</code>	Comma-separated list of host names
<code>--eas &lt;EXECUTION_AGENTS&gt;</code>	Comma-separated list of execution agent indices
<code>--strict</code>	Use <b>strict</b> mode: if no pinned host is available, the app will not start (default)
<code>--relaxed</code>	Use <b>relaxed</b> pinning mode: other hosts may run the app, if no pinned host is available
<code>--unset-pinning</code>	Reset the current host-pinning setting

## Related Information

[XS CLI: Spaces \[page 892\]](#)

[The XS Command-Line Interface Reference \[page 848\]](#)

## 13.6 XS CLI: Domains

Maintain domains and certificates.

Table 237: XS Domains Command Overview

Command	Description
<code>domains</code>	List all known domains
<code>create-domain</code>	Create a new domain
<code>delete-domain</code>	Delete an existing domain
<code>set-certificate</code>	Sets the SSL certificate to be used for a domain
<code>delete-certificate</code>	Deletes the SSL certificate used by a domain

## domains

List all known domains.

### Usage

```
xs domains
```

## create-domain

Create a new domain.

### Usage

```
xs create-domain <DOMAIN> [-o <ORG>]
```

### Arguments

Table 238: Command Arguments Overview

Argument	Description
<DOMAIN>	The name of the domain to create

### Options

Table 239: Command Options Overview

Option	Description
-o <ORG>	The name of the organization to which the domain belongs. If the domain is "shared", the organization field is empty.

## delete-domain

Delete an existing domain.

### Usage

```
xs delete-domain <DOMAIN> [-f]
```

## Arguments

Table 240: Command Arguments Overview

Argument	Description
<DOMAIN>	The name of the domain to delete

## Options

Table 241: Command Options Overview

Option	Description
-f <ORG>	Force deletion without any system prompt or confirmation
--quiet	Hide any error messages generated when you try to delete a non-existent domain

## set-certificate

Set the SSL certificate to be used for a domain.

### Usage

```
xs set-certificate <DOMAIN> -k <KEY_FILE> -c <CERT_FILE>
```

## Arguments

Table 242: Command Arguments Overview

Argument	Description
<DOMAIN>	The name of the domain for which the certificate is to be set
<KEY_FILE>	The RSA key file in PKCS8 PEM format
<CERT_FILE>	The X.509 certificate-chain file in PEM format

## Options

Table 243: Command Options Overview

Option	Description
-k <KEY_FILE>	Specify the path to (and name of) an RSA key file
-c <CERT_FILE>	Specify the path to (and name of) an X.509 certificate-chain file

## delete-certificate

Delete the SSL certificate used by a domain.

## Usage

```
xs delete-certificate <DOMAIN>
```

## Arguments

Table 244: Command Arguments Overview

Argument	Description
<DOMAIN>	The name of the domain whose SSL certificate is to be deleted

## Related Information

[XS CLI: Domains \[page 896\]](#)

[The XS Command-Line Interface Reference \[page 848\]](#)

## 13.7 XS CLI: Certificates

Maintain and manage X509 certificates, for example: list certificates, add or delete a certificate.

Table 245: XS Certificates Commands Overview

Command	Alias	Description
<a href="#">trusted-certificates</a>		Retrieve the list of trusted certificates
<a href="#">trust-certificate</a>		Add a trusted X509 certificate
<a href="#">untrust-certificate</a>		Delete a trusted X509 certificate

## trusted-certificates

Retrieve the list of trusted certificates.

## Usage

```
xs trusted-certificates
```

## trust-certificate

Add a trusted X509 certificate.

### Usage

```
xs trust-certificate <ALIAS> -c <CERT_FILE> [-u HTTP|JDBC]
```

### Arguments

Table 246: Command Arguments Overview

Argument	Description
<ALIAS>	The alias of the trusted certificate

### Options

Table 247: Command Options Overview

Option	Description
-c <CERT_FILE>	The path to the X509 certificate chain file in PEM format
-u <HTTP   JDBC>	The main certificate usage: HTTP ( <b>default</b> ) or JDBC

## untrust-certificate

Delete a trusted X509 certificate.

### Usage

```
xs untrust-certificate <ALIAS>
```

### Arguments

Table 248: Command Arguments Overview

Argument	Description
<ALIAS>	The alias of the trusted certificate

## Related Information

[The XS Command-Line Interface Reference \[page 848\]](#)

## 13.8 XS CLI: Routes

Maintain and manage application routes, for example: create, list, map, unmap.

Table 249: XS Routes Commands Overview

Command	Alias	Description
<code>routes</code>	<code>r</code>	List all routes in the current space
<code>create-route</code>		Create a URL route in a space for later use
<code>map-route</code>		Assign or change the route assigned to an application
<code>unmap-route</code>		Remove the URL route assigned to an application
<code>delete-route</code>		Delete a route from a space

### routes

List all defined routes.

#### Usage

```
xs routes
```

### create-route

Create a URL route in a space for later use (for example, for assignment to an application).

#### Usage

```
xs create-route <SPACE> <DOMAIN> [-n <HOSTNAME>] [--path <PATH>]
```

Or:

```
xs create-route <SPACE> -p <PORT> [<DOMAIN>] [--path <PATH>]
```

#### Arguments

Table 250: Command Arguments Overview

Argument	Description
<code>&lt;SPACE&gt;</code>	Name of the space in which to create the new route

Argument	Description
<DOMAIN>	The name of the domain containing the space where the new route must be created

## Options

Table 251: Command Options Overview

Option	Description
-n <HOSTNAME>	Name of the host to use for the new route
--hostname <HOSTNAME>	
-p <PORT>	The port number to user for connections using port-based routing
--path <PATH>	The context path, for example, /myapp

## map-route

Map an existing route.

### Usage

```
xs map-route <APP> <DOMAIN> [-n <HOSTNAME>] [--path <PATH>]

xs map-route <APP> -p <PORT> [<DOMAIN>] [--path <PATH>]
```

### Arguments

Table 252: Command Arguments Overview

Argument	Description
<APP>	Name of the application to which you want to map the route
<DOMAIN>	The name of the domain containing the application where the route must be mapped

### Options

Table 253: Command Options Overview

Option	Description
-n <HOSTNAME>	Name of the host to map the route to
--hostname <HOSTNAME>	
-p <PORT>	The port number to use for connections with port-based routing
--path <PATH>	The context path, for example, /myapp

## unmap-route

Remove an existing route assignment from an application.

### Usage

```
xs unmap-route <APP> <DOMAIN> [-n <HOSTNAME>] [--path <PATH>]
```

or

```
xs unmap-route <APP> [-p <PORT>] [<DOMAIN>] [--path <PATH>]
```

### Arguments

Table 254: Command Arguments Overview

Argument	Description
<APP>	Name of the application from which you want to remove the route mapping
<DOMAIN>	The name of the domain containing the application where the route must be removed

### Options

Table 255: Command Options Overview

Option	Description
-n <HOSTNAME>	Name of the host from which to remove the route mapping
--hostname <HOSTNAME>	
-p <PORT>	The port number to use for connections with port-based routing
--path <PATH>	The context path, for example, /myapp

## delete-route

Delete an existing route.

### Usage

```
xs delete-route <DOMAIN> [-n <HOSTNAME>] [--path <PATH>] [-f]
```

```
xs delete-route -p <PORT> [<DOMAIN>] [--path <PATH>] [-f]
```

## Arguments

Table 256: Command Arguments Overview

Argument	Description
<DOMAIN>	The name of the domain containing the application where the route must be removed

## Options

Table 257: Command Options Overview

Option	Description
-n <HOSTNAME>	Name of the host
--hostname <HOSTNAME>	
-p <PORT>	The port number to use for connections with port-based routing
--path <PATH>	The context path, for example, /myapp
-f	Force deletion of the route without any confirmation prompt

## Related Information

[XS CLI: Routes \[page 901\]](#)

[The XS Command-Line Interface Reference \[page 848\]](#)

## 13.9 XS CLI: Buildpacks

Maintain application buildpacks, for example: create, list, update, rename, and delete.

Table 258: XS Buildpack Commands Overview

Command	Description
<a href="#">buildpacks</a>	List all available buildpacks
<a href="#">create-buildpack</a>	Create a new buildpack
<a href="#">update-buildpack</a>	Update a buildpack
<a href="#">rename-buildpack</a>	Rename a buildpack
<a href="#">delete-buildpack</a>	Delete a buildpack

## buildpacks

List all available buildpacks.

### Usage

```
xs buildpacks
```

## create-buildpack

Create a new buildpack.

### Usage

```
xs create-buildpack <BUILDPACK> <PATH> <POSITION> [--enable|--disable]
```

### Arguments

Table 259: Command Arguments Overview

Argument	Description
<BUILDPACK>	The name of the buildpack to create
<PATH>	The buildpack path; this should be to a zip file or a local directory
<POSITION>	The buildpack position: an integer that sets the buildpack priority.  <b>i Note</b> If multiple buildpacks are available, they are sorted for detection purposes according to priority, where position 1 (one) is the highest priority. A build pack with position 1 (one) is inspected first during any detection process.

### Options

Table 260: Command Options Overview

Option	Description
--enable	Enable the buildpack
--disable	Disable the buildpack

## update-buildpack

Update a buildpack.

## Usage

```
xs update-buildpack <BUILDPACK> [-p <PATH>] [-i <POSITION>] [--enable|--disable]
[--lock|--unlock]
```

## Arguments

Table 261: Command Arguments Overview

Argument	Description
<BUILDPACK>	The name of the buildpack to update

## Options

Table 262: Command Options Overview

Option	Description
-p <PATH>	The buildpack path, should be a zip file or a local directory
-i <POSITION>	The buildpack position: an integer that sets priority  <b>i Note</b> Buildpacks are sorted according to priority: from lowest to highest
--enable	Enable the buildpack
--disable	Disable the buildpack
--lock	Lock the buildpack
--unlock	Unlock the buildpack

## rename-buildpack

Rename a buildpack.

## Usage

```
xs rename-buildpack <BUILDPACK> <NEW_BUILDPACK>
```

## Arguments

Table 263: Command Arguments Overview

Argument	Description
<BUILDPACK>	The current name of the buildpack whose name you want to change

Argument	Description
<NEW_BUILDPACK>	The new name for the buildpack

## delete-buildpack

Delete a buildpack.

### Usage

```
xs delete-buildpack <BUILDPACK> [-f]
```

### Arguments

Table 264: Command Arguments Overview

Argument	Description
<BUILDPACK>	The name of the buildpack to remove

### Options

Table 265: Command Options Overview

Option	Description
-f	Force deletion without any confirmation prompt
--quiet	Do not display any error message if the buildpack specified for deletion does not exist

## Related Information

[XS CLI: Buildpacks \[page 904\]](#)

[The XS Command-Line Interface Reference \[page 848\]](#)

## 13.10 XS CLI: Run-Time Environments

Maintain SAP HANA XS run-time components, for example: create, list, display information, search, update, delete.

Table 266: XS Runtime Commands Overview

Command	Description
<code>runtimes</code>	List all run-time components
<code>runtime</code>	Display information about a specific run-time component
<code>create-runtime</code>	Create a new run-time component
<code>update-runtime</code>	Update the properties of a run-time component
<code>delete-runtime</code>	Delete a run-time component
<code>search-runtime</code>	Search for a run-time component that matches a specified query
<code>pinned-runtimes</code>	List all pinned run-time components for an application
<code>pin-runtime</code>	Pin a run-time component to an application
<code>unpin-runtime</code>	Unpin a run-time component from an application

### runtimes

List all available run-time components.

#### Usage

```
xs runtimes
```

### runtime

Display information about the specified run-time component.

#### Usage

Display information about the run-time component with the specified ID.

```
xs runtime -i <ID>
```

Display information about the run-time component that matches a query:

```
xs runtime <QUERY>
```

## Arguments

Table 267: Command Arguments Overview

Argument	Description
<QUERY>	A query defining the run-time component whose details you want to display  <b>i Note</b> The query must have the same format as the QUERY <a href="#">argument</a> used in the search-runtime command and must result in exactly one run-time.

## Options

Table 268: Command Options Overview

Option	Description
-i <ID>	The ID of the run-time component, whose details you want to display

# create-runtime

Create a new run-time component.

## Usage

```
xs create-runtime -p <PATH> [--inactive] [<DESCRIPTION>]
```

## Arguments

Table 269: Command Arguments Overview

Argument	Description
<DESCRIPTION>	Optional description for the new run-time component you want to create.  <b>➔ Tip</b> If the description contains spaces, enclose the description in quotation marks (""), for example, "This is a description with spaces".

## Options

Table 270: Command Options Overview

Option	Description
-p <PATH>	Path to the run-time directory or Zip archive containing the new run-time component
--inactive	Create the new run-time component in an <b>inactive</b> state

## update-runtime

Update the properties of an existing run-time component.

### Usage

Update the run-time with the specified ID:

```
xs update-runtime -i <ID> [--enable|--disable] [-d <DESCRIPTION>] [-f]
```

Update the properties of the run-time component that matches the specified query:

```
xs update-runtime <QUERY> [--enable|--disable] [-d <DESCRIPTION>] [-f]
```

### Arguments

Table 271: Command Arguments Overview

Argument	Description
<QUERY>	A query defining the run-time component whose properties you want to update  <b>i Note</b> The query must have the same format as the <QUERY> argument used in the <code>search-runtime</code> command and must result in exactly one run-time.
<ID>	The ID of the run-time component whose properties you want to update

## Options

Table 272: Command Options Overview

Option	Description
-i <ID>	Specify the ID of the run-time component whose properties you want to update
--enable	Enable the run-time component specified in <ID> or returned by <QUERY>
--disable	Disable the run-time component specified in <ID> or returned by <QUERY>
-d <DESCRIPTION>	A new description for the run-time component specified in <ID> or returned by <QUERY>

Option	Description
-f	Force update of the run-time component specified in <ID> or returned by <QUERY> without any confirmation prompt

## delete-runtime

Delete an existing run-time component.

### Usage

Delete the run-time with the specified ID:

```
xs delete-runtime -i <ID> [-f]
```

Delete the run-time component that matches the specified query:

```
xs delete-runtime <QUERY> [-f]
```

### Arguments

Table 273: Command Arguments Overview

Argument	Description
<QUERY>	A query defining the run-time component you want to delete <div style="background-color: #ffffcc; padding: 10px;"> <b>i Note</b>            The query must have the same format as the &lt;QUERY&gt; argument used in the search-runtime command and must result in exactly one run-time.         </div>
<ID>	The ID of the run-time component you want to delete

### Options

Table 274: Command Options Overview

Option	Description
-i <ID>	Specify the ID of the run-time component to delete
-u	Remove any unresolved run-time components
-f	Force deletion of the run-time component specified in <ID> or returned by <QUERY> without any confirmation prompt

## search-runtime

Search for a run-time component that best fits the specified search query.

### Usage

```
xs search-runtime <QUERY>
```

### Arguments

Table 275: Command Arguments Overview

Argument	Description
<QUERY>	A query defining the run-time component you want to find. Queries consist of a sequence of simple filters separated by the ampersand character (&).  ➔ <b>Tip</b> If you define multiple filters in <QUERY>, the search returns only those run-time components that match <b>all</b> filters.

### Search Filters

The following filters are supported in <QUERY>:

➔ **Tip**

If you define multiple filters in <QUERY>, the filters must be separated by the ampersand character (&), for example, filter1&filter2

Table 276: Command Filters Overview

Filter	Result
type = <type>	Matches all run-time components with the specified <type>
id = <ID>	Matches all run-time components with the specified <ID>
guid = <PREFIX>	Matches all run-time components with the specified GUID <PREFIX>
version = <major>[.<minor>]	Matches all run-time components with the given <b>major</b> version number. If the <b>minor</b> version is specified, too, this must also match. The search returns only those run-time components that match <b>both</b> major and minor version numbers.
version >= <major>[.<minor>]	Matches all run-time components with at least the specified major and minor version.
version <= <major>[.<minor>]	Matches all run-time components with a version number less than (or equal to) the version specified in <major>[.<minor>].

Filter	Result
<code>version &gt; &lt;major&gt;[.&lt;minor&gt;]</code>	Matches all run-time components with a version number higher than the version specified in <code>&lt;major&gt;[.&lt;minor&gt;]</code> .
<code>version &lt; &lt;major&gt;[.&lt;minor&gt;]</code>	Matches all run-time components with a version number lower than the version specified in <code>&lt;major&gt;[.&lt;minor&gt;]</code> .

**i Note**

If the query matches more than one run-time component, the runtime with the highest version is returned.

## pinned-runtimes

List all pinned run-time components for an application.

### Usage

```
xs pinned-runtimes <APPNAME>
```

### Arguments

Table 277: Command Arguments Overview

Argument	Description
<code>&lt;APPNAME&gt;</code>	The name of the application for which you want to display a list of pinned run-time components

## pin-runtime

Pin a run-time component to an application.

### Usage

```
xs pin-runtime <ALIAS> <APPNAME> -i <ID>
```

When the application requests a run-time component that matches the specified alias, the runtime provided by ID is given to the application.

## Arguments

Table 278: Command Arguments Overview

Argument	Description
<APPNAME>	The name of the application for which you want to display a list of pinned run-time components
<ALIAS>	The name of the run-time component you want to pin to the specified application. The alias can be composed of the "type" and "version" of a run-time component, for example, "sapjvm8" or "node6.9.1".
<ID>	The ID of the run-time component to show

## Options

Table 279: Command Options Overview

Option	Description
-i <ID>	Specify the ID of the pinned run-time component to show

## unpin-runtime

Unpin a run-time component from an application.

### Usage

```
xs unpin-runtime <RUNTIME> <APPNAME>
```

## Arguments

Table 280: Command Arguments Overview

Argument	Description
<APPNAME>	The name of the application for which you want to display a list of pinned run-time components
<RUNTIME>	The name of (or alias for) the run-time component you want to unpin from the specified application

## Related Information

[XS CLI: Run-Time Environments \[page 908\]](#)

[The XS Command-Line Interface Reference \[page 848\]](#)

## 13.11 XS CLI: Tasks

Maintain and manage application-related tasks in the XS advanced run-time environment, for example: list all tasks, run and cancel tasks, and so on.

Table 281: XS Advanced Task Commands Overview

Command	Alias	Description
<code>tasks</code>		List all tasks configured for an XS advanced application
<code>run-task</code>		Run a specified task on an XS advanced application
<code>cancel-task</code>		Cancel a specified task on an XS advanced application

### tasks

List all tasks configured for an XS advanced application.

#### Usage

```
xs tasks <APPNAME>
```

#### Arguments

Table 282: Command Arguments Overview

Argument	Description
<code>&lt;APPNAME&gt;</code>	The name of the XS advanced application whose configured tasks you want to display

### run-task

Run a specified task on an XS advanced application.

#### Usage

```
xs run-task <APPNAME> <TASK> <COMMAND> [-e <ENV_AS_JSON>]
```

#### Arguments

Table 283: Command Arguments Overview

Argument	Description
<code>&lt;APPNAME&gt;</code>	The name of the XS advanced application on which you want to run a task

Argument	Description
<TASK>	The name of the task you want to run
<COMMAND>	The command you want to run as the task

## Options

Table 284: Command Options Overview

Option	Description
-e <ENV_VARS_AS_JSON>   <PATH_TO_ENV_VAR_F ILE>	The task-specific environment, which can be provided either as a JSON string (<ENV_VARS_AS_JSON>) or a file containing the JSON string (<PATH_TO_ENV_VAR_FILE>).

## cancel-task

Stop and cancel a specified task on an XS advanced application.

### Usage

```
xs cancel-task <APPNAME> <TASK>
```

### Arguments

Table 285: Command Arguments Overview

Argument	Description
<APPNAME>	The name of the XS advanced application on which you want to run a task
<TASK>	The name of the task you want to run

## Related Information

[The XS Command-Line Interface Reference \[page 848\]](#)

## 13.12 XS CLI: User Administration

Maintain and manage SAP HANA users, for example: create, list, purge, and delete users; assign and revoke roles in organizations and spaces.

Table 286: XS User Administration Command Overview

Command	Description
<code>users</code>	List all users
<code>purge-users</code>	Delete all users in Controller who are not known to the User Account and Authentication (UAA) service [-f]
<code>space-users</code>	Show space users by role
<code>set-space-role</code>	Assign a space role to a user
<code>unset-space-role</code>	Revoke a space role from a user
<code>org-users</code>	Show organization users by role
<code>set-org-role</code>	Assign a organization role to a user
<code>unset-org-role</code>	Revoke a organization role from a user
<code>roles</code>	Display a list all existing application roles
<code>role-collections</code>	Display a list of all existing application role collections
<code>role-collection</code>	Display details of a specific application role collection
<code>create-role-collection</code>	Create a new application role collection
<code>update-role-collection</code>	Modify an existing application role collection
<code>assigned-role-collections</code>	Display a list of the application role collections currently assigned to a specific user
<code>assign-role-collection</code>	Assign an application role collections to a specific user
<code>unassign-role-collection</code>	Remove an assigned application role collection from a specific user

### users

List all users.

## Usage

```
xs users
```

## purge-users

Delete all users in Controller which are not known to the User Account and Authentication (UAA) service.

## Usage

```
xs purge-users
```

## Options

Table 287: Command Options Overview

Option	Description
-f	Force deletion without showing any confirmation prompt

## space-users

Show space users by role, for example:

- Space Manager
- SpaceDeveloper
- SpaceAuditor

## Usage

```
xs space-users <ORG> <SPACE>
```

## Arguments

Table 288: Command Arguments Overview

Argument	Description
<ORG>	Name of the organization to which the space-users belong
<SPACE>	Name of the organizational space to which the users belong

## set-space-role

Assign a space-specific user role to a user. The following roles are available for assignment:

- SpaceManager  
Invite and manage users, and enable features for a given organizational space
- SpaceDeveloper  
Create and manage applications and services, and view logs and reports
- SpaceAuditor  
View logs, reports, and settings for the specified space

### Usage

```
xs set-space-role <USERNAME> <ORG> <SPACE> <ROLE>
```

### Arguments

Table 289: Command Arguments Overview

Argument	Description
<USERNAME>	Name of the user to whom you want to assign the specified space-specific role
<ORG>	The name of the organization in which the user space is defined
<SPACE>	The name of the organizational space in which you want to assign a user role
<ROLE>	The role you want to assign to the specified user

## unset-space-role

Revoke a space-specific user role from a user. The following space-specific roles can be revoked:

- SpaceManager  
Invite and manage users, and enable features for a given organizational space
- SpaceDeveloper  
Create and manage applications and services, and view logs and reports
- SpaceAuditor  
View logs, reports, and settings for the specified space

### Usage

```
xs unset-space-role <USERNAME> <ORG> <SPACE> <ROLE>
```

## Arguments

Table 290: Command Arguments Overview

Argument	Description
<USERNAME>	Name of the user from whom you want to revoke the specified role
<ORG>	The name of the organization containing the space where a user's role must be revoked
<SPACE>	The name of the organizational space where you want to revoke a user's role
<ROLE>	The role you want to revoke from the user specified in <USERNAME>

## org-users

Show organization users by role, for example:

- Space Manager
- SpaceDeveloper
- SpaceAuditor

### Usage

```
xs org-users <ORG>
```

## Arguments

Table 291: Command Arguments Overview

Argument	Description
<ORG>	Name of the organization to which the users belong

## set-org-role

Assign an organization-specific role to a user. The following roles are available for assignment in an organization:

- OrgManager  
    Invite and manage users, and enable features for a specified organization
- OrgAuditor  
    View logs, reports, and settings for this organization

### Usage

```
xs set-org-role <USERNAME> <ORG> <ROLE>
```

## Arguments

Table 292: Command Arguments Overview

Argument	Description
<USERNAME>	Name of the user to whom you want to assign the specified organization-specific role
<ORG>	The name of the organization for which the user role is to be assigned
<ROLE>	The role you want to assign to the user specified in <USERNAME>

## unset-org-role

Revoke an organization-specific role from a user. The following organization-specific roles can be revoked from a user:

- OrgManager  
Invite and manage users, and enable features for a specified organization
- OrgAuditor  
View logs, reports, and settings for this organization

## Usage

```
xs unset-org-role <USERNAME> <ORG> <ROLE>
```

## Arguments

Table 293: Command Arguments Overview

Argument	Description
<USERNAME>	Name of the user from whom you want to revoke the specified organization-specific role
<ORG>	The name of the organization in which the user role is to be revoked
<ROLE>	The role you want to revoke from the user specified in <USERNAME>

## roles

Display a list of all existing application roles.

## Usage

```
xs roles [<APP>] [-u <USERNAME>] [-p <PASSWORD>] [--stdin]
```

To run this command, you need the privileges of a role-administrator. If the user running the command does not have these privileges, you will be prompted to enter the password of a user with the required privileges. You can specify the name and password of a user with the required privileges in the command itself, for example, with the -u and -p options.

## Arguments

Table 294: Command Arguments Overview

Argument	Description
<APPNAME>	The name of the application whose assigned roles you want to display

## Options

Table 295: Command Options Overview

Option	Description
-u <USERNAME>	The name of a specific user (other than the current session user) in whose account you want to run the command. The default is the current session user.  <b>⚠ Restriction</b> To retrieve and display role-related information, the specified user requires role-administrator privileges in XS advanced.
-p <PASSWORD>	The password for the user specified in <USERNAME>; you can also use the --stdin option to obtain the password.
--stdin	Pipe the password in from standard input (stdin)

## role-collections

Display a list of all existing application role collections.

## Usage

```
xs role-collections [<APP>] [-u <USERNAME>] [-p <PASSWORD>] [--stdin]
```

To run this command, you need the privileges of a role-administrator. If the user running the command does not have these privileges, you will be prompted to enter the password of a user with the required privileges. You can specify the name and password of a user with the required privileges in the command itself, for example, with the -u and -p options.

## Arguments

Table 296: Command Arguments Overview

Argument	Description
<APPNAME>	The name of the application whose assigned role collections you want to display

## Options

Table 297: Command Options Overview

Option	Description
<code>-u &lt;USERNAME&gt;</code>	The name of a specific user (other than the current session user) in whose account you want to run the command. The default is the current session user.  <b>⚠ Restriction</b> To retrieve and display role-related information, the specified user requires role-administrator privileges in XS advanced.
<code>-p &lt;PASSWORD&gt;</code>	The password for the user specified in <code>&lt;USERNAME&gt;</code> ; you can also use the <code>--stdin</code> option to obtain the password.
<code>--stdin</code>	Pipe the password in from standard input ( <code>stdin</code> )

## role-collection

Display details of the specified application role collection.

### Usage

```
xs role-collection [<ROLE_COLLECTION>] [-u <USERNAME>] [-p <PASSWORD>] [--stdin]
```

**⚠ Restriction**

To run this command, you need the privileges of a role-administrator.

If the user running the command does not have these privileges, you will be prompted to enter the password of a user with the required privileges. You can specify the name and password of a user with the required privileges in the command itself, for example, with the `-u` and `-p` options.

### Arguments

Table 298: Command Arguments Overview

Argument	Description
<code>&lt;ROLE_COLLECTION&gt;</code>	The name of the role collection whose details you want to display

## Options

Table 299: Command Options Overview

Option	Description
<code>-u &lt;USERNAME&gt;</code>	The name of a specific user (other than the current session user) in whose account you want to run the command. The default is the current session user.  <b>⚠ Restriction</b> To retrieve and display role-related information, the specified user requires role-administrator privileges in XS advanced.
<code>-p &lt;PASSWORD&gt;</code>	The password for the user specified in <code>&lt;USERNAME&gt;</code> ; you can also use the <code>--stdin</code> option to obtain the password.
<code>--stdin</code>	Pipe the password in from standard input ( <code>stdin</code> )

## create-role-collection

Create a new application role collection.

### Usage

```
xs create-role-collection <ROLE_COLLECTION> [<DESCRIPTION>] [-u <USERNAME>] [-p <PASSWORD>] [--stdin]
```

#### ⚠ Restriction

To run this command, you need the privileges of a role-administrator.

If the user running the command does not have these privileges, you will be prompted to enter the password of a user with the required privileges. You can specify the name and password of a user with the required privileges in the command itself, for example, with the `-u` and `-p` options.

### Arguments

Table 300: Command Arguments Overview

Argument	Description
<code>&lt;ROLE_COLLECTION&gt;</code>	The name of the role collection whose details you want to display
<code>&lt;DESCRIPTION&gt;</code>	An optional description of the role collection you want to create

## Options

Table 301: Command Options Overview

Option	Description
<code>-u &lt;USERNAME&gt;</code>	The name of a specific user (other than the current session user) in whose account you want to run the command. The default is the current session user.  <b>⚠ Restriction</b> To retrieve and display role-related information, the specified user requires role-administrator privileges in XS advanced.
<code>-p &lt;PASSWORD&gt;</code>	The password for the user specified in <code>&lt;USERNAME&gt;</code> ; you can also use the <code>--stdin</code> option to obtain the password.
<code>--stdin</code>	Pipe the password in from standard input ( <code>stdin</code> )

## update-role-collection

Manage the details of the roles assigned to an existing application role collection.

### Usage

Add a role to an existing role collection:

```
xs update-role-collection <ROLE_COLLECTION> --add-role <ROLE> [--app <APP>] [-t <TEMPLATE>] [-u <USERNAME>] [-p <PASSWORD>] [--stdin]
```

Remove an assigned role from an existing role collection:

```
xs update-role-collection <ROLE_COLLECTION> --remove-role <ROLE> [-a <APP>] [-t <TEMPLATE>] [-u <USERNAME>] [-p <PASSWORD>] [--stdin]
```

**⚠ Restriction**

To run this command, you need the privileges of a role-administrator.

If the user running the command does not have these privileges, you will be prompted to enter the password of a user with the required privileges. You can specify the name and password of a user with the required privileges in the command itself, for example, with the `-u` and `-p` options.

### Arguments

Table 302: Command Arguments Overview

Argument	Description
<code>&lt;ROLE_COLLECTION&gt;</code>	The name of the role collection whose details you want to display
<code>&lt;DESCRIPTION&gt;</code>	An optional description of the role collection you want to create

## Options

Table 303: Command Options Overview

Option	Description
<code>-a   --add-role &lt;ROLE&gt;</code>	Assign a new role to the specified role collection
<code>-r &lt;ROLE&gt;</code>	Remove an assigned role from the specified role collection
<code>--app &lt;APPNAME&gt;</code>	The name of the application that provides the role ( <b>optional</b> )
<code>-t &lt;TEMPLATE&gt;</code>	The name of the role template used to instantiate the role ( <b>optional</b> )  → <b>Tip</b> Role templates are defined in the application's security descriptor ( <code>xs-security.json</code> ).
<code>-u &lt;USERNAME&gt;</code>	The name of a specific user (other than the current session user) in whose account you want to run the command. The default is the current session user.  ⚠ <b>Restriction</b> To retrieve and display role-related information, the specified user requires role-administrator privileges in XS advanced.
<code>-p &lt;PASSWORD&gt;</code>	The password for the user specified in <code>&lt;USERNAME&gt;</code> ; you can also use the <code>--stdin</code> option to obtain the password.
<code>--stdin</code>	Pipe the password in from standard input ( <code>stdin</code> )

## assigned-role-collections

Display a list of all the role collections currently assigned to a user.

### Usage

```
xs assigned-role-collections <USER> [-u <USERNAME>] [-p <PASSWORD>] [--stdin]
```

#### ⚠ **Restriction**

To run this command, you need the privileges of a role-administrator.

If the user running the command does not have these privileges, you will be prompted to enter the password of a user with the required privileges. You can specify the name and password of a user with the required privileges in the command itself, for example, with the `-u` and `-p` options.

## Arguments

Table 304: Command Arguments Overview

Argument	Description
<USER>	The name of the user whose assigned role collections you want to display

## Options

Table 305: Command Options Overview

Option	Description
-u <USERNAME>	The name of a specific user (other than the current session user) in whose account you want to run the command. The default is the current session user.
	<b>⚠ Restriction</b> To retrieve and display role-related information, the specified user requires role-administrator privileges in XS advanced.
-p <PASSWORD>	The password for the user specified in <USERNAME>; you can also use the --stdin option to obtain the password.
--stdin	Pipe the password in from standard input (stdin)

## assign-role-collection

Assign a role collections to a user.

### Usage

```
xs assign-role-collection <ROLE_COLLECTION> <USER> [-u <USERNAME>] [-p <PASSWORD>] [--stdin]
```

**⚠ Restriction**

To run this command, you need the privileges of a role-administrator.

If the user running the command does not have these privileges, you will be prompted to enter the password of a user with the required privileges. You can specify the name and password of a user with the required privileges in the command itself, for example, with the -u and -p options.

## Arguments

Table 306: Command Arguments Overview

Argument	Description
<ROLE_COLLECTION>	The name of the role collection that you want to assign

Argument	Description
<USER>	The name of the user to whom you want to assign a role collection

## Options

Table 307: Command Options Overview

Option	Description
-u <USERNAME>	<p>The name of a specific user (other than the current session user) in whose account you want to run the command. The default is the current session user.</p> <p><b>⚠ Restriction</b> To retrieve and display role-related information, the specified user requires role-administrator privileges in XS advanced.</p>
-p <PASSWORD>	The password for the user specified in <USERNAME>; you can also use the --stdin option to obtain the password.
--stdin	Pipe the password in from standard input (stdin)

## unassign-role-collection

Remove an assigned role collections from a user.

### Usage

```
xs unassign-role-collection <ROLE_COLLECTION> <USER> [-u <USERNAME>] [-p <PASSWORD>] [--stdin]
```

#### ⚠ Restriction

To run this command, you need the privileges of a role-administrator.

If the user running the command does not have the required privileges, you will be prompted to enter the password of a user with the required privileges. You can specify the name and password of a user with the required privileges in the command itself, for example, with the -u and -p options.

### Arguments

Table 308: Command Arguments Overview

Argument	Description
<ROLE_COLLECTION>	The name of the assigned role collection that you want to remove
<USER>	The name of the user from whom you want to remove the assigned role collection

## Options

Table 309: Command Options Overview

Option	Description
<code>-u &lt;USERNAME&gt;</code>	The name of a specific user (other than the current session user) in whose account you want to run the command. The default is the current session user.  <b>⚠ Restriction</b> To retrieve and display role-related information, the specified user requires role-administrator privileges in XS advanced.
<code>-p &lt;PASSWORD&gt;</code>	The password for the user specified in <code>&lt;USERNAME&gt;</code> ; you can also use the <code>--stdin</code> option to obtain the password.
<code>--stdin</code>	Pipe the password in from standard input ( <code>stdin</code> )

## Related Information

[XS CLI: User Administration \[page 917\]](#)

[The XS Command-Line Interface Reference \[page 848\]](#)

## 13.13 XS CLI: Administration

Maintain and manage XS application traces and back-up operations for the configuration store.

Table 310: XS Administration Commands Overview

Command	Description
<code>traces</code>	List all available tracing components
<code>enable-trace</code>	Enable tracing components
<code>disable-trace</code>	Disable tracing components
<code>ps</code>	Display the process ID (PID) of all application instances in the current space

### traces

List all available tracing components.

## Usage

```
xs traces
```

### ➔ Tip

Use the list of trace components returned by the `xs traces` command to identify the `<TRACE>` component you want to trace in `xs enable-trace <TRACE>` or `xs disable-trace <TRACE>`.

## enable-trace

Enable tracing for the specified component.

## Usage

```
xs enable-trace <TRACE> [--debug]
```

## Arguments

Table 311: Command Arguments Overview

Argument	Description
<code>&lt;TRACE&gt;</code>	Name of the component for which you want to enable tracing. The trace component can be given as a pattern, for example, "Controller.*".

### ➔ Tip

To display a list of trace components, use the `xs traces` command.

## Options

Table 312: Command Options Overview

Option	Description
<code>--debug</code>	Enable debug-level information

## disable-trace

Disable tracing for a specific component.

## Usage

```
xs disable-trace <TRACE>
```

## Arguments

Table 313: Command Arguments Overview

Argument	Description
<TRACE>	Name of the component for which you want to disable tracing. The trace component can be given as a pattern, for example, "Controller.*".  ➔ <b>Tip</b> To display a list of trace components, use the <code>xs traces</code> command.

## ps

Displays the process ID (PID) of all application instances in the current space.

### Usage

```
xs ps
```

## Related Information

[XS CLI: Administration \[page 929\]](#)

[The XS Command-Line Interface Reference \[page 848\]](#)

## 13.14 XS CLI: Configuration

Set and maintain environment variables.

Table 314: XS Configuration Command Overview

Command	Alias	Description
<code>running-environment-variable-group</code>	<code>rrevg</code>	Retrieve the contents of the running environment variable group
<code>set-running-environment-variable-group</code>	<code>srevg</code>	Pass parameters as JSON to create a running environment variable group

Command	Alias	Description
<code>staging-environment-variable-group</code>	<code>sevg</code>	Retrieve the contents of the staging environment variable group
<code>set-staging-environment-variable-group</code>	<code>ssevg</code>	Pass parameters as JSON to create a staging environment variable group

## running-environment-variable-group

Retrieve the contents of the running environment variable group.

### Usage

```
xs running-environment-variable-group
```

#### ➔ Tip

You can use the alias `revg` in place of the command `running-environment-variable-group`.

## set-running-environment-variable-group

Pass parameters as JSON to create a running environment variable group.

### Usage

```
xs set-running-environment-variable-group
"{"name":"value", "name":"value"}"
```

#### ➔ Tip

You can use the alias `srevg` in place of the command `set-running-environment-variable-group`.

### Arguments

Table 315: Command Arguments Overview

Argument	Description
<code>&lt;ENVIRONMENT-GROUP&gt;</code>	The environment variable group as JSON parameters

## staging-environment-variable-group

Retrieve the contents of the staging environment variable group.

### Usage

```
xs running-environment-variable-group
```

#### ➔ Tip

You can use the alias `sevg` in place of the command `staging-environment-variable-group`.

## set-staging-environment-variable-group

Pass parameters as JSON to create a staging environment variable group.

### Usage

```
xs set-staging-environment-variable-group  
"{"name":"value","name":"value"}"
```

#### ➔ Tip

You can use the alias `ssevg` in place of the command `set-staging-environment-variable-group`.

### Arguments

Table 316: Command Arguments Overview

Argument	Description
<code>&lt;ENVIRONMENT-GROUP&gt;</code>	The environment variable group as JSON parameters

## Related Information

[XS CLI: Configuration \[page 931\]](#)

[The XS Command-Line Interface Reference \[page 848\]](#)

## 13.15 XS CLI: Blob Store

Manage and maintain the contents of the blob store.

Table 317: XS Blob Store Command Overview

Command	Description
<code>blob-store-info</code>	Show information about the blob store
<code>blob-set-list</code>	List all blob sets in the blob store
<code>blob-list</code>	List all blobs in the blob set
<code>blob-set-download</code>	Download the content of a blob set as a zip file
<code>blob-store-gc</code>	Start a garbage collection of the blob store

### **blob-store-info**

Show information about the blob store.

#### **Usage**

```
xs blob-store-info
```

### **blob-set-list**

List all blob sets in the blob store.

#### **Usage**

```
xs blob-set-list
```

### **blob-list**

List all blobs in the blob set.

#### **Usage**

```
xs blob-list [--show-hash] <set-id>
```

## Arguments

Table 318: Command Arguments Overview

Argument	Description
<set-id>	The ID of the blob set for which you want to display blob information

## Options

Table 319: Command Options Overview

Option	Description
--show-hash <ID>	Display the hash for each blob

## blob-set-download

Download the content of a blob set as a zip archive.

### Usage

```
xs blob-set-download <set-id> <zip-file>
```

## Arguments

Table 320: Command Arguments Overview

Argument	Description
<set-id>	The ID of the blob set that you want to download
<zip-file>	The name of the Zip archive in which to store and download the blob content

## blob-store-gc

Start a garbage collection of the blob store.

### Usage

```
xs blob-store-gc
```

## Related Information

[XS CLI: Blob Store \[page 934\]](#)

[The XS Command-Line Interface Reference \[page 848\]](#)

## 13.16 XS CLI: Advanced

A list of the tools available for advanced administration tasks in SAP HANA XS advanced model.

Table 321: XS Commands Overview

Command	Description
<code>curl</code>	Executes a raw request, content-type set to application/json by default
<code>oauth-token</code>	Retrieve and display the OAuth token for the current session

### curl

Execute a raw request; the content-type is set to “application/json” by default.

#### Usage

```
xs curl <PATH> [-X <METHOD>] [-d <DATA>]
```

#### Arguments

Table 322: Command Arguments Overview

Argument	Description
<code>&lt;COMMAND&gt;</code>	Name of the command for which you want to display usage help

#### Options

Table 323: Command Options Overview

Argument	Description
<code>-X &lt;METHOD&gt;</code>	The name of the HTTP method to use, for example: GET, POST, PUT, DELETE, [...]
<code>-d &lt;DATA&gt;</code>	The HTTP data to include in the request body

### oauth-token

Retrieve and display the OAuth token for the current session.

#### Usage

```
xs oauth-token [--decode]
```

## Options

Table 324: Command Options Overview

Argument	Description
--decode <METHOD>	Try to decode the OAuth token

## Related Information

[XS CLI: Advanced \[page 936\]](#)

[The XS Command-Line Interface Reference \[page 848\]](#)

## 13.17 XS CLI: Other Commands

Display details of the installed SAP HANA XS version, the command-line interface (CLI), and system information.

Table 325: XS Commands Overview

Command	Alias	Description
<a href="#">version</a>		Show server version information
<a href="#">help</a>	h	Show a list of all available xs command-line tools as well as a description of each tool
<a href="#">system-info</a>		Show information about the system infrastructure
<a href="#">service-urls</a>		Display a list of all registered service URLs

### version

Show information about the installed server version.

#### Usage

```
xs version
```

### help

Show a list of all available xs command-line tools as well as a description of each tool

## Usage

```
xs h[elp]
```

```
xs h[elp] [<COMMAND>]
```

## Arguments

Table 326: Command Arguments Overview

Argument	Description
<COMMAND>	Name of the command for which you want to display usage help

## Examples

To display detailed information for the enable-trace command:

```
xs help enable-trace
```

## system-info

Show information about the system infrastructure.

## Usage

```
xs system-info
```

## service-urls

Display a list of all registered service URLs.

## Usage

```
xs service-urls
```

## Related Information

[XS CLI: Other Commands \[page 937\]](#)

[The XS Command-Line Interface Reference \[page 848\]](#)

## 13.18 XS CLI: Plug-ins

A list of additional commands that have been implemented as plug-ins to extend the base XS CLI client, for example, to install archives and deploy multi-target applications (MTA) to the SAP HANA XS advanced runtime environment.

Table 327: XS Commands Overview

Command	Alias	Description
<code>deploy</code>		Deploy a new multi-target application (MTA) to an SAP HANA instance or synchronize changes to an existing MTA
<code>bg-deploy</code>		Deploy a multi-target application using “blue-green” (zero-downtime) deployment
<code>undeploy</code>		Undeploy a multi-target application (MTA)
<code>install</code>	<code>ins</code>	Install a software component version (SCV) or a product
<code>uninstall</code>	<code>unins</code>	Uninstall a software component version or a product version.
<code>display-installation-logs</code>	<code>dil</code>	Display the logs generated by the Product Installer
<code>display-installation-history</code>	<code>dih</code>	Display the history of installation and uninstallation processes.
<code>list-components</code>	<code>lc</code>	List all installed software component versions
<code>list-products</code>	<code>lp</code>	List all installed product versions or details of a specific product version
<code>mta</code>		Display information about a deployed multi-target application (MTA)
<code>mtas</code>		List all deployed multi-target applications (MTA)
<code>mta-ops</code>		List all active operations for multi-target applications
<code>plugins</code>		List all commands that are provided as plug-ins to the CLI client
<code>download-mta-op-logs</code>	<code>dmol</code>	Download the log files for one or more operations concerning multi-target applications
<code>purge-mta-config</code>		Purge all configuration entries and subscriptions which are no longer valid

Command	Alias	Description
		<p><b>⚠ Restriction</b></p> <p>A target platform is an alias for a user space in an organization. Administrator permissions are required to perform actions on "target platforms".</p>
<code>deploy-target</code>		Display information about a target platform for deployment of multi-target apps (MTA).
<code>deploy-targets</code>		List all target platforms for the deployment of multi-target apps (MTA)
<code>create-deploy-target</code>	cdt	Create a target platform for deployment of multi-target apps (MTA)
<code>update-deploy-target</code>	udt	Update a target platform for deployment of multi-target apps (MTA)
<code>delete-deploy-target</code>	ddt	Delete a target platform for deployment of multi-target apps (MTA)

## deploy

Deploy a new multi-target application (MTA) to an SAP HANA instance or synchronize changes to an existing MTA.

### Usage

Deploy a new multi-target application or synchronize changes to an existing one:

```
xs deploy <MTA_ARCHIVE> [--deploy-target <DEPLOY_TARGET>]
[-s <SCHEMA_VERSION>] [-e <EXT_DESCRIPTOR_1>[,<EXT_DESCRIPTOR_2>]]
[-u <URL>] [-t <TIMEOUT>] [-v <VERSION_RULE>]
[--no-start] [--use-namespaces] [--no-namespaces-for-services]
[--delete-services] [--delete-service-keys] [--delete-service-brokers]
[--keep-files] [--no-restart-subscribed-apps] [--git-uri URI] [--git-ref NAME]
[--git-skip-ssl-validation] [--do-not-fail-on-missing-permissions]
```

Interact with an active MTA deploy operation, for example, by performing an action:

```
xs deploy [-i <OPERATION_ID>] [-a <ACTION>]
```

Deploy an MTA from a Git repository:

```
xs deploy mtas/jobscheduler/ --git-uri https://github.acme.com/myapps/
jobscheduler.git --git-skip-ssl-validation --git-ref OQtests
```

In the following example, the dot after the `deploy` command (“`xs deploy .`”) signifies the location of the content in the Git repository (that is, the root of the specified repository); the option `--git-uri` specifies the URL of the repository where the content to be downloaded (and deployed) is stored.

```
xs deploy . --git-uri https://github.acme.com/myapps/jobscheduler.git --git-skip-ssl-validation --git-ref unzipped_mtar
```

## Arguments

Table 328: Command Arguments Overview

Argument	Description
<code>&lt;MTA_ARCHIVE&gt;</code>	The path to (and name of) the archive or the directory containing the multi-target application to deploy; the application archive must have the format (and file extension) <code>mtar</code> , for example, <code>MTApp1.mtar</code> .

## Options

Table 329: Command Options Overview

Option	Description
<code>--deploy-target &lt;DEPLOY_TARGET&gt;</code>	<code>--deploy-target &lt;DEPLOY_TARGET&gt;</code> Specify the target platform where the application must be deployed (by default: “ <code>&lt;org&gt; &lt;space&gt;</code> ”)
<code>-s --schema-version &lt;SCHEMA_VERSION&gt;</code>	Specify the major component of the schema version used by the target platform (by default: “2”)
<code>-e &lt;EXT_DESCRIPTOR_1&gt;[,&lt;EXT_DESCRIPTOR_2&gt;]</code>	Define one or more extensions to the deployment descriptors; multiple extension descriptors must be separated by commas.
<code>-u &lt;URL&gt;</code>	Specify the URL for the deployment-service end-point that is to be used for the deployment operation
<code>-t &lt;TIMEOUT&gt;</code>	Specify the maximum amount of time (in seconds) that the deployment service must wait for before starting the deployed application
<code>-v &lt;VERSION_RULE&gt;</code>	Specify the rule to apply to determine how the application version number is used to trigger an application-update deployment operation, for example: “ <code>HIGHER</code> ”, “ <code>SAME_HIGHER</code> ”, or “ <code>ALL</code> ”.
<code>-i &lt;OPERATION_ID&gt;</code>	Specify the ID of the deploy operation that you want to perform an action on
<code>-a &lt;ACTION&gt;</code>	Specify the action to perform on the deploy operation, for example, “ <code>abort</code> ”, “ <code>retry</code> ”, or “ <code>monitor</code> ”, or “ <code>resume</code> ”
<code>-f</code>	Force deploy without requiring any confirmation for aborting any conflicting processes

Option	Description
--no-start	Do not start application after deployment
--use-namespaces	Use namespaces in application (and service) names during application deployment
--no-namespaces-for-services	Do not use namespaces in service names
--delete-services	Recreate changed services and delete discontinued services
--delete-service-keys	Delete the existing service keys and apply the new ones
--delete-service-brokers	Delete discontinued service brokers
--keep-files	Keep the files used for deployment
--no-restart-subscribed-apps	Do not restart subscribed applications that are updated during the deployment
--git-uri <URL>	<p>The Universal Resource Locator (URL) or name of a Git repository in the XS advanced embedded GIT service</p> <p><b>i Note</b></p> <p>If the Git repository is provided in the XS advanced integrated git service, the git-uri parameter can only be used to specify the name of the Gerrit project (repository name).</p> <p>The content in the specified Git repository should be built and ready for deployment, either as an MTA archive (MTAR) or a suitably structured directory.</p>
--git-ref <NAME>	The name of a Git repository branch, tag, or reference
--git-skip-ssl-validation	Skip the SSL verification step when establishing an HTTPS connection to the GIT repository
--do-not-fail-on-missing-permissions	Perform the deployment, even if required administrator permission are missing for some operations (for example, the creation of service brokers).

## bg-deploy

Deploy a new multi-target application (MTA) to an SAP HANA instance using “blue-green” (zero-downtime) deployment.

“Blue-green” deployment is a release technique that helps to reduce application downtime and the resulting risk by running two identical target deployment environments called “blue” and “green”. Only one of the two

target environments is “live” at any point in time and it is much easier to roll back to a previous version after a failed (or undesired) deployment.

## Usage

Deploy a new multi-target application using “blue-green” deployment:

```
xs bg-deploy <MTA_ARCHIVE> [--deploy-target <DEPLOY_TARGET>]
[-s <SCHEMA_VERSION>] [-e <EXT_DESCRIPTOR_1>[,<EXT_DESCRIPTOR_2>]]
[-u <URL>] [-t <TIMEOUT>] [-v <VERSION_RULE>]
[--no-start] [--use-namespaces] [--no-namespaces-for-services]
[--delete-services] [--delete-service-keys] [--delete-service-brokers]
[--keep-files] [--no-restart-subscribed-apps] [--git-uri URI] [--git-ref NAME]
[--git-skip-ssl-validation] [--no-confirm] [--do-not-fail-on-missing-permissions]
```

Interact with an active MTA deploy operation, for example, by performing an action:

```
xs bg-deploy [-i <OPERATION_ID>] [-a <ACTION>]
```

## Arguments

Table 330: Command Arguments Overview

Argument	Description
<MTA_ARCHIVE>	The path to (and name of) the archive or the path to the directory containing the multi-target application to deploy. The application archive must have the format (and file extension) <code>mtar</code> , for example, <code>MTApp1.mtar</code> ; the specified directory can be specified as a path (for example, <code>myApp/</code> or <code>.</code> (current directory)).

## Options

Table 331: Command Options Overview

Option	Description
--deploy-target <DEPLOY_TARGET>	Specify the target platform where the application must be deployed, by default: “<ORG> <SPACE>”
-s --schema-version <SCHEMA_VERSION>	Specify the major component of the schema version used by the target platform (by default: “2”)
-e <EXT_DESCRIPTOR_1>[,<EXT_DESCRIPTOR_2>]	Define one or more extensions to the deployment descriptors; multiple extension descriptors must be separated by commas.
-u <URL>	Specify the URL for the deployment-service end-point that is to be used for the deployment operation
-t <TIMEOUT>	Specify the maximum amount of time (in seconds) that the deployment service must wait for before starting the deployed application
-v <VERSION_RULE>	Specify the rule to apply to determine how the application version number is used to trigger an application-update deployment operation, for example: “HIGHER”, “SAME_HIGHER”, or “ALL”.

Option	Description
<code>-i &lt;OPERATION_ID&gt;</code>	Specify the ID of the deploy operation that you want to perform an action on
<code>-a &lt;ACTION&gt;</code>	Specify the action to perform on the deploy operation, for example, "abort", "retry", or "monitor", or "resume"
<code>-f</code>	Force deploy without requiring any confirmation for aborting any conflicting processes
<code>--no-start</code>	Do not start application after deployment
<code>--use-namespaces</code>	Use namespaces in application (and service) names during application deployment
<code>--no-namespaces-for-services</code>	Do not use namespaces in service names
<code>--delete-services</code>	Recreate changed services and delete discontinued services
<code>--delete-service-keys</code>	Delete the existing service keys and apply the new ones
<code>--delete-service-brokers</code>	Delete discontinued service brokers
<code>--keep-files</code>	Keep the files used for deployment
<code>--no-restart-subscribed-apps</code>	Do not restart subscribed applications that are updated during the deployment
<code>--git-uri &lt;URI&gt;</code>	The Universal Resource Indicator (URI) or name of a Git repository in the XS advanced embedded GIT service
<code>--git-ref &lt;NAME&gt;</code>	The name of a Git repository branch, tag, or reference
<code>--git-skip-ssl-validation</code>	Skip the SSL verification step when establishing an HTTPS connection to the GIT repository
<code>--no-confirm</code>	Do not require confirmation for the deletion of the previously deployed MTA applications
<code>--do-not-fail-on-missing-permissions</code>	Perform the deployment, even if required administrator permission are missing for some operations (for example, the creation of service brokers).

## undeploy

Undeploy a multi-target application (MTA), or interact with an undeploy MTA operation.

## Usage

Undeploy an MTA.

```
xs undeploy <MTA_ID> [--deploy-target <DEPLOY_TARGET>]
[-s <SCHEMA_VERSION>] [-u <URL>] [-f]
[--delete-services] [--delete-service-brokers] [--no-restart-subscribed-apps]
[--do-not-fail-on-missing-permissions]
```

Interact with an undeploy MTA operation, for example, by performing an action.

```
xs undeploy [-i <UNDEPLOY_ID>] [-a <ACTION>]
```

## Arguments

Table 332: Command Arguments Overview

Argument	Description
<MTA_ID>	The ID of the MTA you want to undeploy

## Options

Table 333: Command Options Overview

Option	Description
--deploy-target <DEPLOY_TARGET>	Specify the target platform from which the application must be undeployed
-s --schema-version <SCHEMA_VERSION>	Specify the major component of the schema version used by the target platform (by default: "2")
-u <URL>	Specify the URL for the service end-point that is to be used for the undeployment operation
-i <OPERATION_ID>	Specify the ID of the undeploy operation that you want to perform an action on
-a <ACTION>	Specify the action to perform on the undeploy operation, for example, "abort", "retry", or "monitor"
-f	Force completion of the undeploy operation without any system prompt or confirmation
--delete-services	Delete any related services
--delete-service-brokers	Delete discontinued service brokers
--no-restart-subscribed-apps	Do not restart subscribed applications that are updated during the deployment
--do-not-fail-on-missing-permissions	Perform the deployment, even if required administrator permission are missing for some operations (for example, the creation of service brokers).

## install

Install or update a software component version (SCV) or a product.

### Usage

```
xs install <ARCHIVE> [-p <TARGET_PLATFORM>] [-pv | --PRODUCT_VERSION] [-scv | --SOFTWARE_COMPONENT_VERSION] [-t <TIMEOUT>] [-e <EXT_DESCRIPTOR_1>[,<EXT_DESCRIPTOR_2>]] [-o <VERSION_OPTION_1>[,<VERSION_OPTION_2>]] [-i | --INSTANCES <INSTANCE_1[,INSTANCE_2>]] [--delete-services] [--delete-service-brokers] [--no-start] [--ignore-lock]
```

#### ➔ Tip

You can use the alias `ins` in place of the `install` command, for example, `xs ins`.

The `install` command detects whether the software component is already installed and executes either an installation or update operation.

### Arguments

Table 334: Command Arguments Overview

Argument	Description
<code>&lt;ARCHIVE&gt;</code>	The path to (and name of) the archive containing the product or software component (SCV) to install, update, or downgrade

### Options

Table 335: Command Options Overview

Option	Description
<code>-p &lt;TARGET_PLATFORM&gt;</code>	Specify the target platform where the product or software component must be installed. If not specified explicitly, a target platform is created implicitly as ' <code>&lt;ORG&gt; &lt;SPACE&gt;</code> '.
<code>-pv   --PRODUCT_VERSION</code>	Install a product
<code>-scv   --SOFTWARE_COMPONENT_VERSION</code>	Install a software component
<code>-e &lt;EXT_DESCRIPTOR_1&gt;[,&lt;EXT_DESCRIPTOR_2&gt;]</code>	Define one or more extensions to the installation/deployment descriptors; multiple extension descriptors must be separated by commas.
<code>-t &lt;TIMEOUT&gt;</code>	Specify the maximum amount of time (in seconds) that the installation service must wait for the installation operation to complete

Option	Description
<code>-o &lt;VERSION_OPTION_1&gt;[,&lt;VERSION_OPTION_2&gt;]</code>	Specify options which can be used to <b>override</b> the default behavior of the <code>install</code> command. The following options are available: <ul style="list-style-type: none"> <li>• <code>ALLOW_SC_SAME_VERSION</code> Re-installs the same version of the software component</li> <li>• <code>ALLOW_SC_DOWNGRADE</code> Allows downgrade of the software component</li> <li>• <code>ALLOW_KEEP_SC_NEWER_VERSION</code> Skips installation of a software component if a newer version is already installed on the target system</li> <li>• <code>ALLOW_PV_DOWNGRADE</code> Allows downgrade of the installed product version</li> </ul>
<code>-i, --INSTANCES &lt;INSTANCE_1&gt;[,&lt;INSTANCE_2&gt;]</code>	By default all instances are installed; a comma-separated list of instances can be specified to limit the number of instances installed
<code>--delete-services</code>	Recreate changed services and/or delete discontinued services
<code>--delete-service-brokers</code>	Delete discontinued service brokers
<code>--no-start</code>	Do not start applications that are updated during the installation
<code>--ignore-lock</code>	Force installation even if the space targeted for installation is locked

## uninstall

Remove a software component version or a product version from the target system.

### Usage

```
xs uninstall <NAME> [<VENDOR>] [-pv | --PRODUCT_VERSION] [-scv | --SOFTWARE_COMPONENT_VERSION] [-f] [--ignore-scv-reuse] [--delete-services] [--delete-service-brokers] [--ignore-lock]
```

#### ➔ Tip

You can use the alias `unins` in place of the `uninstall` command, for example, `xs unins`.

### Arguments

Table 336: Command Arguments Overview

Argument	Description
<code>&lt;NAME&gt;</code>	The name of an installed product version (PV) or Software Component Version (SCV)

Argument	Description
[<VENDOR>]	The name of the vendor of the specified product or software component version; <b>optional:</b> only needed if the same <b>product</b> name is used by different vendors

## Options

Table 337: Command Options Overview

Option	Description
-pv   --PRODUCT_VERSION	Remove the specified product; if the component specified in <NAME> is not a product, the removal operation fails.
-scv   --SOFTWARE_COMPONENT_VERSION	Remove the specified software component version; if the component specified in <NAME> is not an SCV, the removal operation fails
--ignore-scv-reuse	Remove the specified software component even if it is used in other products.  <b>⚠ Restriction</b>  Only for use software components. If you use this option to uninstall a product version (PV), any software components used by other products are ignored by the uninstall operation and left in place.
-f	Remove the specified product or software component without any system prompts or confirmation
-i, --INSTANCES <INSTANCE_1>[,<INSTANCE_2>]	By default all instances are installed; a comma-separated list of instances can be used to limit the number of instances that can be installed
--delete-services	Recreate changed services and/or delete discontinued services
--delete-service-brokers	Delete discontinued service brokers
--ignore-lock	Force removal of the product or software component even if the target space is locked

## display-installation-logs

Display logs generated by the Product Installer application.

### Usage

```
xs display-installation-logs <PROCESS_ID>
```

## ➔ Tip

You can use the alias `dil` in place of the `display-installation-logs` command.

## Arguments

Table 338: Command Arguments Overview

Argument	Description
<code>&lt;PROCESS_ID&gt;</code>	The ID of the product installer process whose log contents you want to display

## Options

Table 339: Command Options Overview

Option	Description
<code>--unins_scv</code>	Show the logs written during the removal of a software component version (SCV)
<code>--unins_pv</code>	Show the logs written during the removal of a product version (PV)
<code>--scv</code>	Show the logs written during the installation of a software component version (SCV)
<code>--pv</code>	Show the logs written during the installation of a product version (PV)

## display-installation-history

Display the history of installation and uninstallation processes. By default, the last ten processes are displayed.

## Usage

```
xs display-installation-history [--all] [--last <NUM>]
```

## ➔ Tip

You can use the alias `dih` in place of the `display-installation-history` command.

## Options

Table 340: Command Options Overview

Option	Description
<code>--all</code>	Display all installation and uninstallation processes

Option	Description
--last <NUM>	Display the last <NUM> installation or uninstallation processes, where <NUM> can be any number.

## list-components

List all installed software component versions.

### Usage

```
xs list-components
```

#### ➔ Tip

You can use the alias `lc` in place of the `list-components` command.

## list-products

List all installed products or details about a specific product.

### Usage

```
xs list-products <NAME> <VENDOR>
```

#### ➔ Tip

You can use the alias `lp` in place of the `list-products` command.

### Arguments

Table 341: Command Arguments Overview

Argument	Description
<NAME>	The name of an installed product name; <b>optional</b>
<VENDOR>	The name of the vendor of the specified product; <b>optional</b> : only needed when you specify a product <NAME> in the command

## mta

Display information about a multi-target application (MTA). The information displayed includes the requested state, the number of instances, information about allocated memory and disk space, as well as details regarding the bound service (and service plan).

### Usage

```
xs mta MTA_ID [--deploy-target <DEPLOY_TARGET>]
[-s <SCHEMA_VERSION>] [-u <URL>]
```

### Arguments

Table 342: Command Arguments Overview

Argument	Description
<MTA_ID>	The ID of the MTA whose details you want to display

### Options

Table 343: Command Options Overview

Option	Description
--deploy-target <DEPLOY_TARGET>	Specify the target platform for the MTA whose details you want to display
-s --schema-version <SCHEMA_VERSION>	Specify the major component of the schema version used by the target platform (by default: "2")
-u <URL>	Specify the URL for the deployment-service end-point to use to obtain details of the selected MTA

## mtas

Display information about all available multi-target applications (MTA).

### Usage

```
xs mtas [--deploy-target <DEPLOY_TARGET>] [-s <SCHEMA_VERSION>]
[-u <URL>]
```

## Options

Table 344: Command Options Overview

Option	Description
--deploy-target <DEPLOY_TARGET>	Specify the target platform for the MTA whose details you want to display
-s --schema-version <SCHEMA_VERSION>	Specify the major component of the schema version used by the target platform (by default: "2")
-u <URL>	Specify the URL for the deployment-service end-point to use to obtain details of the selected MTA

## mta-ops

Display information about all **active** operations for multi-target applications (MTA). The information includes the ID, type, status, the time the MTA-related operation started, as well as the name of the user that started the operation.

### Usage

```
xs mta-ops [--deploy-target <DEPLOY_TARGET>] [-s <SCHEMA_VERSION>] [-u <URL>] [--last <NUM>] [--all]
```

## Options

Table 345: Command Options Overview

Option	Description
--deploy-target <DEPLOY_TARGET>	Specify the target platform for the MTAs, whose active operations you want to list
-u <URL>	Specify the URL for the deployment-service end-point to use to obtain details of the selected MTA operations
-s, --schema-version <SCHEMA_VERSION>	Specify the major component of the schema version used by the target platform (default=2)
--last <NUM>	List the last <NUM> active MTA operations
--all	List all active MTA operations

## plugins

List all commands that are provided as plugins to the CLI client and, if available, show the plugin name and version, too.

## Usage

```
xs plugins
```

This produces output similar to the following example:

Output Code			
Plugin name	Version	Command name	Command help
MtaPlugin	1.31.0	deploy	Deploy a new...
MtaPlugin	1.31.0	bg-deploy	Deploy a multi...
MtaPlugin	1.31.0	undeploy	Undeploy a...
MtaPlugin	1.31.0	mta-ops	List all active...
MtaPlugin	1.31.0	download-mta-op-logs	Download logs ...
MtaPlugin	1.31.0	mtas	List all apps...
MtaPlugin	1.31.0	mta	Display info...
MtaPlugin	1.31.0	deploy-targets	List target....
MtaPlugin	1.31.0	deploy-target	Display info...
MtaPlugin	1.31.0	create-deploy-target	Create a...
MtaPlugin	1.31.0	update-deploy-target	Update a...
MtaPlugin	1.31.0	delete-deploy-target	Delete a...
MtaPlugin	1.31.0	purge-mta-config	Purge no longer
valid...			
Product-Installer	1.11.1	install	Install or update...
Product-Installer	1.11.1	list-components	List all...
Product-Installer	1.11.1	list-products	List all...
Product-Installer	1.11.1	display-installation-logs	Display Product...
Product-Installer	1.11.1	uninstall	Uninstall...

## download-mta-op-logs

Download the log files for one or more operations concerning multi-target applications.

### Usage

```
xs download-mta-op-logs [--deploy-target <DEPLOY_TARGET>]
[-s <SCHEMA_VERSION>] [-u <URL>]
[-i <OPERATION_ID>] [-d <DIRECTORY>]
```

#### Tip

You can use the alias `dml` in place of the `download-mta-op-logs` command.

### Options

Table 346: Command Options Overview

Option	Description
--deploy-target <DEPLOY_TARGET>	Specify the target deployment platform for the MTAs, whose active operations you want to list

Option	Description
<code>-s --schema-version &lt;SCHEMA_VERSION&gt;</code>	Specify the major component of the schema version used by the target platform (by default: "2")
<code>-u &lt;URL&gt;</code>	Specify the URL for the deployment-service end-point to use to obtain details of the selected MTA operations
<code>-i &lt;OPERATION_ID&gt;</code>	Specify the identity (ID) of the MTA operation whose logs you want to download
<code>-d &lt;DIRECTORY&gt;</code>	Specify the path to the location where you want to save the downloaded MTA operation logs; by default, the location is <code>./mta-op-&lt;OPERATION_ID&gt;/</code>

## purge-mta-config

Purge all configuration entries and subscriptions which are no longer valid.

### Usage

```
xs purge-mta-config [--deploy-target <DEPLOY_TARGET>]
[-s <SCHEMA_VERSION>] [-u <URL>]
```

Invalid configuration entries are often produced when the application that is providing configuration entries is deleted by the user without using the deploy-service, for example, the `xs delete` command. In this case, the configuration remains in the deploy-service database even though the corresponding application is no longer available. This could lead to a failure during subsequent attempts to resolve the configuration entries.

### Options

Table 347: Command Options Overview

Option	Description
<code>--deploy-target &lt;DEPLOY_TARGET&gt;</code>	The name of the target platform where the MTA configurations are to be purged. If not specified explicitly, the target platform is assumed to be " <code>&lt;org&gt;&lt;space&gt;</code> ".
<code>-s, --schema-version &lt;SCHEMA_VERSION&gt;</code>	The major component of the schema version used by the deploy target (by default "2")
<code>-u &lt;URL&gt;</code>	The URL of the deploy service

## deploy-target

Display information about a target platform for deployment of multi-target applications (MTA). A “target platform” is an alias for an organizational space (with optional configuration parameters) to which you want to deploy an MTA.

### ⚠ Restriction

Administrator permissions are required to perform this action.

### Usage

```
xs deploy-target <DEPLOY_TARGET> [-u <URL>] [-s <SCHEMA_VERSION>]
```

### Arguments

Table 348: Command Arguments Overview

Argument	Description
<DEPLOY_TARGET>	The name of the target platform where the MTA are to be deployed. If not specified explicitly, a target platform is created implicitly as “<org><space>”.

### Options

Table 349: Command Options Overview

Option	Description
-u <URL>	Specify the URL for the deployment-service end-point to use to obtain details of the selected target platform
-s, --schema-version <SCHEMA_VERSION>	The major component of the schema version used by the deploy target (by default “2”)

## deploy-targets

Display a list of all available target platforms for deployment of multi-target applications (MTA). A “deploy target” is an alias for an organizational space (with optional configuration parameters) to which you want to deploy an MTA.

### ⚠ Restriction

Administrator permissions are required to perform this action.

### Usage

```
xs deploy-targets [-u <URL>] [-s <SCHEMA_VERSION>]
```

## Options

Table 350: Command Options Overview

Option	Description
<code>-u &lt;URL&gt;</code>	Specify the URL for the deployment-service end-point to use to obtain details of the selected target platform
<code>-s,--schema-version &lt;SCHEMA_VERSION&gt;</code>	The major component of the schema version used by the deploy target (by default "2")

## create-deploy-target

Create a target platform for deployment of multi-target apps (MTA). A “target platform” is an alias for an organizational space (with optional configuration parameters) to which you want to deploy an MTA.

### ⚠ Restriction

Administrator permissions are required to perform this action.

### Usage

```
xs create-deploy-target <DEPLOY_TARGET_FILE> [-u <URL>] [-s <SCHEMA_VERSION>]
```

### ➔ Tip

You can use the alias `cdt` in place of the `create-deploy-target` command.

Table 351: Command Arguments Overview

Argument	Description
<code>&lt;DEPLOY_TARGET_FILE&gt;</code>	The path to (and name of) the JSON file containing details of the target deployment platform to create

## Options

Table 352: Command Options Overview

Option	Description
<code>-u &lt;URL&gt;</code>	Specify the URL for the deployment-service end-point to use when creating the new target platform
<code>-s,--schema-version &lt;SCHEMA_VERSION&gt;</code>	The major component of the schema version used by the deploy target (by default "2")

## update-deploy-target

Update a target platform for deployment of multi-target applications (MTA).

### ⚠ Restriction

Administrator permissions are required to perform this action.

### Usage

```
xs update-deploy-target <DEPLOY_TARGET_NAME> [-p <DEPLOY_TARGET_FILE>]
[-u <URL>] [-s <SCHEMA_VERSION>]
```

### ➔ Tip

You can use the alias `udt` in place of the `update-deploy-target` command.

### Arguments

Table 353: Command Arguments Overview

Argument	Description
<code>&lt;DEPLOY_TARGET_NAME&gt;</code>	The name of the target deployment platform that you want to update

### Options

Table 354: Command Options Overview

Option	Description
<code>-p &lt;DEPLOY_TARGET_FILE&gt;</code>	Specify the path to (and name of) the JSON file containing details of the target deployment platform to update
<code>-u &lt;URL&gt;</code>	Specify the URL for the deployment-service end-point to use when updating the new target deployment platform
<code>-s,--schema-version &lt;SCHEMA_VERSION&gt;</code>	The major component of the schema version used by the deployment target (by default "2")

## delete-deploy-target

Remove a target platform for deployment of multi-target applications (MTA).

### ⚠ Restriction

Administrator permissions are required to perform this action.

## Usage

```
xs delete-deploy-target <DEPLOY_TARGET_NAME> [-u <URL>] [-s <SCHEMA_VERSION>]
```

### ➔ Tip

You can use the alias `ddt` in place of the `delete-deploy-target` command.

## Arguments

Table 355: Command Arguments Overview

Argument	Description
<code>&lt;DEPLOY_TARGET_NAME&gt;</code>	The name of the target deployment platform that you want to delete

## Options

Table 356: Command Options Overview

Option	Description
<code>-u &lt;URL&gt;</code>	Specify the URL for the deployment-service end-point to use when deleting the selected target platform
<code>-s,--schema-version &lt;SCHEMA_VERSION&gt;</code>	The major component of the schema version used by the deploy target (by default "2")

## Related Information

[The XS Command-Line Interface Reference \[page 848\]](#)

# 14 SAP Web IDE for SAP HANA Reference

Reference information about SAP Web IDE for SAP HANA.

The following section describes the various development tools provided by SAP Web IDE for SAP HANA and the main tasks required for developing SAP HANA based applications.

## [Product Overview \[page 959\]](#)

SAP Web IDE for SAP HANA is a browser-based integrated development environment (IDE) for the development of SAP HANA-based applications comprised of web-based or mobile UIs, business logic, and extensive SAP HANA data models.

## [Post-Installation Administration Tasks \[page 962\]](#)

Perform these administration tasks after installing SAP Web IDE, or later as required.

## [Getting Started \[page 966\]](#)

Start working with SAP Web IDE for SAP HANA by getting a user account, logging on, and getting familiar with the environment.

## [Developing Multi-Target Applications \[page 983\]](#)

Multi-target, or multi-tier applications are comprised of multiple software modules representing the data, business logic and UI tiers. These modules are created with different technologies and are deployed to different target platforms, yet share the same development lifecycle.

## [Customizing Your Project \[page 1093\]](#)

Customize the developer experience for your SAP Web IDE project.

## [Using Code Editors \[page 1096\]](#)

SAP Web IDE provides customizable editors in which you edit the code for your applications.

## [Using Source Control \(Git\) \[page 1113\]](#)

The SAP Web IDE includes the Git source control system, letting you connect and interact with remote Git repositories.

## [Troubleshooting \[page 1133\]](#)

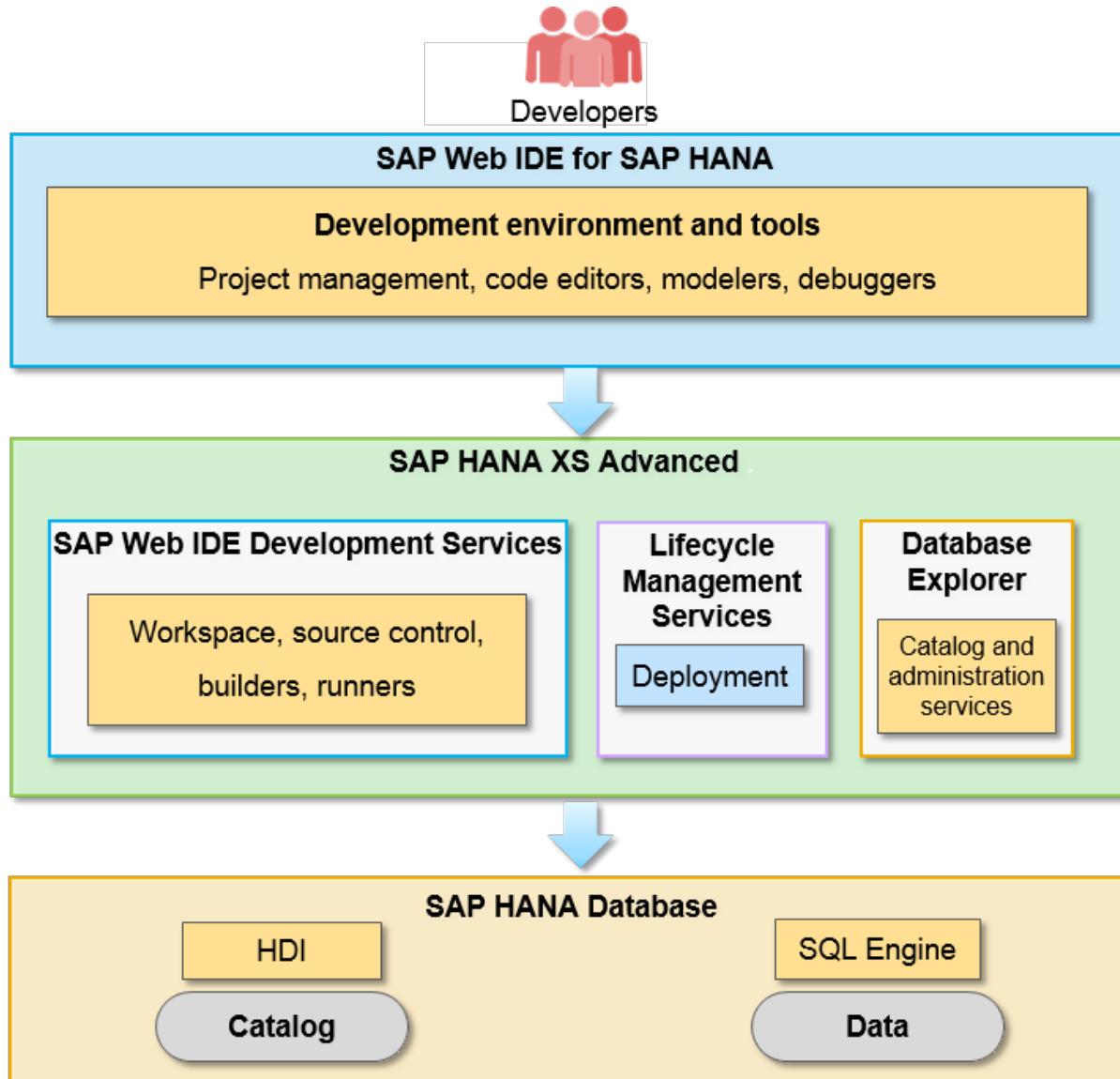
Here are some common troubleshooting issues in SAP Web IDE.

## 14.1 Product Overview

SAP Web IDE for SAP HANA is a browser-based integrated development environment (IDE) for the development of SAP HANA-based applications comprised of web-based or mobile UIs, business logic, and extensive SAP HANA data models.

SAP Web IDE supports developers who use SAP HANA Extended Application Services, advanced model (XS Advanced), by providing a variety of tools, such as syntax-aware editors for code and SAP HANA artifacts, graphical editors for CDS data models and calculation views, as well as inspection, testing and debugging tools.

SAP Web IDE works in conjunction with the SAP HANA database explorer, the SAP HANA deployment infrastructure (HDI), the Application Lifecycle Management tools (ALM) and the XS Advanced runtime platform, as shown in the following diagram.



## Features and Benefits

SAP Web IDE for SAP HANA and the database explorer enhance the development experience by providing the following features:

### Integrated Workspace and Project Management

- The latest tool versions are always available without additional installation
- Full workspace management on the server
- Integrated Git-based version control

- A dedicated template and wizards for multi-target application projects

## Development Tools

- For SAP HANA database artifact development:
  - Graphical modelers for complex artifacts, such as data models or calculation views
  - An SQL console and an MDX console in the database explorer
  - Integrated browsing of the database catalog in the database explorer
  - Syntax highlighting and content assistance for selected artifacts
  - Integrated performance analysis tools for SQL script and calculation views
- For Node.js and Java development
  - Integrated debugger
  - Integrated test executions and results with source code navigation
  - Integrated test coverage
  - Direct application start in the browser
- For UI development
  - HTML5, JavaScript, and SAPUI5-aware syntax and content assistance
  - Direct application start in the browser

## Build, Run, and Deploy

- Incremental build support
- Structured console output and logging
- Automatic creation of development sandboxes on SAP HANA (HDI containers)
- Automatic provisioning of HDI container services
- Generation of deployment archives

## Supported Browsers

SAP Web IDE supports the following browsers:

- Chrome (latest version)
- Firefox (latest version)
- Internet Explorer 11 on Windows
- Edge (latest version) on Windows
- Safari (latest version) on macOS

For additional release-specific information and known issues, see the SAP Note [2457320](#).

## Related Information

[Developing Multi-Target Applications \[page 983\]](#)

## 14.2 Post-Installation Administration Tasks

Perform these administration tasks after installing SAP Web IDE, or later as required.

### [Enabling Access to the SAP Web IDE Administration and Development Tools \[page 962\]](#)

How to enable users to access the SAP Web IDE administration and development tools.

### [Managing Spaces for Development \[page 965\]](#)

To support the isolation of the development environment, SAP Web IDE allows developers to use dedicated spaces in XS Advanced for building and running their projects.

### [Managing SSL Certificates \[page 966\]](#)

The SAP Web IDE SSL Certificate Management tool enables administrators to manage SSL certificates for remote Git repositories that issue SSL certificates that are not trusted publicly.

### 14.2.1 Enabling Access to the SAP Web IDE Administration and Development Tools

How to enable users to access the SAP Web IDE administration and development tools.

## Obtain the URLs of SAP Web IDE and Administration Tools

To obtain the URLs of SAP Web IDE and its administration tools, run the following commands in the CLI of XS Advanced:

Table 357:

Command	Description
<code>xs app webide --urls</code>	Returns the SAP Web IDE URL, which can be opened in a supported browser. Pass this URL on to the developers who will be using SAP Web IDE.
<code>xs app xsa-admin --urls</code>	Returns the URL of the <i>XS Advanced Administration</i> tool.
<code>xs app di-space-enablement-ui --urls</code>	Returns the URL of the <i>Space Enablement</i> admin tool.
<code>xs app di-cert-admin-ui --urls</code>	Returns the URL of the <i>SAP Web IDE SSL Certificate Management</i> admin tool.

## Roles and Permissions for Administration and Development

The following table lists the roles and role collections required to access the tools and perform the administration and development tasks.

Table 358:

Function	Role/Role Collections	Description
Administration Development	XS_CONTROLLER_USER role collection	Grants read-write permissions within the assigned organization or space.
Administration Development	SpaceDeveloper role	Assigned per space in XS Advanced. Enables users to access the shared resources of the space, and to deploy, build, and run applications.
Administration	A role collection containing the WebIDE_Administrator role template	Enables users to access the SAP Web IDE administration tools, such as SSL management and space enablement.
Administration	XS_AUTHORIZATION_ADMIN role collection	Enables users to access the XS Advanced administration tools.
Development	A role collection containing the WebIDE_Developer role template	Enables users to develop applications using SAP Web IDE and SAP HANA database explorer.

## Assign the SAP Web IDE Role Templates to the Role Collections

### Context

SAP Web IDE supplies the following predefined role templates: `WebIDE_Administrator` and `WebIDE_Developer`.

You can assign these templates to existing role collections for administrators or developers, or to role collections newly created for this purpose. Assigning the supplied templates to these collections will grant the relevant SAP Web IDE permissions to the users in the respective roles.

You can create new role collections for SAP Web IDE users and assign the predefined templates to the role collections in the *Application Role Builder* tool available in *XS Advanced Administration*. To access this tool, you need the authorization scopes defined in the `XS_AUTHORIZATION_ADMIN` role collection.

### Procedure

1. Open *XS Advanced Administration*.
2. Choose the *Application Role Builder* tool.
3. To create a new role collection, choose *Configure Role Collections*, and click "+".
4. Enter a name, for example, `WEBIDE`, and optionally a description for the role collection, and click *Create*.

The newly created role collection appears in the left-side pane.

5. To assign a template to a role collection, click the collection in the list, click .
6. In the *Application Name* dropdown list, select a role to which you want to assign a template.
7. In the *Template Name* dropdown list, select either `WebIDE_Administrator` or `WebIDE_Developer`.
8. In the *Application Role* dropdown list, make the same selection, and save.

## Grant Developer Permissions to Users

### Context

To enable users to develop applications with SAP Web IDE for SAP HANA, create the necessary SAP HANA database users, and assign them to the development role collections listed in the *Roles and Permissions for Administration and Development* section above.

### Procedure

1. Open [XS Advanced Administration](#).
2. Choose the [User Management](#) tool.
3. Select the user to whom you want to assign a role collection, and click .
4. In the *Role Collections* list, select the role collection that you want to assign, and save.

## Enable Browser Access to the Administration Tools

### Context

Certain browsers require additional actions to enable access to the [Space Enablement](#) and [SSL Certificate Management](#) administration tools.

For Firefox, modify the browser proxy settings by adding the XS Advanced hostname to the [No Proxy for](#) field under [Manual proxy configuration](#).

For both Firefox and Internet Explorer, perform the following steps:

### Procedure

1. Open [XS Advanced Administration](#), and choose the [Application Monitor](#) tool.
2. Locate the `devx-ui5` application in the list, and click [URL](#).

- 
3. Ignore the security certificate warnings, choose to continue to the website, and allow to open the site.

## 14.2.2 Managing Spaces for Development

To support the isolation of the development environment, SAP Web IDE allows developers to use dedicated spaces in XS Advanced for building and running their projects.

To create, maintain, and enable spaces for development, administrators perform the following tasks.

## Create and Manage Spaces in XS Advanced

### Context

You create and manage spaces in XS Advanced for different development teams in your organization.

### Procedure

1. In *XS Advanced Administration and Monitoring Tools*, open the *Organizations and Space Management administration* tool, and create the spaces required for your development teams.  
For more information, see *SAP HANA Administration Guide* → *Maintaining Organizations and Spaces in SAP HANA XS Advanced Model*.
2. Open the *SAP HANA Logical Database* tool, and enable the logical database (tenant) for XS Advanced.  
For more information, see *SAP HANA Administration Guide* → *Enable a Logical Database for use with XS Advanced*.
3. Open the *SAP HANA Service Broker Configuration* tool, and map the logical database to your organization and/or space.  
For more information, see *SAP HANA Administration Guide* → *Configure the Service Broker in XS Advanced*.

## Enable Spaces for Development

Enable development in the spaces that you have created, using the Space Enablement administration tool.

For information about accessing the tool, see [Enabling Access to the SAP Web IDE Administration and Development Tools \[page 962\]](#).

The Space Enablement tool allows you to enable spaces for development by deploying the builder component in each space. In this tool, you can view the status and builder version of all the spaces defined in your organization, and perform the required actions:

- If the status is *Enabled*, the current builder up-to-date, so no action is required.
- If the status is *Not Enabled*, the space has just been created. Click *Enable* to deploy the builder in the space.
- If the status is *Outdated*, click *Redeploy* to update the builder.

The process steps are displayed in the *Log* window. You can view the latest log for each space by clicking the



icon in the space row in the table.

### 14.2.3 Managing SSL Certificates

The SAP Web IDE SSL Certificate Management tool enables administrators to manage SSL certificates for remote Git repositories that issue SSL certificates that are not trusted publicly.

For information about accessing the tool, see [Enabling Access to the SAP Web IDE Administration and Development Tools \[page 962\]](#).

In this tool, you can view the list of the SSL certificates currently stored on the SAP Web IDE server, and perform the following actions:

- Upload a new certificate to the server.
- Choose to download a certificate, and open the downloaded file to view the details.
- Choose to delete a certificate from the server.

## 14.3 Getting Started

Start working with SAP Web IDE for SAP HANA by getting a user account, logging on, and getting familiar with the environment.

### Prerequisites

- Your SAP HANA administrator has installed SAP Web IDE for SAP HANA (SAP Web IDE).
- Your SAP HANA administrator has granted you the permissions required to access SAP Web IDE, and performed the necessary configuration tasks, as described in [Post-Installation Administration Tasks \[page 962\]](#).

## Procedure

1. Navigate to the URL of the installed SAP Web IDE, and log on to XS Advanced with your user credentials.  
The SAP Web IDE window opens.
2. Get familiar with the environment, and learn how to perform the basic tasks.

### Note

After a period of inactivity of 80 minutes, you will be logged off automatically. To resume work, you will need to log on again.

#### [Navigating SAP Web IDE \[page 967\]](#)

In SAP Web IDE, you can use the menu options and toolbar icons to perform various operations.

#### [Set User Preferences \[page 969\]](#)

You can set your preferences for working in SAP Web IDE.

#### [Enabling Optional Features \[page 971\]](#)

You can enable optional SAP Web IDE features to use in application development.

#### [Keyboard Shortcuts \[page 972\]](#)

You can use keyboard shortcuts to perform actions in SAP Web IDE.

#### [Workspace Search Options \[page 975\]](#)

SAP Web IDE's search options include a simple search within an open file in the code editor, advanced search and replace across multiple files in a project, and a search to find references.

#### [Working with Files and Folders \[page 980\]](#)

Use keyboard shortcuts and context menus to work with files and folders.

#### [Resizing Panes \[page 982\]](#)

You can show and hide various panes, as well as change their sizes, to fit your way of working.

## 14.3.1 Navigating SAP Web IDE

In SAP Web IDE, you can use the menu options and toolbar icons to perform various operations.

SAP Web IDE contains two perspectives: development and database explorer. Each perspective has its own set and layout of menus, toolbars and panes, as described below.

## Development Perspective

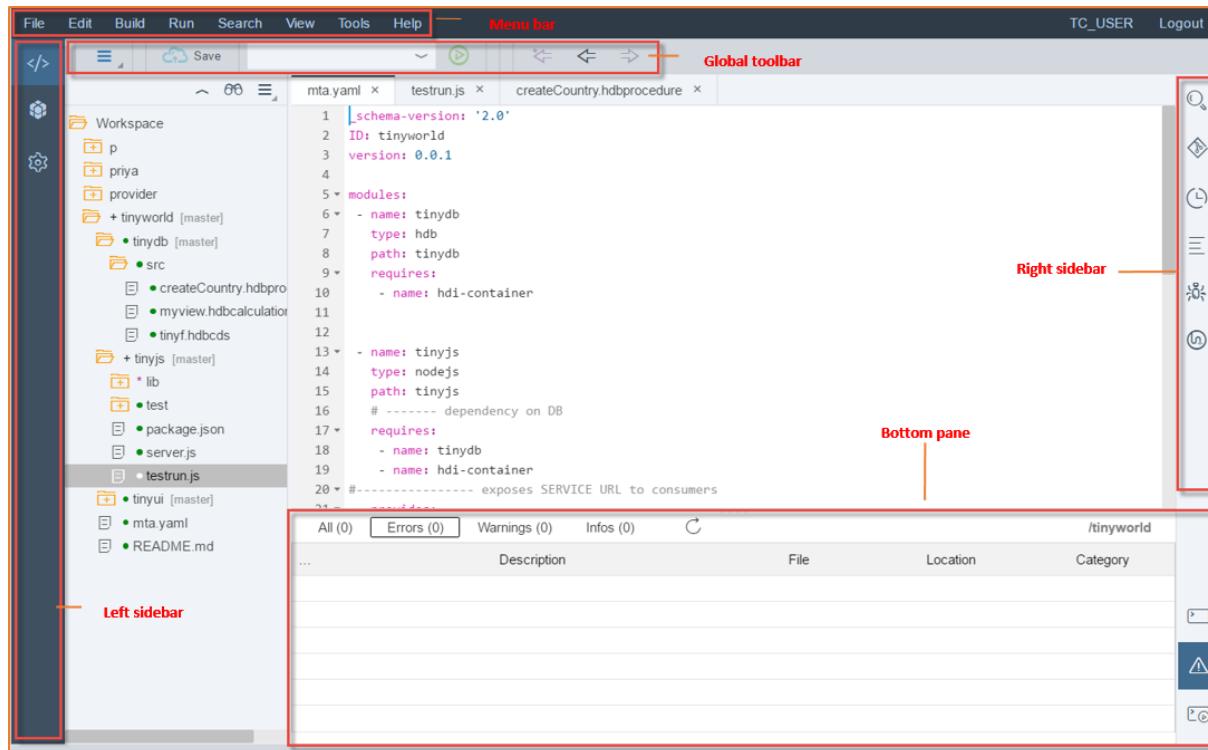
**Menu bar** - Provides access to all operations available in SAP Web IDE.

**Global toolbar** - Depending on the item that is activated in the workspace, you can choose from the icons in the global toolbar (icons of actions that are not applicable are grayed out).

**Left sidebar** - Use the buttons to switch between the development workspace, the database explorer, and user preferences.

**Right sidebar** - Use the buttons to switch between the different panes available in SAP Web IDE (for example, Git pane, Outline pane, and so on).

**Bottom pane** - Use the toggle buttons to display the console or the problems view.



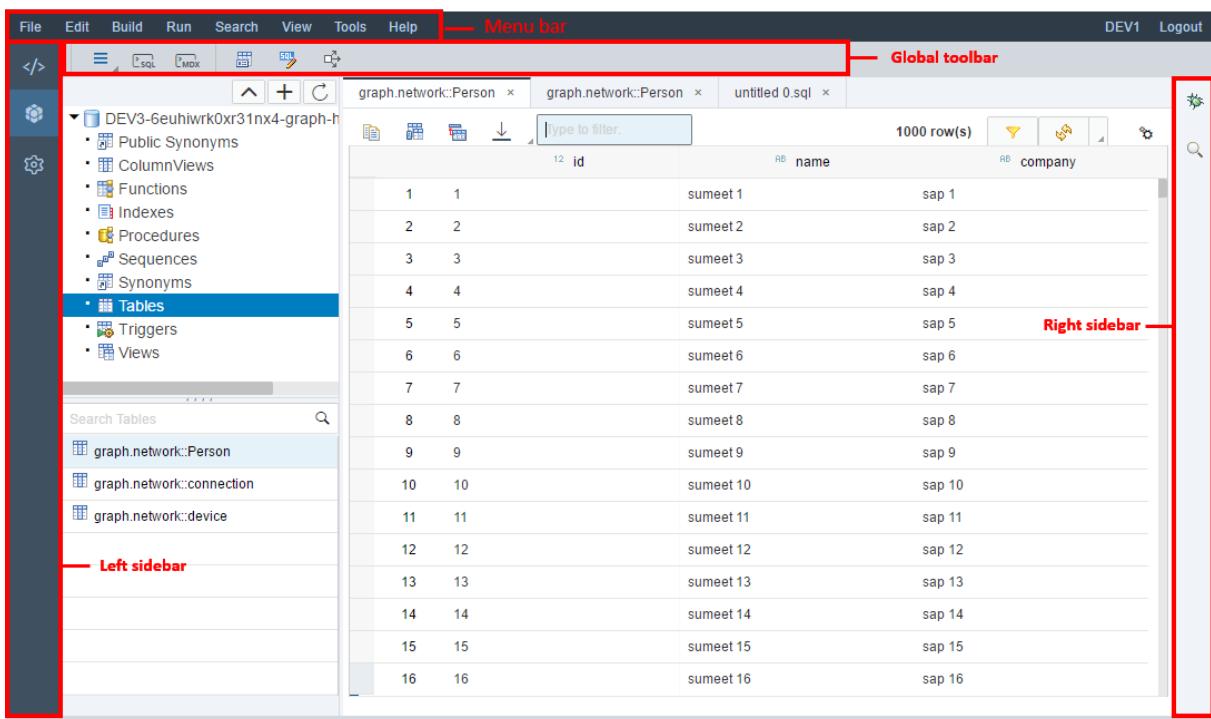
## Database Explorer Perspective

**Menu bar** - Only the Help menu bar item is activated for the database explorer.

**Global toolbar** - Depending on the item that is activated in the workspace, you can choose from the icons in the global toolbar (icons of actions that are not applicable are grayed out).

**Left sidebar** - Use the buttons to switch between the development workspace, the database explorer, and user preferences.

**Right sidebar** - Use the buttons to switch between the different panes available in the database explorer.



**Parent topic:** Getting Started [page 966]

## Related Information

- [Set User Preferences \[page 969\]](#)
- [Enabling Optional Features \[page 971\]](#)
- [Keyboard Shortcuts \[page 972\]](#)
- [Workspace Search Options \[page 975\]](#)
- [Working with Files and Folders \[page 980\]](#)
- [Resizing Panes \[page 982\]](#)
- [Working with the SAP HANA Database Explorer \[page 70\]](#)

### 14.3.2 Set User Preferences

You can set your preferences for working in SAP Web IDE.

#### Procedure

1. To open the *Preferences* perspective, in the left sidebar, choose (Preferences).

2. Select the type of preference that you want to change.

For example, you can set the following SAP Web IDE preferences:

Table 359:

Preferences	More Information
Code check settings to enable and disable inline code validation. The project-level appearance settings are used to validate code and flag messages.	<a href="#">Configuring Code Checking [page 1105]</a>
Code completion settings to enable inline code completion and comment completion.	<a href="#">Configuring Code Completion [page 1104]</a>
Code editor settings to select the theme to be used by the code editor and the font size. You can also choose to have the changes in all open documents saved automatically at preset intervals.	<a href="#">Configuring the Code Editor [page 1096]</a>
Set up Git for source control.	<a href="#">Setting Up Git [page 1117]</a>
Customize SAP Web IDE keyboard shortcuts.	<a href="#">Customizing Keyboard Shortcuts [page 974]</a>
Delete or ignore workspace persistence.	<p>If you start SAP Web IDE after you have been logged out, the system restores the latest status of the workspace with its preference settings and all the editors that have been opened.</p> <p>If you want to reload SAP Web IDE while ignoring this persistence feature, for example, because an editor cannot be loaded or is frozen, add the parameter <b>settings=ignore</b> to the URL and refresh your browser. The persistence information is not deleted (meaning that when you remove the parameter from the URL and refresh your browser, SAP Web IDE restores the latest status of the workspace and the editors).</p>

**Task overview:** [Getting Started \[page 966\]](#)

## Related Information

[Navigating SAP Web IDE \[page 967\]](#)

[Enabling Optional Features \[page 971\]](#)

[Keyboard Shortcuts \[page 972\]](#)

[Workspace Search Options \[page 975\]](#)

[Working with Files and Folders \[page 980\]](#)

[Resizing Panes \[page 982\]](#)

### 14.3.3 Enabling Optional Features

You can enable optional SAP Web IDE features to use in application development.

#### Context

SAP Web IDE includes features that are not enabled by default. If you would like to use additional features, perform the following steps:

#### Procedure

1. To open the Preferences perspective, in the left sidebar, choose  (Preferences).
2. Choose *Features*.
3. Select the toggle button for the feature you want to enable.
4. Choose *Save*.
5. Refresh your browser.

#### Results

You can use the enabled features in your projects.

**Task overview:** [Getting Started \[page 966\]](#)

#### Related Information

[Navigating SAP Web IDE \[page 967\]](#)

[Set User Preferences \[page 969\]](#)

[Keyboard Shortcuts \[page 972\]](#)

[Workspace Search Options \[page 975\]](#)

[Working with Files and Folders \[page 980\]](#)

[Resizing Panes \[page 982\]](#)

## 14.3.4 Keyboard Shortcuts

You can use keyboard shortcuts to perform actions in SAP Web IDE.

The following shortcuts are available in SAP Web IDE:

### i Note

- The majority of these shortcuts are available only for English language keyboards.
- On Mac OS keyboards, the `Alt` key is also called the `Option` key.

You can modify the SAP Web IDE predefined keyboard shortcuts as described in [Customizing Keyboard Shortcuts \[page 974\]](#).

Table 360:

Action	Microsoft Windows Keyboard Shortcut	Mac OS Keyboard Shortcut
New file	<code>Ctrl</code> + <code>Alt</code> + <code>N</code>	<code>Command</code> + <code>Alt</code> + <code>N</code>
New folder	<code>Ctrl</code> + <code>Alt</code> + <code>Shift</code> + <code>N</code>	<code>Command</code> + <code>Alt</code> + <code>Shift</code> + <code>N</code>
New project	<code>Ctrl</code> + <code>Alt</code> + <code>Shift</code> + <code>O</code>	<code>Command</code> + <code>Alt</code> + <code>Shift</code> + <code>O</code>
Close file	<code>Alt</code> + <code>W</code>	<code>Alt</code> + <code>W</code>
Close all files	<code>Alt</code> + <code>Shift</code> + <code>W</code>	<code>Alt</code> + <code>Shift</code> + <code>W</code>
Save file	<code>Ctrl</code> + <code>S</code>	<code>Command</code> + <code>S</code>
Save all files	<code>Ctrl</code> + <code>Shift</code> + <code>S</code>	<code>Command</code> + <code>Shift</code> + <code>S</code>
Show Code Completion Suggestions	<code>Ctrl</code> + <code>Space</code>	<code>Ctrl</code> + <code>Space</code>
Undo	<code>Ctrl</code> + <code>Z</code>	<code>Command</code> + <code>Z</code>
Redo	<code>Ctrl</code> + <code>Y</code>	<code>Command</code> + <code>Y</code>
Cut	<code>Ctrl</code> + <code>X</code>	<code>Command</code> + <code>X</code>
Copy	<code>Ctrl</code> + <code>C</code>	<code>Command</code> + <code>C</code>
Paste	<code>Ctrl</code> + <code>V</code>	<code>Command</code> + <code>V</code>
Rename file or folder	<code>F2</code>	<code>F2</code>
Delete	<code>Del</code>	<code>Del</code>
Move to the tab on the right	<code>Alt</code> + <code>R</code>	<code>Alt</code> + <code>Right Arrow</code>
Move to the tab on the left	<code>Alt</code> + <code>Q</code>	<code>Alt</code> + <code>Q</code>
Navigate back to the previous file	<code>Ctrl</code> + <code>Alt</code> + <code>R</code>	<code>Command</code> + <code>Alt</code> + <code>R</code>
Navigate forward	<code>Ctrl</code> + <code>Alt</code> + <code>Y</code>	<code>Command</code> + <code>Alt</code> + <code>Y</code>
Navigate to the file that was edited last	<code>Ctrl</code> + <code>Shift</code> + <code>9</code>	<code>Command</code> + <code>Shift</code> + <code>9</code>

Action	Microsoft Windows Keyboard Shortcut	Mac OS Keyboard Shortcut
Show/Hide all characters	[Ctrl] + [I]	[Command] + [I]
Toggle line comment	[Ctrl] + [/]	[Command] + [/]
Toggle line comment (German language keyboard)	[Alt] + [7]	[Alt] + [H]
Toggle block comment	[Ctrl] + [Shift] + [/]	[Command] + [Shift] + [/]
Toggle block comment (German language keyboard)	[Ctrl] + [Shift] + [7]	[Command] + [Shift] + [7]
Add todo comment	[Ctrl] + [Alt] + [T]	[Command] + [Alt] + [T]
Indent line	[Tab]	[Tab]
Outdent line	[Shift] + [Tab]	[Shift] + [Tab]
Move lines up	[Alt] + [Up Arrow]	[Shift] + [Tab]
Move lines down	[Alt] + [Down Arrow]	[Alt] + [Down Arrow]
Copy lines up	[Alt] + [Shift] + [Up Arrow]	[Alt] + [Shift] + [Up Arrow]
Copy lines down	[Alt] + [Shift] + [Down Arrow]	[Alt] + [Shift] + [Down Arrow]
Beautify file format	[Ctrl] + [Alt] + [B]	[Command] + [Alt] + [B]
Generate JSDoc Comment	[Ctrl] + [Alt] + [J]	[Command] + [Alt] + [J]
Goto JavaScript definition	[Ctrl] + [Alt] + [G]	[Command] + [Alt] + [G]
Run	[Alt] + [Shift] + [R]	[Alt] + [Shift] + [R]
Run without frame	[Ctrl] + [Alt] + [Shift] + [R]	[Command] + [Alt] + [Shift] + [R]
Find	[Ctrl] + [F]	[Command] + [F]
Find and replace	[Ctrl] + [H]	[Command] + [H]
Find references	[Ctrl] + [Alt] + [W]	[Ctrl] + [Alt] + [W]
Advanced repository search	[Ctrl] + [Shift] + [F]	[Command] + [Shift] + [F]
Maximize/Restore Active Editor	[Ctrl] + [M]	[Command] + [M]
View console	[Ctrl] + [Shift] + [M]	[Command] + [Shift] + [M]
View Git pane	[Ctrl] + [Shift] + [V]	[Command] + [Shift] + [V]
View outline	[Ctrl] + [Shift] + [U]	[Command] + [Shift] + [U]
Open Preferences perspective	[Ctrl] + [.]	[Command] + [.]
Open Resource	[Ctrl] + [Shift] + [R]	[Command] + [Shift] + [R]
Go to line	[Ctrl] + [L]	[Command] + [L]

Action	Microsoft Windows Keyboard Shortcut	Mac OS Keyboard Shortcut
Refactor	[Alt] + [J]	[Alt] + [J]
Open i18n	[Alt] + [I]	[Alt] + [I]

**Parent topic:** [Getting Started \[page 966\]](#)

## Related Information

[Navigating SAP Web IDE \[page 967\]](#)

[Set User Preferences \[page 969\]](#)

[Enabling Optional Features \[page 971\]](#)

[Workspace Search Options \[page 975\]](#)

[Working with Files and Folders \[page 980\]](#)

[Resizing Panes \[page 982\]](#)

### 14.3.4.1 Customizing Keyboard Shortcuts

You can modify the SAP Web IDE predefined keyboard shortcuts according to your preferences by recording them.

## Context

Follow the instructions below to change a keyboard shortcut.

### ⚠ Caution

Using certain browser shortcuts can trigger the assigned browser action. For example, the shortcuts below are already assigned in the Chrome browser and should not be used as SAP Web IDE shortcuts.

- [Ctrl] + [N]
- [Ctrl] + [T]
- [Ctrl] + [W]
- [Ctrl] + [Shift] + [N]
- [Ctrl] + [Shift] + [T]
- [Ctrl] + [Shift] + [W]

## Procedure

1. In the *Preferences* area, choose *Keyboard Shortcuts*.
2. In the *Shortcut* column, double-click a shortcut that you want to modify.
3. Record your new shortcut by pressing a key combination on your keyboard.

 Note

You can revert to the SAP Web IDE default setting by clicking the *Revert*  button.

4. Choose the *Save* button.

## Results

The new customized keyboard shortcuts can now be used.

To use your customized shortcuts in SAP Web IDE, select the *User-Defined* option and then choose the *Save* button. You can switch between the SAP Web IDE default shortcuts and your customized shortcuts without losing your shortcut definitions. If needed, you can remove all your modified shortcuts by choosing the *Clear User-Defined* button. This action restores the SAP Web IDE default shortcuts and deletes your customized shortcuts.

## Related Information

[Keyboard Shortcuts \[page 972\]](#)

## 14.3.5 Workspace Search Options

SAP Web IDE's search options include a simple search within an open file in the code editor, advanced search and replace across multiple files in a project, and a search to find references.

[Searching for Files or Content in the Workspace \[page 976\]](#)

Perform a file or string search within a folder or across all projects in your workspace.

[Replacing Strings Across Multiple Files \[page 977\]](#)

Perform an advanced find and replace across multiple files with the Search pane.

[Finding and Replacing in an Open File \[page 979\]](#)

Perform a simple find and replace within a single open file from the code editor.

**Parent topic:** [Getting Started \[page 966\]](#)

## Related Information

[Navigating SAP Web IDE \[page 967\]](#)

[Set User Preferences \[page 969\]](#)

[Enabling Optional Features \[page 971\]](#)

[Keyboard Shortcuts \[page 972\]](#)

[Working with Files and Folders \[page 980\]](#)

[Resizing Panes \[page 982\]](#)

### 14.3.5.1 Searching for Files or Content in the Workspace

Perform a file or string search within a folder or across all projects in your workspace.

#### Context

To determine the success of your search, use the preview to quickly validate results, and refine your search further as needed.

#### Procedure

1. Open the *Search* pane by choosing the search icon in the right sidebar.
2. In the *Search* pane, define the type of search:
  - To search for a file, enter a full or partial file name, then choose *File name*.  
The file name search is case sensitive. If you do not know whether the file name is capitalized, use a wildcard, such as an asterisk (\*). For example, instead of searching for `file1`, search for `*ile1`.
  - To search for a string, enter the string, then choose *File content*.
3. (Optional) Refine search results further:
  - Set *Search in* to determine where the search scope is to be limited: use *All* to search across the entire workspace, use *Project* to search across a single project, or use *Folder* to search within the current directory. If you choose *Folder* or *Project*, type the path to the target that you want to search, or select it in the workspace.
  - Set *Find* to limit file or content searches to files with a specified extension.
4. Choose *Search*.
5. In the results list at the bottom of the *Search* pane, you see search results, organized according to file.
  - Expand or collapse files with results, which is useful when lists are long.
  - Below each file, matches are highlighted.
  - Hover over a specific result to get the context of that item.

If you do a search of file content for the string `test`, you might see results shown as in the following example. You can choose the /Test/ folder as the target, then change *Find* from all file types to `*.js`.

Search results are limited to two files. By hovering over the first result in the first file, you can quickly review the context without opening the file.

6. When you have identified the file and row that you require, double-click the specific search result to open the file with the appropriate row highlighted.

**Task overview:** [Workspace Search Options \[page 975\]](#)

## Related Information

[Replacing Strings Across Multiple Files \[page 977\]](#)

[Finding and Replacing in an Open File \[page 979\]](#)

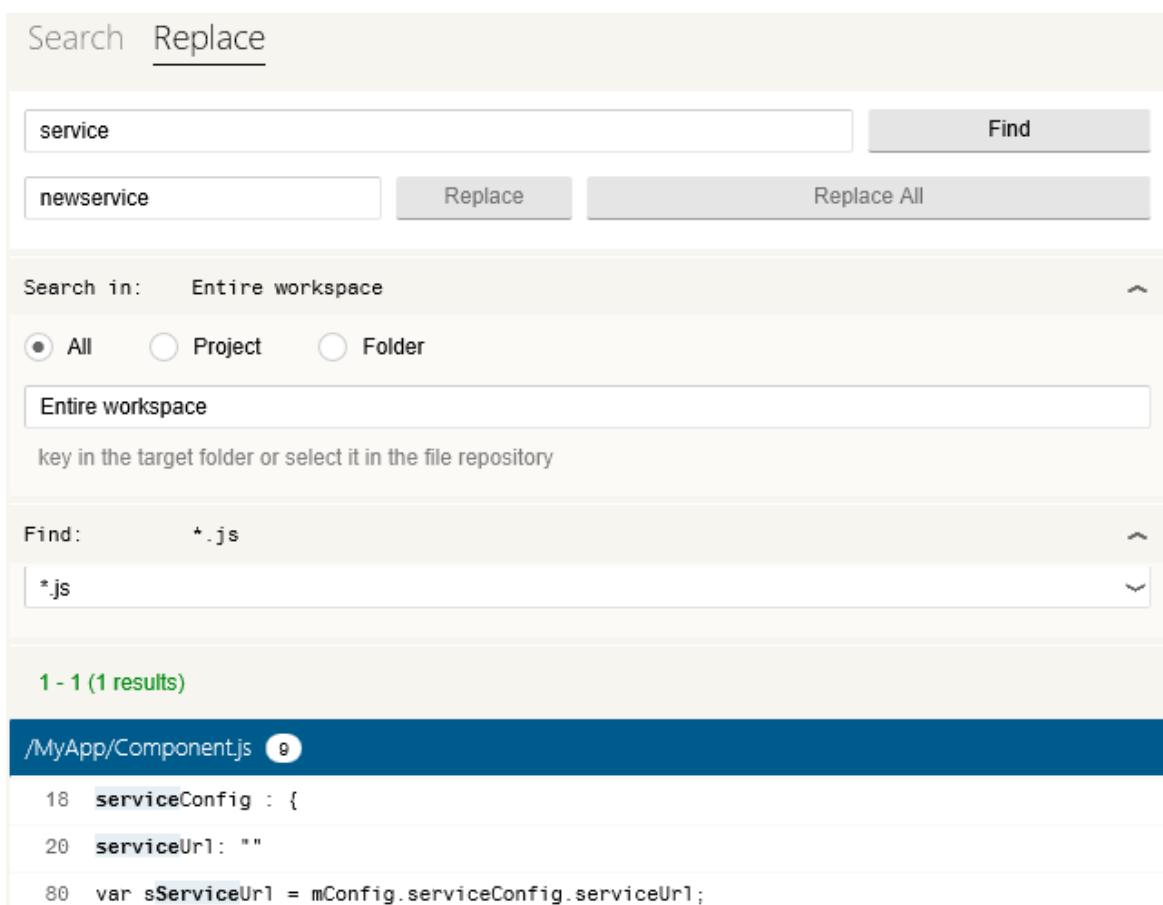
### 14.3.5.2 Replacing Strings Across Multiple Files

Perform an advanced find and replace across multiple files with the Search pane.

#### Procedure

1. With or without any files open in the code editor, open the [Search](#) pane by choosing the search icon in the right sidebar.
2. In the [Search](#) pane, type the string you want to find.
3. (Optional) Refine search results further:
  - Set [Search in](#) to determine where the search scope is to be limited: use [All](#) to search across the entire workspace, use [Project](#) to search across a single project, or use [Folder](#) to search within the current directory. If you choose [Folder](#) or [Project](#), enter the path to the target that you want to search.
  - Set [Find](#) to limit searches to files with a specified extension.
4. Choose [Search](#).

In the results list at the bottom of the [Search](#) pane, you see search results, organized according to file.



5. In the *Replace* pane, enter the replace text and perform one of the following:

- A serial replace that allows you to evaluate each instance case-by-case. In the results list, hover over each result and click the replace icon at the end of the line to replace the string. As each instance is replaced, it disappears from the results list. Skip all results that you do not want to replace.
- An inline replace that allows you to select only specific file results, and choose the replace icon adjacent to the result itself.
- A global replace of all results in the list. Choose *Replace All*.

**i Note**

You cannot undo the global replace operation.

**Task overview:** [Workspace Search Options \[page 975\]](#)

## Related Information

[Searching for Files or Content in the Workspace \[page 976\]](#)

---

[Finding and Replacing in an Open File \[page 979\]](#)

### 14.3.5.3 Finding and Replacing in an Open File

Perform a simple find and replace within a single open file from the code editor.

#### Context

These searches are limited to the file that is currently in view. To search and replace across multiple files, perform an advanced search. See [Replacing Strings Across Multiple Files \[page 977\]](#).

#### Procedure

1. Open the file that you want to perform a simple string search in.
2. Select   *Search > Find and Replace*.
3. Do one of the following:
  - To perform a basic search only, enter the search string in the search field (the first field). Use the adjacent up and down arrows to find instances previous to or following the current cursor location. Use the icons to further limit the search for:
    - Regular expressions
    - Case-sensitive
    - Whole words only
  - To replace text, enter both a search string and a *Replace with* string. To search and replace one by one, click *Replace*. If other instance exists, the next one will be selected. Otherwise, to automatically search and replace all instances, click *All*.
  -
4. If you replaced strings, save the file.

**Task overview:** [Workspace Search Options \[page 975\]](#)

#### Related Information

[Searching for Files or Content in the Workspace \[page 976\]](#)

[Replacing Strings Across Multiple Files \[page 977\]](#)

## 14.3.6 Working with Files and Folders

Use keyboard shortcuts and context menus to work with files and folders.

You can open multiple files in the SAP Web IDE code editor, and you can navigate between them. Before closing files, save or discard your changes. You can cut, copy, paste, and rename files and folders.

### Note

For keyboard shortcuts, see [Keyboard Shortcuts \[page 972\]](#).

Table 361:

Action	Context Menu or Toolbar Icon
Open selected files in the workspace.	 <a href="#">Open with Code Editor</a>
Search and open files in the workspace.	<a href="#">Open Resource</a>
Navigate between open files in the workspace.  For more navigation options in the code editor, see <a href="#">Working in the Code Editor [page 1097]</a> .	 (Back)   (Forward)   (Last Edit Location)
Close one or more files.	Tab menu: <ul style="list-style-type: none"><li><a href="#">Close</a></li><li><a href="#">Close All</a></li><li><a href="#">Close Other</a> (Closes all files except the active file.)</li></ul>
Import a file from the local file system or from an archive to a selected folder.	 <a href="#">Import From File System</a>
 <b>Note</b>  File names can contain only letters, numbers, and the period (.) and underscore (_) characters.	Check the <a href="#">Extract Archive</a> checkbox to extract files in the archive into the selected folder
Export selected files or folders.	<a href="#">Export</a>
Archive a folder and save it as a compressed (.zip) file under the parent folder.  Compressed files larger than 20 MB cannot be archived.	<a href="#">Archive</a>
Refresh the view of the selected files and folders in the workspace.	<a href="#">Refresh</a>

Action	Context Menu or Toolbar Icon
Link the code editor to the workspace to easily see the location of any active file or show the location of the current active file without linking.	<p>Tab menu:</p> <ul style="list-style-type: none"> <li>• <a href="#">Link Workspace to Editor</a></li> <li>• <a href="#">Show in Workspace</a></li> </ul>

### [Working with Multiple Open Files \[page 981\]](#)

You can open multiple files in the code editor, each of which appears in a separate tab.

**Parent topic:** [Getting Started \[page 966\]](#)

## Related Information

[Navigating SAP Web IDE \[page 967\]](#)

[Set User Preferences \[page 969\]](#)

[Enabling Optional Features \[page 971\]](#)

[Keyboard Shortcuts \[page 972\]](#)

[Workspace Search Options \[page 975\]](#)

[Resizing Panes \[page 982\]](#)

## 14.3.6.1 Working with Multiple Open Files

You can open multiple files in the code editor, each of which appears in a separate tab.

### Tab Area

If more than one file is opened, each one is shown in a different editor tab in the code editor. Each tab shows the file name and its extension as well as an , which allows you to close the file by clicking it. If a file is being edited, its file name is shown in bold and a small asterisk appears next to its name.

When you hover over the file name, the file path appears in a tooltip.

### Maximizing the Editing Area

To enlarge the editing area, double-click any tab in the editor. All open SAP Web IDE panes disappear, and the complete current editor area is fitted to your screen. To restore, double-click the tab.

### Tab Overflow

If you open more files in the editor than there is space to display their tabs in the editor area, a small menu appears on the right of the tab area:



Use the arrows to navigate through the tabs in either direction. You can also click the list icon on the right to see the names of all opened files and then choose one from the list to make it the active editor tab.

**Parent topic:** [Working with Files and Folders \[page 980\]](#)

## 14.3.7 Resizing Panes

You can show and hide various panes, as well as change their sizes, to fit your way of working.

### Context

You can make the following adjustments:

- Show and hide the workspace by choosing [View > Workspace](#).
- Change the sizes of the workspace, editors area, and panes by dragging the splitters between them.
- Maximize an editor window (that is, hide the workspace and any panes on the right) by double-clicking the title bar of the editor window.

**Note**

Double-click again to restore the editor window to its original size. This always redisplays the workspace, and displays any previously opened pane on the right.

- Restore SAP Web IDE pane defaults by choosing [View > Reset to Default](#).

Changes to the layout are saved, so the SAP Web IDE will appear the same the next time you open it.

**Task overview:** [Getting Started \[page 966\]](#)

### Related Information

[Navigating SAP Web IDE \[page 967\]](#)

[Set User Preferences \[page 969\]](#)

[Enabling Optional Features \[page 971\]](#)

[Keyboard Shortcuts \[page 972\]](#)

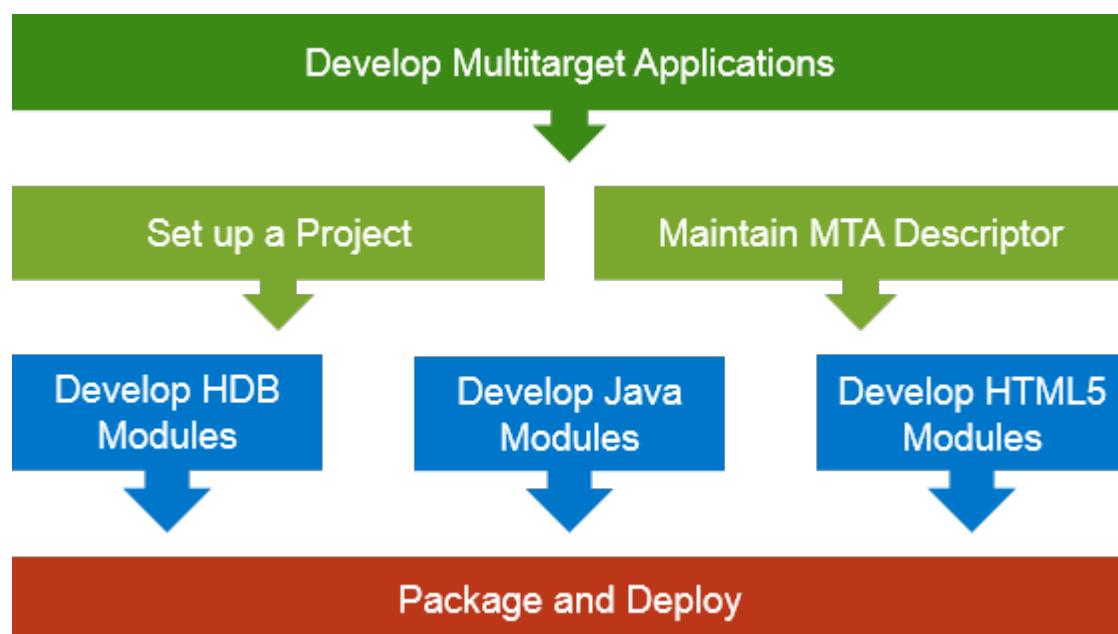
[Workspace Search Options \[page 975\]](#)

[Working with Files and Folders \[page 980\]](#)

## 14.4 Developing Multi-Target Applications

Multi-target, or multi-tier applications are comprised of multiple software modules representing the data, business logic and UI tiers. These modules are created with different technologies and are deployed to different target platforms, yet share the same development lifecycle.

The multi-target application concept aims at orchestrating the deployment of all these modules so that all runtime dependencies are properly resolved and the application functions as expected. This is achieved by supplying to the deployment tools a set of descriptors that define the interdependencies and deployment scenarios for all modules in the application.



- [Developing Multi-Target Applications \[page 983\]](#)
- [Setting Up Application Projects \[page 989\]](#)
- [Inside an MTA Descriptor \[page 985\]](#)
- [Developing SAP HANA Database \(HDB\) Modules \[page 992\]](#)
- [Developing Java Modules \[page 1033\]](#)
- [Developing HTML5 Modules \[page 1039\]](#)
- [Packaging and Deploying Applications \[page 1092\]](#)

## Terms and Concepts

Table 362:

Term	Description
Multi-target application (MTA)	An application comprised of multiple software modules, which are created with different technologies and deployed to different target platforms, yet share the same lifecycle. In the context of this guide, an "application" is an MTA.
Target platform	A platform to which a module is deployed, such as an SAP HANA database.
MTA descriptor	A YAML file named <code>mta.yaml</code> that contains a list of all entities, such as modules, resources, and properties that belong to an application or are used by it at runtime, and the dependencies between them. It is automatically generated when an MTA project is created or modified, or when a module is added or removed. The developer needs to edit the descriptor manually to define resources, properties, and dependencies, as well as fill in missing information.
Module	A self-contained application of a certain type, which is developed, packaged, and deployed to a target platform as part of an MTA.
Module type	A type that defines the structure, development technology, and target platform of a module. SAP Web IDE supports the following module types, representing the three application tiers: <ul style="list-style-type: none"><li>• SAP HANA database (HDB) module - represents the data tier.</li><li>• Java and Node.js modules - represent the business logic tier.</li><li>• HTML5 module - represents the UI tier.</li></ul> Additional module types: <ul style="list-style-type: none"><li>• Streaming analytics module - if SAP HANA streaming analytics is installed on your SAP HANA system</li></ul>
Resource	Any resource, such as an external service, property, or environment variable, that is required by a module at runtime but not provided by the module itself.
Property	A property (key-value pair) of an application, module, or resource, that is used during deployment or runtime.
Parameter	A reserved variable belonging to a module or resource, whose value is used during deployment or runtime. Parameters can be read-only, write-only, or read-write. The values of writable parameters can be specified in the descriptor.
Dependency	A relationship between a module and another module, resource, or property, such as <code>provides</code> and <code>requires</code> . <ul style="list-style-type: none"><li>• <code>provides</code>: indicates the properties or parameters that are provided by a module or resource to other modules.</li><li>• <code>requires</code>: indicates other modules or resources that are required by a module in order to run.</li></ul>
Deployment archive	An archive similar to JAR into which all the application artifacts are packaged for deployment.

### [Inside an MTA Descriptor \[page 985\]](#)

The multi-target application (MTA) descriptor contains the metadata of all entities comprising an application or used by it during deployment or runtime, and the dependencies between them.

### [Application Development Workflow \[page 989\]](#)

Development workflows and tasks that you perform to create and maintain multi-target applications in SAP Web IDE for SAP HANA.

### [Setting Up Application Projects \[page 989\]](#)

You can set up a multi-target application (MTA) project by creating it from scratch, importing from an archive, or cloning from a Git repository.

### [Developing SAP HANA Database \(HDB\) Modules \[page 992\]](#)

An SAP HANA database (HDB) module is a collection of related design-time database artifacts, such as data models, views, or procedures.

### [Developing Node.js Modules \[page 1025\]](#)

A Node.js module is a collection of related JavaScript files and service definitions that implement the business logic of your application.

### [Developing Java Modules \[page 1033\]](#)

A Java module is a collection of related Java files and service definitions. Java modules implement the business logic of your application, either instead of or in addition to Node.js modules. A Java module can be either a Java Web Archive (WAR) or Java Archive (JAR) built with Apache Maven.

### [Developing HTML5 Modules \[page 1039\]](#)

An HTML5 module is a collection of related HTML5 files that implement the user interface of your application.

### [Developing SAP Fiori Launchpad Modules \[page 1080\]](#)

SAP Fiori launchpad modules are used to create a SAP Fiori launchpad site that runs on the XS Advanced Model.

### [Packaging and Deploying Applications \[page 1092\]](#)

At the last stage of multi-target application development, you need to package your application and deploy it to the SAP HANA XS Advanced Model (XS Advanced) production system.

## 14.4.1 Inside an MTA Descriptor

The multi-target application (MTA) descriptor contains the metadata of all entities comprising an application or used by it during deployment or runtime, and the dependencies between them.

The MTA descriptor (the `mta.yaml` file located in the root project folder) is automatically generated when an application project is created from scratch, and it is updated when the project properties change or when a module is added or removed. However, not all the necessary information can be generated automatically. You need to maintain the descriptor manually to define resources, properties, and dependencies, as well as fill in missing information.

## MTA Editor

The MTA descriptor is written in the YAML format, which has strict syntax requirements. You can edit the descriptor in the text-based code editor, but we recommend you use the visual MTA editor because it provides input validation.

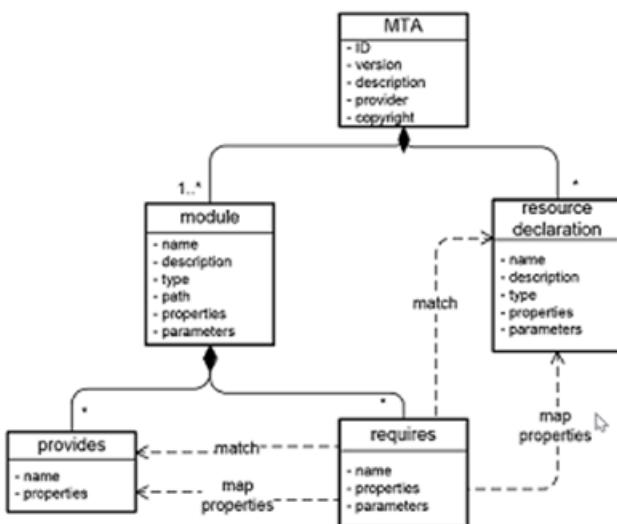
To set the MTA editor as the default for the MTA descriptor, go to [Preferences](#) [Default Editors](#) and set the editor for the [MTA Application Descriptor](#).

### Note

The MTA visual editor removes comments and formats the file. If you want to add comments, use the code editor. To open the code editor, either make the code editor the default editor or right-click the file and choose  *Open With > Code Editor*.

If you edit the file with the code editor, it is important to use spaces rather than tabs for indentation.

## MTA Descriptor Model



## MTA Descriptor Example

```
ID: com.sap.node.hello.world.db
version: 1.0.0
description: A Hello World sample application
provider: SAP Sample generator
copyright: 2016 SAP SE
modules:
  - name: node-hello-world-db
    type: hdb
    path: db
    requires:
      - name: hdi-container
    provides:
      - name: node-hello-world-db
  - name: node-hello-world
    type: html5
    path: web
    requires:
      - name: uaa
      - name: backend_api
noje.js module
  group: destinations
  properties:
    # dependency on the UAA service
    # name of the 'provides' element in the
    # name of a target environment variable
```

```

        name: nodejs          # key values in json format in the
'destinations' variable
        url: ~{url}
        forwardAuthToken: true
      - name: node-hello-world-backend
        type: nodejs
        path: js
        requires:
          - name: node-hello-world-db
          - name: hdi-container
          - name: uaa
        provides:
          - name: backend_api
            properties:
              url: ${default-url}
# Resources describe required services
resources:
  - name: hdi-container
    type: com.sap.xs.hdi-container
  - name: uaa
    type: com.sap.xs.uaa

```

## MTA Descriptor Elements

Table 363:

Sections	Element	Description	Possible Values
General	ID	The unique application ID.	Any number of unicode characters in a reverse-URL dot-notation, for example com.sap.mta.sample. The ID should be unique in the target runtime environment.
General	version	The application version.	Should follow the semantic versioning standard format (currently only the basic MAJOR.MINOR.PATCH format is supported). For more information, refer to the official site for semantic versioning.
General	description	Optional. A description of the application.	
General	provider	Optional. The application vendor/provider name.	
General	copyright	Optional. The copyright notice of the provider.	
modules	name	The module name.	Should be unique within the descriptor.
modules	path	The relative path to a module from the application root.	For example, ./backend
modules	type	The module type, which defines the design-time tools and builders required for the module.	One of the following values: hdb, nodejs, html5.

Sections	Element	Description	Possible Values
modules	requires	A subsection of a module section that contains the names of resources, other modules and/or properties provided by other modules, which are required by the current module to run.	<p>For example:</p> <pre>requires:   - name: backend     properties:       name: external       url: ~{url}</pre>
modules	provides	A subsection of a module section that specifies the properties provided by the current module or resource.	<p>For example:</p> <pre>provides:   - name: price_opt     properties:       protocol: http       uri:         host.domain</pre>
modules, resources	properties	Optional. A flat or hierarchical collection of properties, provided or required by a module or resource.	<p>For example:</p> <pre>properties:   - name: node-hello-world-db   - name: hdi-container   - name: uaa</pre>
modules, resources	parameters	<p>Optional. Reserved variables that can contain read-only, read-write, or write-only values, which can be used by deployment tools, but not at runtime.</p> <p>Parameters can be referenced with placeholders enclosed in \${ and }, which are resolved during deployment or runtime.</p>	<p>For example:</p> <pre>parameters: domain: price.sap.com</pre>
modules	group	Optional. A named group of properties from different providers used as a single lookup object.	<p>For example:</p> <pre>group: DESTINATIONS</pre>
modules	description	Optional. A description of the module, which does not appear in the application UI.	

## Related Information

[The Multi-Target Application Model](#) 

## 14.4.2 Application Development Workflow

Development workflows and tasks that you perform to create and maintain multi-target applications in SAP Web IDE for SAP HANA.

Workflow/Task	Instructions
1. Start SAP Web IDE for SAP HANA and get familiar with the working environment.	<a href="#">Getting Started [page 966]</a>
2. Set up a project for developing a multi-target application.	<a href="#">Setting Up Application Projects [page 989]</a>
3. Implement the database tier of your application.	<a href="#">Developing SAP HANA Database (HDB) Modules [page 992]</a>
4. Implement the business logic tier of your application.	<a href="#">Developing Node.js Modules [page 1025]</a> <a href="#">Developing Java Modules [page 1033]</a>
5. Implement the user interface of your application.	<a href="#">Developing HTML5 Modules [page 1039]</a>
6. Prepare your application for deployment.	<a href="#">Packaging and Deploying Applications [page 1092]</a>

**i** Note

If you have SAP HANA streaming analytics installed on your SAP HANA system, you can create streaming analytics modules. For more information, see *Working with Projects in SAP Web IDE for SAP HANA* in the *SAP HANA streaming analytics Developer Guide*.

## 14.4.3 Setting Up Application Projects

You can set up a multi-target application (MTA) project by creating it from scratch, importing from an archive, or cloning from a Git repository.

### Creating a Project from Scratch

You create an MTA project using a dedicated template provided by SAP Web IDE.

#### Procedure

1. From the *Workspace* menu, choose  *New*  *Project from Template*, and click *Next*.
2. Choose the *Multi-Target Application Project* template, and click *Next*.
3. Enter a project name, and click *Next*.
4. If needed, modify the *Application ID* and *Version* properties of the project. Both properties appear in the MTA descriptor (`mta.yaml` file).

5. Optionally, enter a description of the project.
6. Select an existing space in SAP HANA XS Advanced for your project from the *Space* dropdown list.

**i Note**

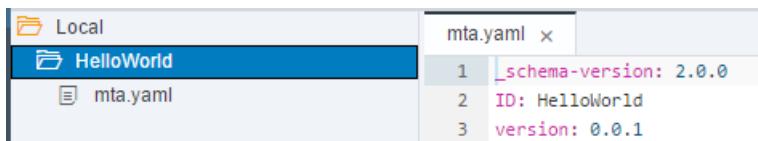
To support the isolation of the development environment, SAP Web IDE enables developers to use dedicated spaces in XS Advanced for building and running their projects. By selecting a space in SAP HANA XS Advanced for your project, you ensure that your application can share the resources with other applications or services running in this space, and that applications running in other spaces cannot access these resources.

The dropdown list contains the available spaces, created by your SAP HANA XS Advanced administrator, in which you have the *SpaceDeveloper* role. If the required space does not appear in the list, contact your administrator. If you do not select any space, an error will occur when you attempt to build or run your project.

You can change the space selection later, as described in the *Changing Space Selection for a Project* section below.

7. Once you have entered all the mandatory fields, click *Next* or *Finish*.

A new project with the specified name is created in your workspace. The project contains the initial MTA descriptor. For example:



The screenshot shows the SAP Web IDE interface. On the left, there is a file tree with a 'Local' folder expanded, showing a 'HelloWorld' folder which is selected and highlighted in blue. Inside 'HelloWorld', there is an 'mta.yaml' file. On the right, the content of 'mta.yaml' is displayed in a code editor. The code is as follows:

```

mta.yaml x
1 schema-version: 2.0.0
2 ID: HelloWorld
3 version: 0.0.1

```

## Importing a Project from an Archive

You can import to the SAP Web IDE workspace a multi-target application project that was previously exported to an archive.

### Prerequisites

A .zip archive of the project that you want to import should be available in the file system.

### Procedure

1. In the workspace, select the top-level `Local` folder.
2. Choose `File > Import > From File System`.
3. Click `Browse` to locate and select your archived project file, and choose `Open`. The file name appears in the `File` field.

The destination folder name is displayed in the *Import to* field. By default, this name is the same as for the archive file. You can change the folder name, but not its location.

4. Choose *OK*. The project is created in the specified folder.

If an MTA descriptor (`mta.yaml` file) is present in the project's root folder and contains the definitions of modules, the corresponding subfolders are automatically converted into modules.

Otherwise, you need to check whether the module subfolders are successfully converted into modules. You can do this by choosing *Project Settings* from each module subfolder context menu, and verifying that the project type is one of the supported module types. If it is not, from the subfolder context menu, choose *Convert To*, and the type of the target module.

## Cloning a Project from Git

You can clone an existing project from a Git repository.

### Procedure

Follow the instructions in [Cloning Repositories \[page 1119\]](#).

### Related Information

[Inside an MTA Descriptor \[page 985\]](#)

[Customizing Your Project \[page 1093\]](#)

## Selecting a Space for a Project

You can change the space selection in SAP HANA XS Advanced for an existing, imported or cloned project.

### Procedure

1. From the project's context menu, choose ► *Project Settings* ► *Space* ▾.
2. From the *Space* dropdown list, select an appropriate space in SAP HANA XS Advanced for building and running your project.

## 14.4.4 Developing SAP HANA Database (HDB) Modules

An SAP HANA database (HDB) module is a collection of related design-time database artifacts, such as data models, views, or procedures.

To develop and deploy these artifacts into the SAP HANA database, perform the following steps:



- [Creating or Importing an HDB Module \[page 993\]](#)
- [Developing Database Artifacts \[page 994\]](#)
- [Inside an MTA Descriptor \[page 985\]](#)
- [Building an HDB Module \[page 1024\]](#)

### Module Folder Structure

The following figure depicts a sample HDB module folder structure alongside the corresponding entry in the `mta.yaml`.

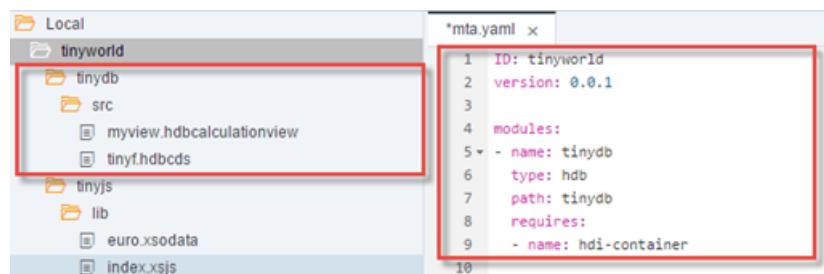


Table 364:

Folder	Description
<code>&lt;module name&gt;</code>	Should not contain any files.
<code>src</code> (default)	The default location in which you store your design-time database artifacts. If needed, you can create additional subfolders for the same purpose.

### Related Information

[Defining the Data Model in XS Advanced \[page 172\]](#)

## 14.4.4.1 Creating or Importing an HDB Module

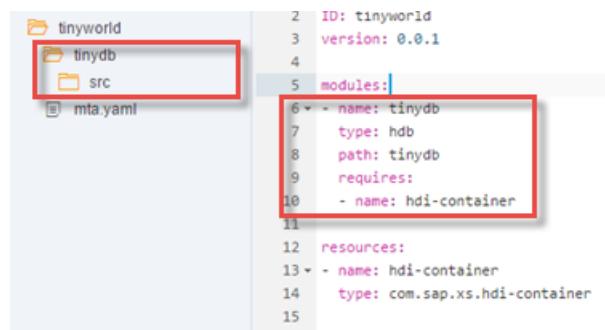
Create a new or import an archived SAP HANA database (HDB) module.

### Creating a New Module

#### Procedure

From the project context menu, choose **New > SAP HANA Database Module**, and follow the wizard steps to enter the module properties.

A new HDB module with the specified name is created in your project, and a corresponding section is added to the MTA descriptor (`mta.yaml`). For example:



```
2 ID: tinyworld
3 version: 0.0.1
4
5 modules:
6 - name: tinydb
7   type: hdb
8   path: tinydb
9   requires:
10   - name: hdi-container
11
12 resources:
13 - name: hdi-container
14   type: com.sap.xs.hdi-container
15
```

### Importing a Module from an Archive

#### Prerequisites

The `.zip` module archive that you want to import, which was exported from another MTA project, is available in the file system.

#### Procedure

1. From the root folder of the project, choose **File > Import > From File System**.
2. Click **Browse** to locate and select your archive, and choose **Open**. The file name appears in the **File** field. The destination folder is displayed in the **Import to** field. To change this folder, choose **Select Folder**, and browse to the required folder, or create a new folder.  
The specified folder, containing the artifacts extracted from the archive, is created in the project.
3. To make the imported folder a proper module in your project, you need to convert it into a module of the matching type. From the folder context menu, choose **Convert To**, and then the type of the target module.

### **i** Note

The conversion process does not check whether the imported folder structure matches the selected module type, and does not generate the module artifacts according to the selected type.

The imported module becomes a part of your MTA project, and the module entry is added to the MTA descriptor.

## 14.4.4.2 Developing Database Artifacts

Create the database artifacts required for your module.

### Procedure

From the context menu of the module's `src` subfolder, choose [New](#), and then choose one of the available artifacts:

Artifact	Instructions
<b>Database procedure (.hdbprocedure)</b>	Choose <a href="#">HDB Procedure</a> , and enter the file name. The new artifact is added to the module, and opens in a dedicated code editor.  For information about developing HDB procedures, see <a href="#">Defining the Data Model in XS Advanced [page 172]</a> .
<b>Calculation view (.hdbcalculationview)</b>	Choose <a href="#">Calculation View</a> . Enter a name, label, and select the type and category. The new artifact is added to the module, and opens in the chosen editor.  For information about developing calculation views, see <a href="#">Setting Up the Analytic Model [page 400]</a>
<b>A design-time definition of a calculation view, which can be referenced in an OData service definition.</b>	Use the Flowgraph Editor to model the transformation of your data. Choose <a href="#">Flowgraph</a> , and then enter a name. In the <a href="#">Flowgraph</a> editor that opens, you can place nodes onto the canvas, and configure the options to output the data you need. For example, you can aggregate, filter and join tables, or create a stored procedure.  For more information about creating flowgraphs, see <a href="#">Virtualizing Data Using Flowgraphs [page 1023]</a> .
<b>CDS (Core Data Services) Document (.hdcds)</b>	Choose <a href="#">CDS Artifact</a> , enter a name and choose a code snippet to add to the new file.  <b>i</b> Note The graphical editor only supports the <a href="#">Empty Context</a> snippet.  The new artifact is added to the module, and opens in the default editor defined in <a href="#">Tools</a> <a href="#">Preferences</a> <a href="#">Default Editor</a> <a href="#">HANA CDS Source</a> .  For information about developing CDS documents, see:

Artifact	Instructions
	<ul style="list-style-type: none"> <li>○ In the text editor: <a href="#">Create a CDS Document (XS Advanced) [page 208]</a></li> <li>○ In the graphical editor: <a href="#">Getting Started with the CDS Graphical Editor [page 996]</a> and <a href="#">Creating Synonyms [page 1020]</a>.</li> </ul> <p>Native datastore object (NDSO) in SAP HANA Datawarehousing Foundation (available with DWF 2.0) is represented as an annotated CDS context.</p> <p>For more information about the NDSO, see <a href="#">Native DataStore Object [page 1022]</a>.</p>
<b>Analytic privilege (.hdbanalyticprivilege)</b>	<p>Choose <a href="#">Analytic Privilege</a>, and enter a name and label. The new artifact is added to the module, and opens in the dedicated editor.</p> <p>For information about developing analytic privileges, see <a href="#">Setting Up the Analytic Model [page 400]</a>.</p>
<b>Text analysis dictionary (.hdbtextdict)</b>	<p>You can specify your own entity types and entity names to be used with text analysis by creating custom text analysis dictionaries. Choose <a href="#">Text Analysis Dictionary</a> and enter a name with or without the file extension. A text editor will open for the new dictionary file, with a XML template automatically populated.</p> <p>For more information about developing text analysis dictionaries, see <a href="#">Developing Text Analysis Artifacts [page 1023]</a>.</p>
<b>Text analysis rule set (.hdbtextrule)</b>	<p>You can specify your own entity types to be used with text analysis by creating custom text analysis extraction rules. Choose <a href="#">Text Analysis Rule Set</a> and enter a name with or without the file extension. A text editor will open for the new rule set file.</p> <p>For more information about developing text analysis rule sets, see <a href="#">Developing Text Analysis Artifacts [page 1023]</a>.</p>
<b>Text analysis configuration (.hdbtextconfig)</b>	<p>You can specify your own configuration settings to be used with text analysis. Choose <a href="#">Text Analysis Configuration</a> and enter a name with or without the file extension. Choose one of the standard SAP HANA configurations to use as a starting point. A text editor will open for the new configuration file, with a XML template automatically populated with the specified standard configuration settings.</p> <p>For more information about developing text analysis configurations, see <a href="#">Developing Text Analysis Artifacts [page 1023]</a>.</p>

### i Note

Currently, SAP Web IDE provides dedicated editors only for the artifacts listed above. To develop other artifacts, supported by SAP HANA XS Advanced (XSA), create a file in the module's `src` subfolder with an appropriate extension, and use a text editor.

## 14.4.4.2.1 Getting Started with the CDS Graphical Editor

SAP Web IDE for SAP HANA (XS advanced model) offers data modelers the graphical modeling capabilities that they require to model artifacts. These artifacts define the data persistence objects using the Core Data Services (CDS).

The CDS graphical editor in the SAP Web IDE for SAP HANA tool helps model data persistence objects in CDS artifacts using the DDL-compliant CDS syntax. The tool recognizes the `.hbcds` file extension required for CDS object definitions and calls the appropriate repository plug-in to parse the CDS artifact. If you right-click a file with the `.hbcds` extension in the *Workspace* view, SAP Web IDE for SAP HANA provides the following choice of editors in the context-sensitive menu.

- **Text Editor**  
View and edit DDL source code in a CDS document as text with the syntax elements highlighted for easier visual scanning.
- **Graphical Editor**  
View graphical representation of CDS artifacts, and also model CDS artifacts with graphical modeling tools.

This section describes how data modelers can use the graphical editor to model CDS artifacts. In the current version of the XS advanced model, you can use a graphical editor only to model the following important artifacts or elements:

- Entities (tables)
- Contexts (Containers)
- Associations
- User-defined data types
- Views (with unions, joins, calculated columns, and more.)

## Related Information

[Create an Artifact with the CDS Graphical Editor \[page 997\]](#)

[Create an Association with the CDS Graphical Editor \[page 1003\]](#)

[Create a User-Defined Structure Type with the CDS Graphical Editor \[page 1007\]](#)

[Create an Entity with the CDS Graphical Editor \[page 999\]](#)

[Create a User-Defined Scalar Type with the CDS Graphical Editor \[page 1009\]](#)

[Create a View with the CDS Graphical Editor \[page 1010\]](#)

## 14.4.4.2.1.1 Create an Artifact with the CDS Graphical Editor

A CDS artifact is a design-time source file that contains definitions of objects, which you want to create in the SAP HANA catalog. You can use graphical modeling tools to model a CDS artifact.

### Context

CDS artifacts are design-time source files that contain DDL code. The code describes a persistence model according to rules defined in Core Data Services. CDS artifacts have the file suffix `.hdbcds`. Building an SAP HANA Database Module that includes this CDS artifact, creates the corresponding catalog objects in the specified schema. To create a CDS artifact, do the following:

### Procedure

1. Start the SAP Web IDE for SAP HANA tool in a Web browser.
2. Display the application project to which you want to add a CDS artifact view.

In XS advanced, SAP Web IDE for SAP HANA creates an application within a context of a project. If you do not already have a project, there are a number of ways to create one, for example: by importing it, cloning it, or creating a new one from scratch.

- a. In the SAP Web IDE for SAP HANA, choose [File](#) [New](#) [Project from Template](#).
- b. Choose the project template type.

Currently, there is only one type of project template available, namely: [\*Multi-Target Application Project\*](#). Select [\*Multi-Target Application Project\*](#) and choose [\*Next\*](#).

- c. Type a name for the new MTA project (for example, `myApp`) and choose [\*Next\*](#) to confirm.
- d. Specify details of the new MTA project and choose [\*Next\*](#) to confirm.
- e. Create the new MTA project; choose [\*Finish\*](#).

3. Select the module in which you want to create the CDS artifact.
4. In the [\*Workspace\*](#) view, browse to the `src` folder.
5. Create a CDS artifact with context.

- a. Right-click the `src` folder, and choose [New](#) [CDS Artifact](#).
- b. In the [\*Name\*](#) field, enter the name of the CDS artifact.
- c. In the [\*Insert Snippet\*](#) dropdown list, select [\*Empty Context\*](#).

It is not necessary to insert the code snippet if your default editor for CDS is a graphical editor. In such cases the tool does not display the [\*Insert Snippet\*](#) dropdown list.

#### Note

In the CDS graphical editor for SAP Web IDE for SAP HANA, you can create CDS artifacts with contexts only. A context helps group related CDS artifacts, and you can create multiple contexts within a context (nested scenarios).

- d. Choose *Create*.

The tool opens a graphical editor that you can use to define the CDS artifact. You use the same editor to model CDS entities, contexts, associations, views, user-defined data types, and more within a CDS artifact.

 Note

You can define your preferred CDS editor (graphical or text) in ► *Tools* ► *Preferences* ► *Default Editor* ► *HANA CDS Source* ▾.

6. Create subcontexts, if required.

You can create nested subcontexts within a CDS artifact. Each of these subcontexts can contain any or multiple CDS artifacts such as entities, views, structure types, and more.

- a. In the editor toolbar, choose  (Create Context).
- b. Drop the context node on the editor.
- c. Provide a name to the context.
- d. Double-click the context node.

You can define new entities, associations, or another context within the subcontext. Use the breadcrumb navigation in the editor toolbar to switch between contexts.

7. (Optional) Publish CDS artifact as an OData service.

You can publish CDS artifacts at the context level as OData v4 services. You can also publish subcontexts as OData services.

In the editor toolbar, choose  (Publish as OData).

## Related Information

[Create an Entity with the CDS Graphical Editor \[page 999\]](#)

[Create an Association with the CDS Graphical Editor \[page 1003\]](#)

[Create a User-Defined Structure Type with the CDS Graphical Editor \[page 1007\]](#)

[Create a User-Defined Scalar Type with the CDS Graphical Editor \[page 1009\]](#)

[Create a View with the CDS Graphical Editor \[page 1010\]](#)

[Classification of Supported Data Types \[page 998\]](#)

## 14.4.4.2.1.1 Classification of Supported Data Types

You can use the CDS graphical tools to model a single CDS artifact that comprises of multiple entities, structure types, and more. When you are defining the elements in the entities or in the structure types, define data types of these elements.

The tool supports multiple data types for defining the data type of an element. For convenience, and based on its characteristics, the data types are classified as follows:

Table 365:

Type	Description
Primitive	Primitive types are standard Data Definition Language (DDL) data types such as String, Binary, or Integer.
Native	Native types are the native SAP HANA data types.
Structure Type	Structure types are user defined data types that comprises of a list of elements, each of which has its own data type.
Scalar Type	Scalar types are user define scalar data types. Unlike user-defined structure types, scalar types do not comprise of any elements in its definition.
Structure Element	Structure elements help reuse the data type of an element defined in a selected structure in another data type definition.
Entity Element	Entity elements help reuse the data types of an element defined in a selected entity in another data type definition.

#### 14.4.4.2.1.2 Create an Entity with the CDS Graphical Editor

Use the CDS graphical editor to model entities. In the SAP HANA database, as in other relational databases, a CDS entity is a table with a set of data elements that are organized using columns and rows.

#### Context

You model entities within a CDS artifact using the graphical modeling capabilities of the CDS graphical editor.

#### Procedure

1. Start the SAP Web IDE for SAP HANA tool in a Web browser.
  2. In the *Workspace* view, select the required CDS artifact within which you want to model the entity.
  3. In the context menu, choose  *Open With Graphical Editor*.
- The tool opens the CDS artifact in a graphical editor where you can model the entity.
4. Create an entity.
    - a. In the editor toolbar, choose  (Create Entity).
    - b. Drop the entity node on the editor.
    - c. Provide a name for the entity.
  5. Define the entity.

You can define one or more elements for the entity.

- a. Double-click the CDS entity node.
- b. In the editor toolbar, choose  (Add) to create a new element in the entity.
- c. In the **Name** field, you can modify the name of the element.
- d. For each element, in the **Type** dropdown list, select the required value.
- e. For each element, in the **Data Type** dropdown list, select the required value.

If you have selected the type of data type as **Structure Element** or **Entity Element**, then in the **Data Type** dropdown list, select the required structure or entity from within the same CDS artifact. Use the **Type of Element** value help list to select the required element.

- f. Define the length and scale based on the selected data type.
- g. Define whether the specified column is a primary key or part of the primary key for the specified entity. Select the **Key** checkbox accordingly.

#### Note

Structured columns can be part of the key, too. In this case, all table fields resulting from the flattening of this structured field are part of the primary key.

- h. Define if an entity column can (null) or cannot (not null) have the value NULL. If neither null nor not null is specified for the column, the default value null applies (except for the key column). Select the **Not Null** checkbox accordingly.
  - i. Define the default value for an entity column in the event that no value is provided during an INSERT operation. Enter a value in the **Default** text field.
6. (Optional) Use an expression to generate the element value at runtime.

The SAP HANA SQL clause generated always as `<expression>` is available for use in CDS entity definitions. This clause specifies the expression that the tool must use to generate the element value at runtime.

- a. Select the required element.
- b. In the **Expression** field, choose  (Element Expression Editor).
- c. In the **Expression** editor, enter the required expression.  
Use the elements and operators from the expression editor to build your expression.
- d. Choose **OK**.
- e. If you want to define the element with a generated expression, select the **Generated** checkbox.  
If you do not select the **Generated** checkbox, the tool considers the expression for the element as calculated expression. Generated elements are physically present in the database table; values are computed on `INSERT` and need not be computed on `SELECT`. Calculated elements are not actually stored in the database table; they are computed when the element is “selected”. Since the value of the generated column is computed on `INSERT`, the expression used to generate the value must not contain any non-deterministic functions, for example: `current_timestamp`, `current_user`, `current_schema` and more.

7. Create child element, if required.

For any selected element in the entity, you can create one or more child elements.

- a. Select the required element.
- b. In the editor toolbar, choose  (Create child).
- c. Provide a name for the child element.
- d. Define the data type and default values for the child element.

## 8. Import elements, if required.

When defining the elements in an entity, you can also do so by importing elements from other catalog tables. These catalog tables must be available in the same HDI container in which you are creating the entity. To reuse the definition of imported elements,

- a. In the editor toolbar, choose  (Import Element(s)).
- b. In the *Import Elements* wizard, search and select the required catalog table.

You can use the elements and its definition from the selected catalog table to define the elements of the entity.

- c. Choose *Next*.
- d. Select the elements you want to import.
- e. Choose *Finish*.

You can modify the definition of imported elements such as, its data type, length, scale, and more.

### Note

The data type of elements that you import from catalog tables, and the data type of the same elements in its design-time entity from which you generated the catalog table, need not be the same. It may vary or be the same depending on the data type defined for the elements.

## 9. Define additional properties.

- a. In the *Storage Type* dropdown list, select the required storage type value (row or column).  
If no storage type is specified, a “column” store table is generated by default.
- b. In the *Unload Priority* text field, specify the required unload priority value for the generated table.

Unload priority specifies the priority with which a table is unloaded from memory. The priority can be set between 0 (zero) and 9 (nine), where 0 means “cannot be unloaded” and 9 means “earliest unload”.

- c. Select the *Auto Merge* checkbox if you want to trigger an automatic delta merge.

### Note

Not selecting the *Auto Merge* checkbox disables the automatic delta merge operation.

- d. If you want to specify the grouping information for the generated tables, select the *Enable Table Grouping* checkbox and specify the group name, type and sub-type.

## 10. (Optional) Specify an identity column.

The SAP HANA SQL clause `generated as identity` is available for use in CDS entity definitions; if you are using an expression to generate the element value at runtime, this clause enables you to specify an identity column. An element that is defined with `generated as identity` corresponds to a field in the database table that is present in the persistence and has a value that is computed as specified in the sequence options defined in the `identity` expression, for example, `( start with 10 increment by 2 )`.

- a. In the menu bar, select the *Properties* tab.
- b. In the *Element* dropdown list, select an element from the entity that you want to use as identity column.
- c. In the *Type* dropdown list, select a value.

You can use either *always* or *by default*. If you select *always*, then values are always generated; if you select *by default*, then values are generated by default.

- d. Provide the *Start With* and *Increment by* values.  
For example, ( start with 10 increment by 2 ).
  - e. In the *Minimum Value* and *Maximum Value* dropdown list, select the required value and provide the minimum and maximum values.
  - f. In the *Cache* text field, provide a value.  
The cache value is typically the difference of maximum value and minimum value.
  - g. In the *Cycle* dropdown list, select a value.  
This value helps the tool determine whether it is a cycling sequence or non-cycling sequence.
11. Choose *Save* to save all your changes.
  12. Build an SAP HANA Database Module.
    - a. From the module context menu, choose *Build*.

## Next Steps

Some business cases may require performing additional tasks on an entity such as defining associations, creating partitions, and more. After creating and defining an entity, you can perform the following additional tasks within the entity.

Table 366:

Requirement	Task to Perform
Define relationship between entities, or between structure types and entities.	Create an Association
Partition the entity using elements in the entity.	Create a Partition
Specify indexes on entities	Create Indexes
Create entities to efficiently store series data	Create Entities to Store Series Data

## Related Information

[Create an Association with the CDS Graphical Editor \[page 1003\]](#)

[Create Indexes with the CDS Graphical Editor \[page 1004\]](#)

[Create a Partition with the CDS Graphical Editor \[page 1005\]](#)

[Create Entities to Store Series Data \[page 1006\]](#)

## 14.4.4.2.1.2.1 Create an Association with the CDS Graphical Editor

Associations define relationships between entities, or relationship between structure types and entities. You create associations in either a CDS entity definition or structure type definition, which are design-time files in SAP HANA.

### Context

The CDS graphical editor tool in SAP Web IDE for SAP HANA provides you graphical modeling capabilities to model associations between entities, or between structure types and entities. If you are creating associations between entities, then the associations are defined as part of the entity definition, which are design-time files in the repository. Similarly, if you are creating associations between structure types and entities, then the associations are maintained in the structure type definition, which are design-time files in the repository.

### Procedure

1. In the *Workspace* view, select the required CDS artifact.
2. In the context menu, choose  *Open With* .
3. Create associations between entities.
  - a. Select the required entity.
  - b. Choose  (Association).
  - c. Drag the cursor to another entity in the CDS artifact, with which you want to create associations.
4. Create associations between structure types and entities.
  - a. Select the required structure type.
  - b. Choose .
  - c. Drag the cursor to an entity in the CDS artifact, with which you want to create associations.
5. Provide association details.

In the *Association Details* dialog box, provide further details for the association.

  - a. In the *Name* text field, provide a name for the association.
  - b. Select the association type.

Select *Managed* or *Unmanaged* association type, depending on whether you want to create managed or unmanaged associations.

### **i** Note

For associations between structure types and entities, you can create only managed associations.

- c. If you are creating managed associations, select required elements from the structure type or entity as association keys.

For managed associations, the relation between source and target entity is defined by specifying a set of elements of the target entity that are used as a foreign key. If no foreign keys are explicitly specified, the elements of the target entity's designated primary key are used. In the *Alias* text field, provide an alias name, if required.

- d. If you are creating unmanaged associations, in the expression editor, provide the required condition.

Unmanaged associations are based on existing elements of the source and target entity; no fields are generated. In the *ON* condition, only elements of the source or the target entity can be used; it is not possible to use other associations. The *ON* condition may contain any kind of expression. Use the elements and operators from the editor, when defining the *ON* condition expression.

6. Define cardinality, if required.

When using an association to define a relationship between entities in a CDS artifact, use the cardinality to specify the type of relation, for example, one-to-one (to-one) or one-to-many (to-n); the relationship is with respect to both the source and the target of the association.

- a. In the *Source Cardinality* and *Target Cardinality* dropdown lists, select the required cardinality values.

7. Choose *OK*.

8. Choose *Save*.

### **i** Note

You can also create associations using a form-based editor. Double-click the required entity or structure type node in the CDS artifact, and in the Associations tab, create, and define the association.

## 14.4.4.2.1.2.2 Create Indexes with the CDS Graphical Editor

You can create indexes for an entity and specify an index type. The tool supports three types of indexes, plain indexes, full text indexes, and fuzzy search indexes.

### Procedure

1. In the menu bar, choose the *Indexes* tab.
2. Choose *+* (Add) to create a new index.
3. In the *General* section, select the required index type.

After selecting the index type, define which of the elements in the entity should be indexed. For the plain index type, you can also select the index order and use the *Unique* checkbox to define whether the index is unique (no two rows of data in the indexed entity can have identical key values). For *Full Text Index*, select the element and define the full text parameters such as the language column, mime type column, and more.

**i** Note

You cannot specify both a full-text index and a fuzzy search index for the same element.

4. Choose [Save](#).

### 14.4.4.2.1.2.3 Create a Partition with the CDS Graphical Editor

Use elements from the entity to partition the entity. You can use hash, range, or round-robin partition types to partition the entity.

#### Procedure

1. In the menu bar, choose the [Partitions](#) tab.
2. Select the [Add Primary Partition](#) checkbox.
3. In the [Partition Type](#) dropdown list, select a partition type.

**i** Note

For [Hash](#) or [Roundrobin](#) partition types, you can also define the secondary partition type. In the [Secondary Partition](#) section, select the required secondary partition type.

4. Choose [+](#) (Add) to add an element.
5. In the [Element](#) dropdown list, select an element, which you want to use to partition the data.
6. In the [Functions](#) dropdown list, select a value. This is applicable if any of the selected element used for partitioning the entity represents time values.
7. In the [Partitions](#) text field, enter the number of partitions required.

**i** Note

If you want to partition the entity based on number of servers, select the [Number of Servers](#) checkbox

8. Choose [Save](#).

## 14.4.4.2.1.2.4 Create Entities to Store Series Data

Use the CDS graphical editor in SAP Web IDE for SAP HANA to create entities that can efficiently store series data.

### Procedure

1. In the menu bar, choose the *Series* tab.
2. Select the *Enable Series* checkbox.
3. In the *Series Key Element(s)* dropdown list, select one or more elements.
4. If you want to create an entity to store time series data, in the *Period for Series* dropdown list, select the required period column.

Each row of a series table has a period of validity. The period represents the period in time that the row applies to. When the series table represents instants, then there is a single period column. When the table represents intervals, there can be one or two columns.

#### i Note

The period for series must be unique and should not be affected by any shift in timestamps. If you want to create an entity to store non-time series data, select the *Null* checkbox.

5. Select the required *Equidistant Type*.

Table 367:

Type	Description
Not Equidistant	If you want to create an entity to store arbitrary time values without any regular pattern.
Equidistant	If you are creating an entity to store time values with regular pattern. For equidistant series, provide the <i>Increment By</i> value that defines the distance between adjacent elements in an equidistant series.  The equidistant series tables offer improved compression compared to non-equidistant tables.

Type	Description
Equidistant Precisewise	<p>If you want to create an entity to store series data that retains a degree of regularity without being equidistant across the entire dataset.</p> <p>Equidistant piecewise data consists of regions where equal increments exist between successive points, but these increments are not always consistent for all regions in the table. Different series or different parts of a series may have different intervals.</p> <p>For equidistant precisewise series, you use only one period column, but you can specify one or more <i>Alternate Period for Series</i>. These columns can be used to record the time for each row, or represent the end of an interval associated with each row. Good compression can be achieved for these columns when successive values within the series differ by a constant amount.</p>

#### 6. (Optional) Provide minimum and maximum values.

You can define equidistant and non-equidistant series tables with a minimum and/or maximum value. These values are used to verify that loaded data corresponds to the range definition. The values in *Period for Series* column must satisfy the minimum values and the maximum value constraint.

- In the *Min Value* dropdown list, select the *Min Value* menu option and provide the required minimum value.
- In the *Max Value* dropdown list, select the *Max Value* menu option and provide the required maximum value.

##### Note

If the *Period for Series* column does not have a lower limit and upper limit, you can use the *No Min Value* and *No Max Value* respectively.

### 14.4.4.2.1.3 Create a User-Defined Structure Type with the CDS Graphical Editor

A structured type is a user-defined data type comprising a list of attributes, each of which has its own data type. You create a user-defined structured type as a design-time file in the SAP HANA repository.

#### Context

Use the CDS graphical editor to create a user-defined structured type as a design-time file in the repository. The attributes of the structured type can be defined manually in the structured type itself and reused by another structured type (or by scalar type, or by an entity). You create a user-defined structure types within a CDS artifact.

## Procedure

1. Start the SAP Web IDE for SAP HANA tool in a Web browser.
  2. In the [Workspace](#) view, select the required CDS artifact within which you want to model the entity.
  3. In the context menu, choose  [Open With > Graphical Editor](#).
- The tool opens the CDS artifact in a graphical editor where you can create the structure type.
4. Create a structured type.
    - a. In the editor toolbar, select  (Create Structure).
    - b. Drop the structure node on the editor.
    - c. Provide a name to the structure.
  5. Define the elements.
    - a. Double-click the structure type node.
    - b. In the editor toolbar, choose  (Add) to create a new element in the structure.
    - c. In the [Name](#) field, you can modify the name of the element.
    - d. For each element, in the [Type](#) dropdown list, select the required value.
    - e. For each element, in the [Data Type](#) dropdown list, select the required value.

If you have selected the type of data type as [Structure Element](#) or [Entity Element](#), then in the [Data Type](#) dropdown list, select the required structure or entity in the CDS artifact. Use the [Type of Element](#) value help list to select the required element.

- f. Define the length and scale based on the selected data type.
6. Create child element, if required.

For any element in the structure, you can also create one or more child elements.

- a. Select the required element.
- b. In the editor toolbar, choose  (Create child).
- c. Define the data type for the child element.

### Note

When you create a child element, the tool automatically changes the type of data type of its parent element to [Inline](#).

7. Import elements, if required.

When defining the elements in a structure types, you can also do so by importing elements from other catalog tables. These catalog tables must be available in the same HDI container in which you are creating the entity. To reuse the definition of imported elements,

- a. In the editor toolbar, choose  (Import Element(s)).
- b. In the [Import Elements](#) wizard, search and select the required catalog table.

You can use the elements and its definition from the selected catalog table to define the elements of the structure type.
- c. Choose [Next](#).
- d. Select the elements you want to import.
- e. Choose [Finish](#).

You can modify the definition of imported elements such as its data type, length, scale, and more.

8. In the editor toolbar, choose  (Navigate Back).
9. Choose [Save](#).

## 14.4.4.2.1.4 Create a User-Defined Scalar Type with the CDS Graphical Editor

Scalar types are user-defined scalar data types that does not comprise of any elements in its definition. You create a user-defined structured type as a design-time file in the SAP HANA repository.

### Context

Use the CDS graphical editor to create a user-defined structured type as a design-time file in the repository. You create a user-defined scalar data types within a CDS artifact. After creating a scalar type, you can reuse it when defining data types of elements in other structure types, scalar types, or entities.

### Procedure

1. Start the SAP Web IDE for SAP HANA tool in a Web browser.
2. In the [Workspace](#) view, select the required CDS artifact within which you want to model the entity.
3. In the context menu, choose  [Open With Graphical Editor](#).

The tool opens the CDS artifact in a graphical editor where you can create the scalar type.
4. Create a scalar type.
  - a. In the editor toolbar, select  (Create Scalar type).
  - b. Drop the scalar type node on the editor.
  - c. Provide a name to the scalar type.
5. Define the scalar type.
  - a. Double-click the scalar type node.
  - b. For each element, in the [Type](#) dropdown list, select the required value.

The data types are classified as primitive data types, native SAP HANA data types, user-defined data types (structure types and scalar types), entity element and structure element. Based on your requirement you can select a value from any of the above data type classifications.
  - c. In the [Data Type](#) dropdown list, select the required value.

If you have selected the type of data type as [Structure Element](#) or [Entity Element](#), then in the [Data Type](#) dropdown list, select the required structure or entity in the CDS artifact. Use the [Type of Element](#) value help list to select the required element.
6. In the editor toolbar, choose  (Navigate Back).

## 14.4.4.2.1.5 Create a View with the CDS Graphical Editor

Use the graphical modeling capabilities of the CDS graphical editor tool in SAP Web IDE for SAP HANA to create a design-time CDS view.

### Context

A view is a virtual table based on the dynamic results returned in response to an SQL statement. You use the graphical modeling tools to create a view with the CDS artifact. After creating the CDS view, use this design-time view definition to generate the corresponding catalog object when you deploy the application that contains the view-definition artifact (or just the application's database module).

### Procedure

1. Start the SAP Web IDE for SAP HANA tool in a Web browser.
2. In the *Workspace* view, select the required CDS artifact within which you want to model a CDS view.
3. In the context menu, choose  *Open With Graphical Editor*.

The tool opens the CDS artifact in a graphical editor where you can create and define the CDS view.

4. Create a CDS view.
  - a. In the editor toolbar, select  (Create View).
  - b. Drop the view node on the editor.
  - c. Provide a name to the view.

5. Add local CDS artifacts as data sources.

You can add local CDS artifacts, active CDS artifacts, or both as a data source in a view node. Local CDS artifacts are those entities that exist within the same artifact in which you are creating the CDS view. Active CDS artifacts are those artifacts that are already built and activated. The active artifacts could also be synonyms pointing to artifacts in other HDI containers.

- a. Double-click the view node.
- b. If you want to add a local CDS artifact, in the editor toolbar, select  (Add Local Objects).
- c. Select the required entities that you want to add as data sources.
- d. Choose *OK*.

6. Add active CDS artifacts as data sources.

Active CDS artifacts are those artifacts that are already built and activated. The active artifacts could also be synonyms pointing to artifacts in other HDI containers.

- a. Double-click the view node.
- b. If you want to add an active CDS artifact, in the editor toolbar, select  (Add Active Objects).
- c. In the *Find Data Source* dialog box, search and add activated artifacts from within the same HDI container, or search and add synonyms pointing to artifacts in other HDI containers.

The active artifacts include built and activated entities (catalog tables), CDS views, calculation views, and SQL views.

### Note

You cannot use calculation views with input parameters as data sources in CDS views.

- d. Choose *Finish*.
- e. Select the required menu option.

### Note

If you have more than one data source in the CDS view, either create a union, or a join between the data sources.

## 7. (Optional) Using elements from associated entities.

You can model a CDS view with entities, which are defined with associations to other entities. In such cases, you can use the elements from the associated entities in the CDS view definition. For using elements from the associated entities, it is not necessary to use associated entities as data sources in the CDS view.

- a. Select the required entity.
- b. Select the association column in the entity.
- c. In the *Select Element(s)* dialog box, select the required elements from the associated entity.

You can use the elements from the associated entities in the CDS view definition. For example, as output columns, to define SQL clauses, perform SQL GROUP BY or ORDER BY operations, and more.

- d. (Optional) Define a filter expression.

In the *Filter Expression* editor, provide a filter expression rule that defines how the elements from the associated entities are used.

When following an association (for example, in a view), it is possible to apply a filter condition; the filter is merged into the ON-condition of the resulting JOIN. The following example shows how to get a list of customers and then filter the list according to the sales orders that are currently “open” for each customer. In the example, the infix filter is inserted after the association orders to get only those orders that satisfy the condition `status='open'`.

### Sample Code

```
view C1 as select from Customer {  
    name,  
    orders[status='open'].{id as orderId, date as orderDate}  
};
```

Use the elements from the associated entities and the operators provided in the dialog box to build your filter expression.

- e. Choose *OK*.

## 8. (Optional) Using elements from structure types.

You can model a CDS view with an entity that has a structure type. In such cases, you can use elements from the structure types in the CDS view definition.

- a. Select the required entity.
- b. Select the structure type element in the entity.
- c. In the *Select Element(s)* dialog box, select the required elements from the structure type.

You can use the elements from the structure types in the CDS view definition. For example, as output columns, to define SQL clauses, perform SQL GROUP BY or ORDER BY operations, and more.

- d. Choose *OK*.
9. Define output columns.
  - a. In the CDS view editor, select the required data source.
  - b. Select the columns from the data source that you want to add to the output.
  - c. In the context menu, choose *Add to Output*.
  - d. In the *Select Element(s)* dialog box, provide an alias name to the output column, if required.
  - e. Choose *OK*.

The tool displays the output columns of the view under the *Columns* section (in the details pane) of the editor.
10. (Optional) Group by result set.

Use one or more columns from the data source to perform a SQL GROUP BY operation.

  - a. Select the data source.
  - b. Select the required columns.
  - c. In the context menu, choose *Add to Group By* to perform a SQL GROUP BY operation with the selected columns.

You can also use an output column to perform a SQL GROUP BY operation. In the *Columns* section, select an output column, and in the context menu, choose *Add to Group By*.
11. (Optional) Order by result set.

Use one or more output columns from the view to perform a SQL ORDER BY operation.

  - a. In the details pane, expand the *Columns* section.
  - b. Select the required columns.
  - c. In the context menu, choose *Add to Order By* to perform a SQL ORDER BY operation with the selected output columns.
12. (Optional) Limit and offset the result set.

You can use the SQL LIMIT clause and the SQL OFFSET clause in a CDS view. In the details pane, specify the required limit value and offset value as integers.

  - a. In the *Limit* text field, provide the required limit value.

Use the LIMIT clause to restrict the number of result sets to a specified limit.
  - b. In the *Offset* text field, provide an integer value required offset value.

Use the OFFSET clause to specify the number of records to skip before displaying the results sets defined by the LIMIT SQL clause.
13. (Optional) Create Subqueries.

You can model CDS views with nested SQL queries (subqueries) to obtain the desired output. Subqueries help model CDS views for complex business scenarios.

  - a. Double-click the CDS view.
  - b. In the editor toolbar, choose  (Sub Query).
  - c. Drop the subquery node on the editor.
  - d. Provide a name to the subquery node.
  - e. Double click the subquery node.
  - f. Define the subquery.

Defining a subquery is similar to defining a view. The current version of the CDS graphical editor supports only defining entities in a subquery.

➔ Tip

You can use the breadcrumb navigation in the editor toolbar to switch between subqueries and the target CDS view.

14. Choose [Save](#) to save all your changes.

15. Build an SAP HANA Database Module.

The build process uses the design-time database artifacts to generate the corresponding actual objects in the database catalog. The activation process generates a corresponding catalog object for each of the artifacts defined within another artifact; the location in the catalog is determined by the type of object generated.

a. From the module context menu, choose [Build](#).

## Next Steps

Modeling CDS views for complex business scenarios could include layers of calculation logic. In such cases, it may require to perform certain additional tasks to obtain the desired output.

The following table lists some important additional tasks that you can perform to enrich the CDS view.

Table 368:

Requirement	Task to Perform
If you want to query data from two data sources and combine records from both the data sources based on a join condition.	Create Joins
If you want to combine the results of two more data sources.	Create Unions
If you want to create new output columns and calculate its values at runtime using an expression.	Create Calculated Columns
If you want to define relationships between CDS views.	Create Associations for CDS Views with CDS Graphical Editor
If you want to create CDS views with parameters that enable you to pass additional values to modify the results of the CDS view at runtime.	Create Parameters

## Related Information

[Create SQL WHERE Clause \[page 1014\]](#)

[Create SQL HAVING Clause \[page 1014\]](#)

[Create Calculated Columns \[page 1015\]](#)

[Create Joins \[page 1015\]](#)

[Create Unions \[page 1017\]](#)

[Creating Synonyms \[page 1020\]](#)

[Create Associations for CDS Views with CDS Graphical Editor \[page 1018\]](#)

[Create Parameters \[page 1019\]](#)

## 14.4.4.2.1.5.1 Create SQL WHERE Clause

Define SQL WHERE clause in a view to filter and retrieve records from the output of a data source based on a specified condition.

### Procedure

1. Open the required CDS artifact.
2. Select the CDS view in which you want to define the SQL WHERE clause.
3. Double-click the CDS view.
4. In the *Where* section, choose **+** (Add Where Clause).
5. In the *Where Clause Editor*, define the SQL WHERE clause condition using the required elements and operators.  
You can also use parameters in the expression.
6. Choose *OK*.
7. Choose *Save*.

## 14.4.4.2.1.5.2 Create SQL HAVING Clause

Define SQL HAVING clause in a view to retrieve records from the output of a data source, only when the aggregate values satisfy a defined condition.

### Procedure

1. Open the required CDS artifact.
2. Select the CDS view in which you want to define the SQL HAVING clause.

#### Note

You can create SQL HAVING clause only if you are using one or more output columns to perform a SQL GROUP BY operation on the selected view.

3. Double-click the CDS view.
4. In the *Having* section, choose **+** (Add Having Clause).

5. In the *Having Clause Editor*, define the SQL `HAVING` clause condition using the required elements and operators.  
You can also use parameters in the `HAVING` clause condition.
6. Choose *OK*.
7. Choose *Save*.

### 14.4.4.2.1.5.3 Create Calculated Columns

Create new output columns for a view and calculate the column values at runtime based on the result of an expression. You can use other column values, functions, or constants when defining the expression.

#### Procedure

1. Open the required CDS artifact.
2. Select the CDS view in which you want to create calculated columns.
3. Double-click the CDS view.
4. In the details pane, select the *Columns* section.
5. Choose *+* (Add Calculated Column).
6. Provide a valid expression.

In the *Add Calculated Column* dialog box, provide the expression that defines the calculated column.

- a. In the *Name* field, provide a name for the calculated column.
- b. In the expression editor, provide a valid expression using the required elements and operators.  
You can also use parameters in the expression.

- c. Choose *OK*.

### 14.4.4.2.1.5.4 Create Joins

Joins help query data from two or more data sources. It helps to limit the number of records, or to combine records from both the data sources, so that they appear as one record in the query results.

#### Context

After modeling a CDS view, you can include a `JOIN` clause in the CDS view definition.

## Procedure

1. Open the required CDS artifact.
2. Select the CDS view in which you want to create a join for data sources.
3. Double-click the CDS view.

### Note

You can create a join only if the CDS view has two or more data sources.

4. Select an entity.



5. Choose (Join).
6. Drag the cursor to the required data source.
7. Define the join properties.

In the *Join Definition* dialog box, define the join properties and the join condition.

- a. In the *Type* dropdown list, select the required join type.
- b. In the expression editor, specify a valid join condition using the necessary element and operators.

Using proposed join condition.

The tool analyzes the data in the participating entities and proposes a join condition. Based on your requirement, you can create your own join condition or use the join condition that the tool proposes. In the *Join Definition* dialog box, choose *Propose Condition* to use the join condition that the tool proposes.

- c. Choose *OK*.

## Related Information

[Supported Join Types \[page 1016\]](#)

### 14.4.4.2.1.5.4.1 Supported Join Types

When creating a join between two data sources, you specify the join type. The following table lists the supported join types in CDS graphical editor.

Table 369:

Join Type	Description
Inner	This join type returns all rows when there is at least one match in both the data sources.
Left Outer	This join type returns all rows from the left data source, and the matched rows from the right data source.

Join Type	Description
Right Outer	This join type returns all rows from the right data source, and the matched rows from the left data source.
Full Outer	This join type displays results from both left and right outer joins and returns all (matched or unmatched) rows from the tables on both sides of the join clause.

## 14.4.4.2.1.5.5 Create Unions

Creating unions help combine the result sets of two or more data sources.

### Context

After modeling a CDS view using the graphical modeling tools, you can include a `UNION` clause in the CDS view definition.

### Procedure

1. Open the required CDS artifact.
2. Select the CDS view in which you want to perform the union operation.
3. Double-click the CDS view.

**i Note**

You can create a union only if the CDS view has two or more data sources.

4. Select a data source.
5. Choose  (Union).
6. Drag the cursor to the required data source.

This operation creates a `ResultSet` node for each of the data sources.

7. Add more data sources to the union.
- You can perform union operation on multiple data sources (more than two data sources). If you have already created a union of two data sources, then to create a union of these two data sources with the third data source, you use the `ResultSet` node.
- a. Select a `ResultSet` node.
- You cannot select the `DefaultResultSet` node to union records of multiple data sources.
- b. Choose  (Union).

- c. Drag the cursor to the required data source.
8. Add columns to the union output.

The columns you add to the *DefaultResultSet* node are the output columns of the union.

- a. Select the data source.
- b. Select the columns in the data source you want to add to the output.
- c. In the context menu, choose *Add to Output*.

#### Note

Union operation combine result sets of two or more SELECT statements. It is necessary that each result set (SELECT statements) has the same number of columns, have similar data types, and also the columns in each result sets are in same order.

## 14.4.4.2.1.5.6 Create Associations for CDS Views with CDS Graphical Editor

Associations define relationships between CDS views, and you create association in the CDS view definition.

### Context

In a CDS view definition, associations can be used to gather information from the specified target entities. You can use the CDS graphical editor to create associations only if the views are simple CDS views. This means that, the CDS views involved in the association must contain only a single entity as a data source.

### Procedure

1. In the *Workspace* view, select the required CDS artifact.
2. In the context menu, choose  *Open With Graphical Editor*.
3. Create associations between views.
  - a. Select the required CDS view.
  - b. Choose  (Association).
  - c. Drag the cursor to another CDS view in the CDS artifact with which you want to create the association.
4. Provide association details.
  - a. In the *Association Details* dialog box, provide further details for the association.
  - b. In the *Name* text field, provide a name for the association.
  - c. (Optional) If you want the tool to propose a condition for the association, choose *Propose Condition*.
  - d. In the expression editor, provide the required ON condition.

In the **ON** condition, only elements of the source or the target CDS view can be used; it is not possible to use other associations. The **ON** condition may contain any kind of expression. Use the elements and operators from the dialog box, when defining the **ON** condition expression.

5. Define cardinality, if required.
  - a. When using an association to define a relationship between CDS views in a CDS artifact, use the cardinality to specify the type of relation, for example, one-to-one (to-one) or one-to-many (to-n); the relationship is with respect to both the source and the target of the association.
  - b. In the *Source Cardinality* and *Target Cardinality* dropdown lists, select the required cardinality values.
6. Choose *OK*.
7. Choose *Save*.

#### 14.4.4.2.1.5.7 Create Parameters

Use the CDS graphical editor in SAP Web IDE for SAP HANA to define CDS views with parameters. The parameters enable you to pass additional values to modify the results of the CDS view at runtime.

#### Context

After creating a parameter, you can use the parameter in the CDS view as an output column, in calculated column expressions, in the SQL `WHERE` clause, in the SQL `Group By` clause, joins, unions, and more.

#### Procedure

1. Open the required CDS artifact.
2. Select the CDS view in which you want to create parameters.
3. Double-click the CDS view.
4. In the details pane, choose **+** in the *Parameters* section.
5. Define the parameter.

In the *Parameter Properties* dialog box, define the parameter.

- a. In the *Name* text field provide a name for the parameter.
- b. In the *Category* dropdown list, select the required category.
- c. In the *Type* dropdown list, select the required data type.

If you have selected the type of data type as *Structure Element* or *Entity Element*, in the *Type* dropdown list, select the required structure or entity from within the same CDS artifact. Use the *Type Of* value help list to select the required element.

- d. Define the *Length* and *Scale* based on the selected data type.
- e. Choose *OK*.
6. If you want to add the parameter as an output column of the CDS view, right-click the parameter and choose *Add As Output Column*.

## 14.4.4.2.2 Creating Synonyms

Synonym are design time files in the repository that provides data independence. After creating a synonym, you can bind this synonym with tables in different schemas.

This means that, in XS advanced, irrespective of which user or which schema holds the table, you can continue using the applications without any modifications. The complete definition of a synonym is contained in two design time files:

- `hdbsynonym` - A synonym declaration (with an optional default configuration).
- `hdbsynonymconfig` - An explicit configuration of the synonym's target.

This section describes creating the above two synonym definition files without writing any synonym definition code and the prerequisites to creating such a database synonym.

### Related Information

[Create a Database Synonym \[page 1020\]](#)

### 14.4.4.2.2.1 Create a Database Synonym

Use the synonym editor in SAP Web IDE for SAP HANA (XS Advanced) tool to create and define synonyms without writing any synonym-definition code.

### Prerequisites

- You have defined a user provided service with an underlying database user. The user must have credentials to access all schemas required in the project. You use the following syntax to define the user provided service.

#### Sample Code

```
xs create-user-provided-service <SERVICE_INSTANCE> -p "{\"host\":\n  \"<host_name>\", \"port\":<port_number>, \"user\":\"<user_name>\",\n  \"password\":\"<USER_PASSWORD>\", \"tags\":[\"hana\"] }"
```

- You have correctly configured all services including the user provided service in the `mta.yaml` file. This allows using the user provided service within the project. A sample `mta.yaml` configuration is shown below.

#### Sample Code

```
ID: DwProject\nversion: 1.2.3\nmodules:\n  - name: DwDbModule
```

```

type: hdb
requires:
  - name: DwHdiService
    properties:
      TARGET_CONTAINER: ~{hdi-service-name}
  - name: CrossSchemaService
resources:
  - name: DwHdiService
    type: com.sap.xs.hdi-container
    properties:
      hdi-service-name: ${service-name}
  - name: CrossSchemaService
    type: org.cloudfoundry(existing-service
    parameters:
      service-name : CROSS_SCHEMA_SERVICE_SYSTEM_ERP

```

- You have created a grantor file (.hdbgrants) that specifies the necessary privileges to access external tables.

### Sample Code

```
{
  "CROSS_SCHEMA_ACCESS_SERVICE_ERP": {
    "object_owner" : {
      "schema_privileges": [
        {
          "reference": "ERP",
          "privileges_with_grant_option": ["SELECT"]
        }
      ]
    },
    "application_user" : {
      "object_privileges": [
        {
          "schema": "ERP",
          "name": "TABLE",
          "privileges": ["SELECT"]
        }
      ]
    }
  }
}
```

## Procedure

1. Start the SAP Web IDE for SAP HANA tool in a browser.
2. Display the application project to which you want to add a calculation view.

In XS Advanced, SAP Web IDE for SAP HANA creates an application within a context of a project. If you do not already have a project, there are a number of ways to create one, for example: by importing it, cloning it, or creating a new one from scratch.

- a. In the SAP Web IDE for SAP HANA, choose   .
- b. Choose the project template type.

Currently, there is only one type of project template available, namely: *Multi-Target Application Project*. Select *Multi-Target Application Project* and choose *Next*.

- c. Type a name for the new MTA project (for example, `myApp`) and choose *Next* to confirm.
  - d. Specify details of the new MTA project and choose *Next* to confirm.
  - e. Create the new MTA project; choose *Finish*.
3. Select the HDB module in which you want to create synonyms.
  4. Browse to the `src` folder.
  5. Create a `.hdbsynonym` file.
    - a. In the context menu of the `src` folder, choose ► *New* ► *File* ▶.
    - b. Provide a name for the synonym declaration file.

**i Note**

You must add `.hdbsynonym` as a suffix to the file name.

- c. Choose *Ok*.  
The tool opens a new synonym editor where you can declare all the synonyms that you require.
- d. Choose .
- e. In the *Find Data Sources* dialog box, search the required data source for which you want to declare a synonym.  
You can declare multiple synonyms in a single file.

**i Note**

You can also directly enter the name of the data source in the *Object Name* text field.

- f. Choose *Save*.
6. Build an HDB module.  
The build process uses the design-time database artifacts to generate the corresponding actual objects in the database catalog.
  - a. From the module context menu, choose ► *Build* ► *Build* ▶.

## Related Information

[HDI Artifact Types and Build Plug-ins Reference \[page 785\]](#)

### 14.4.4.2.3 Native DataStore Object

A native datastore object is a specialized CDS artifact that allows merging of data sets, including delta sets, into their reportable content.

This can be achieved with a simple move or aggregation and it's also possible to mark records in the delta loads for deletion. In addition, you can identify and process the delta data set for connected NDSOs. The

object also provides interoperability between native Data Warehouses and SAP BW/4HANA. The native datastore object is only available with DWF 2.0 or higher.

For more information about creating an NDSO, see the *SAP HANA Data Warehousing Foundation - Native DataStore Object Administration Guide* → *Creating Native DataStore Object*.

## Related Information

<https://uacp2.hana.ondemand.com/viewer/1e4f857a22aa477081d41d3b6fa48d99/2.0.0.0/en-US>

### 14.4.4.2.4 Virtualizing Data Using Flowgraphs

Use the Flowgraph Editor to model the transformation of your data.

You can better virtualize data by transforming it using a flowgraph editor and various transforming nodes. Use the nodes to design a flowgraph to retrieve data from an external system, transform it, and view it.

The Flowgraph Editor helps model the transformation of your data through various node such as Join, Projection (Filter), Union, Aggregation, and so on.

For more information about data virtualization using the Flowgraph Editor, see *SAP HANA Administration Guide* → *Virtualizing Data*.

#### i Note

The SAP HANA smart data integration and SAP HANA smart data quality options add much more functionality to flowgraph design, such as data cleansing, geocoding, masking, date and row generation, and so on, as well as many capabilities outside of flowgraphs. Be aware that you need additional licenses for SAP HANA options and capabilities. For more information, see [Important Disclaimer for Features in SAP HANA Platform and Options \[page 1136\]](#).

### 14.4.4.2.5 Developing Text Analysis Artifacts

Text analysis allows you to discover and classify entities and relationships in your documents and unstructured text content. You can specify your own entity types to be used with text analysis by creating custom text analysis dictionaries and extraction rules.

### 14.4.4.3 Building an HDB Module

The build process uses the design-time database artifacts to generate the corresponding actual objects in the SAP HANA database catalog.

#### Procedure

From the module context menu, choose *Build*.

The steps and messages of the build process are displayed in the console that opens automatically at the bottom of the browser window. You can show and hide the console by choosing *View* *Console* from the main menu.

##### Note

Sometimes the build fails because there is an error in the MTA descriptor (mta.yaml). The error messages displayed in the console indicate the corrections you should make in the descriptor. For more information about the descriptor syntax, see [Inside an MTA Descriptor \[page 985\]](#).

After making the corrections, rebuild the module.

By default, the build process generates a database schema with a unique name derived from the `schema` parameter defined in `mta.yaml`. If you want the schema name to be exactly the same as defined, set the `makeUniqueName` parameter to `false`.

For example:

##### Sample Code

```
- name: hdi-container
  parameters:
    config:
      schema: mySpecialSchema
      makeUniqueName: false
  properties:
    hdi-container-name: ${service-name}
  type: com.sap.xs.hdi-container
```

However, if you do so and use the same schema name in different projects, naming conflicts might occur in the database during builds.

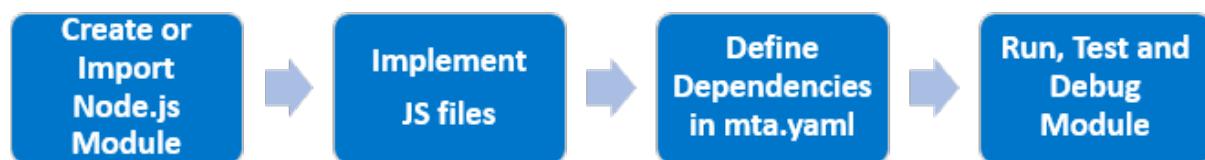
#### Related Information

[Developing SAP HANA Database \(HDB\) Modules \[page 992\]](#)

## 14.4.5 Developing Node.js Modules

A Node.js module is a collection of related JavaScript files and service definitions that implement the business logic of your application.

To develop Node.js modules, perform the following steps:



- Creating or Importing a Node.js Module [page 1026]
- Implementing Node.js files [page 1026]
- Inside an MTA Descriptor [page 985]
- Running and Testing Node.js Modules [page 1028]

### Module Folder Structure

The following figure depicts a sample Node.js module folder structure alongside its corresponding entry in the `mta.yaml`.

The screenshot shows a file explorer window with a sidebar and a main content area. The sidebar lists several subfolders: `tinyworld`, `tinydb`, `src`, `tinyjs`, `lib`, `euro.xsodata`, `index.xsjs`, `test`, `sampleTest.xsjslib`, `package.json`, `server.js`, `testrunner.js`, `tinyui`, `resources`, `index.html`, `package.json`, `xs-app.json`, and `mta.yaml`. The `tinyjs` folder and its contents are highlighted with a red box. The main content area shows the `mta.yaml` file with the following content:

```
*mta.yaml* x
1 ID: tinyworld
2 version: 0.0.1
3
4 modules:
5 - name: tinydb
6   type: hdb
7   path: tinydb
8   requires:
9     - name: hdi-container
10
11 - name: tinyjs
12   type: nodejs
13   path: tinyjs
14 # ----- dependency on DB
15   requires:
16     - name: tinydb
17     - name: hdi-container
18 #----- exposes SERVICE URL to consumers
19   provides:
20     - name: tinyjs_api
21       properties:
22         service_url: ${default-url}
23
```

Table 370:

Folder	Description
<code>&lt;module name&gt;</code>	Contains the main .js file, which by default is <code>server.js</code> , and the <code>package.json</code> file (description and dependencies of Node.js modules). You can place other .js files anywhere in this folder.

Folder	Description
<i>lib</i>	Created only if XSJS support is enabled for the module. Contains .xsjs (SAP HANA XS JavaScript) source files.
<i>test</i>	Created only if XSJS support is enabled for the module. Contains XSUnit test (.xsjslib) files.
<i>tests</i>	This folder is created for modules without XSJS support. It contains .js test files.

**i Note**

A manual adjustment is required for any Node.js module with XSJS support that you created in SAP Web IDE SPS11. In such modules replace the package.json with the package.json created from the template in SPS 12

## Implementing Node.js files

In the root module folder, create and implement the required .js files.

## Related Information

[Developing Multi-Target Applications \[page 983\]](#)

<https://nodejs.org/en/> ↗

[Node.js Data Services in XSJS Compatibility Mode \[page 570\]](#)

### 14.4.5.1 Creating or Importing a Node.js Module

You can create a new or import an archived Node.js module.

## Creating a New Module

### Procedure

From the project context menu, choose **New > Node.js Module**, and follow the wizard steps to enter the module properties.

Select *Enable XSJS support* to run your module in the XS JavaScript (XSJS) compatibility mode.

A new Node.js module with the specified name is created in your project, and a corresponding section is added to the MTA descriptor (`mta.yaml`). For example:

```

2 ID: tinyworld
3 version: 0.0.1
4
5 modules:
6 - name: tinydb
7   type: hdb
8   path: tinydb
9   requires:
10     - name: hdi-container
11
12 - name: tinyjs
13   type: nodejs
14   path: tinyjs
15

```

The default files for the new module contain basic Hello World code, which varies according to whether you enabled XSJS support.

## Importing a Module from an Archive

### Prerequisites

The `.zip` module archive that you want to import, which was exported from another MTA project, is available in the file system.

### Procedure

- From the root folder of the project, choose *File* *Import* *From File System*.
- Click *Browse* to locate and select your archive, and choose *Open*. The file name appears in the *File* field. The destination folder is displayed in the *Import to* field. To change this folder, choose *Select Folder*, and browse to the required folder, or create a new folder.
- The specified folder, containing the artifacts extracted from the archive, is created in the project.
- To make the imported folder a proper module in your project, you need to convert it into a module of the matching type. From the folder context menu, choose *Convert To*, and then the type of the target module.

#### Note

The conversion process does not check whether the imported folder structure matches the selected module type, and does not generate the module artifacts according to the selected type.

The imported module becomes a part of your MTA project, and the module entry is added to the MTA descriptor.

## 14.4.5.1.1 Implement OData Service Definitions

### Context

To expose a part of the data model, such as a table or a view, using OData, you should create a corresponding OData service definition in an `.xsodata` file. Since these files must be deployed to a JavaScript runtime, you should place them in a Node.js module.

### Procedure

1. From the context menu of a Node.js module's `lib` subfolder, choose  *New* > *File*.
2. In the *New File* box that opens, enter a file name with the extension `.xsodata`.
3. Double-click the file to open it in the dedicated editor. Use the Ctrl-Space key combination for hints while editing.  
For more information, see [Defining OData v2 Services for XS Advanced JavaScript Applications \[page 408\]](#).

## 14.4.5.2 Running and Testing Node.js Modules

You can run, test, and debug your Node.js modules using the tools provided by SAP Web IDE.

### Prerequisites

Before running a module, make sure that:

- All the relevant dependencies of the module are defined in the MTA descriptor (`mta.yaml`).
- All the modules and resources on which the module is dependent are implemented and available .

#### Note

You don't need to build your module explicitly before running, because it will be built automatically during the run process.

## Running a Module

To create a running application in XS Advanced and resolve all dependencies defined in `mta.yaml` run each Node.js module in your project. From the module context menu, choose [Run](#), then one of the following options:

Table 371:

Option	Description
<a href="#">Run Configurations</a>	Create a new or modify an existing run configuration. For more information, see <a href="#">Creating Run Configurations for Node.js Modules [page 1032]</a> .
 <a href="#">Run as Node.js Application</a>	Run the main .js file, which by default is <code>server.js</code> .

When the build finishes, its result is available in your workspace in the `node_modules` folder.

 Note

When you run a module for the first time, it is pushed to XS Advanced and bound to the required XS Advanced services that are defined in the MTA descriptor, which can take some time. After this, the run starts more quickly. If you modify only the source files, or `xs-app.json` in HTML5 modules, the module remains in XS Advanced, and the changes are injected directly into the running module.

## Show Dependency Updates in package.json

External dependencies are defined in the `package.json` file. To check on dependency updates, select your Node.js module and from the toolbar choose  [Build > Show Dependency Updates](#). Then go to [\(problems view\)](#) to see the outdated dependencies.

## Using the Run Console

Progress and status messages, generated during the run process, as well as the log records, are displayed in the Run console, which opens automatically in the main browser tab. You can show and hide the console by clicking the  [\(run console\)](#) icon in the bottom pane's toolbar.

The left pane of the console displays a list of all running modules in the current MTA project, with the corresponding run configurations, accompanied by a status icon with a tooltip. Click the run configuration name below a module name to view its most recent run log.

The controls displayed at the top of the console enable you to perform the following tasks:

- If the application URL is visible, click it to access the running application in the browser.
- To view the detailed server log of a run process, click [Logs](#).
- To clear the log in the console, click  [\(clear the log\)](#).
- To stop running the module, click  [\(stop\)](#).

This causes the module to be removed from XS Advanced, so that if you run the module again, the run process starts from the beginning.

- To run the module using a different configuration, select the configuration and choose the  (*run*) button in the task bar.

## Testing a Module

Build and test your JavaScript application with unit tests.

- Add unit tests in the `test` folder structure.
- Execute your test run configuration, which could be named, for example, *Run script test*. From the context menu of your module, select  *Run*  *Run script test*. In your run configuration, specify whether to run the test with code coverage.
- Review the test results.

Each JavaScript module must provide its own unit tests. The JavaScript module template comes with a sample unit test. Use this test to get familiar with the test functionality and the  (*test results*) pane. By default, the  (*test results*) pane shows *Test Results*, *Stack Trace*, and *Coverage*. If you have specified it in the run configuration, code coverage, which enables you to write targeted unit tests for uncovered code, is shown in the editor. To disable this function, edit your  (*settings*) .

You can also use the  (*test results*) pane to  (*export*) or  (*delete*) all test results for the selected module. Your workspace is connected to your  (*test results*) pane. The results you see are for the currently selected module, which is also referenced in the pane as the path to your current module (/<projectname>/<modulename>). If you miss the latest test result, select  (*refresh*). Also check the logs to see if your test execution failed; if it has, no test result is available.

## Debugging a Module

If you have enabled debugging in the run configuration, and set breakpoints in your `.js` files, the  (*debugger*) panel opens as soon as the module is up and running. When it reaches a breakpoint, you are notified that the execution is suspended. In this panel, you can perform the regular debugging tasks, such as viewing the call stack, examining the variables, stepping in and out of the functions, and so on. In the debugger console, you can also perform interactive evaluation of JavaScript statements in the context of the script used at a breakpoint.

If you open the context menu on a breakpoint, select *Edit Breakpoint Condition* to enhance that breakpoint with a condition.

### Note

Beginning with SAP HANA XS Advanced 2.0 SP00, you don't have to explicitly run Node.js modules in debug mode. Instead, you can  (*attach*) to any running Node.js module in the  (*debugger*) pane. Your Node.js module automatically switches to debug mode.

## Breakpoints

Table 372:

Breakpoint Status	Description
Enabled standard breakpoint	Shape outline: blue Shape fill: light blue
Enabled conditional breakpoint	Shape outline: orange Shape fill: light orange  An expression has been defined for a breakpoint. If the condition is met, the application suspends when it reaches the breakpoint.
Disabled standard and conditional breakpoint	Shape outline: blue or orange Shape fill: transparent  Breakpoints have been explicitly disabled. Disable breakpoints in the <i>Breakpoints</i> section by selecting the corresponding checkbox or deactivate globally by pressing the  ( <i>deactivate breakpoints</i> ) button.
Checkmark	Shape outline and fill are not affected.  You set the breakpoint successfully on a running debugging session. The checkmark indicates that the corresponding source file has been loaded. As a consequence, the application suspends when it reaches the breakpoint. For a conditional breakpoint, the user-defined expression has to be met.  A breakpoint without a checkmark means, there is no debug session running, the source file has not been loaded, or the breakpoint could not be successfully resolved. An unresolved breakpoint may occur when the workspace resource version and the currently executed resource version don't match.

## Related Information

[Running and Testing Java Modules \[page 1036\]](#)

[Running and Testing HTML5 Modules \[page 1078\]](#)

## 14.4.5.2.1 Creating Run Configurations for Node.js Modules

You can configure how to run, test, and debug Node.js modules in your project.

### Procedure

1. From the context menu of a Node.js module in your project, choose **Run** **Run Configurations**.
2. To create a new run configuration for a module, click **+Node.js Test**.  
A new configuration with a default name appears under the selected category in the left pane, and choose
3. Select a configuration to modify its name, and edit its properties as follows.

Table 373: General tab

Property	Description
<a href="#">Start with package.json script</a>	Run one of the scripts listed in the package.json file of the module, for example, a test script generated in an XSJS file.
<a href="#">Start with application file</a>	Run a .js file of your module.
<a href="#">Application File</a>	The name of .js file to run.
<a href="#">Debug Enabled</a>	If debugging is enabled, you can select or unselect the <a href="#">Break on first statement</a> .
<a href="#">Script Name</a>	Select a script name: <ul style="list-style-type: none"><li><input type="radio"/> <a href="#">start</a> Start the application.</li><li><input type="radio"/> <a href="#">test</a> Start the application with test execution.</li><li><input type="radio"/> <a href="#">test-coverage</a> Start the application with test execution and show code coverage. This script requires more resources than others.</li></ul>
<a href="#">Arguments</a>	(Optional) The arguments to be passed to the script.
<a href="#">Open a new browser window</a>	Select whether to open the script in a new browser tab. By default, this option is unselected for .js files.
<a href="#">Service Path</a>	(Optional) Enter a relative path to append to the server root. The browser is redirected to this path.

4. To save the configuration and run your module using this configuration, choose [Save and Run](#). To save all configurations without running, choose [OK](#). To discard all your changes, choose [Cancel](#).  
These settings are persistent and are used every time that you run your application with the selected run configuration, until you edit or delete the settings.

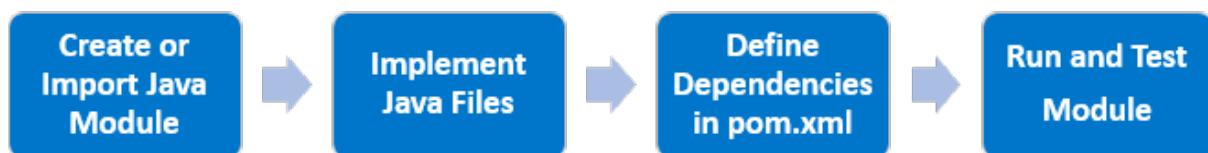
## Related Information

[Running and Testing Node.js Modules \[page 1028\]](#)

### 14.4.6 Developing Java Modules

A Java module is a collection of related Java files and service definitions. Java modules implement the business logic of your application, either instead of or in addition to Node.js modules. A Java module can be either a Java Web Archive (WAR) or Java Archive (JAR) built with Apache Maven.

To develop Java modules, perform the following steps:



- [Creating or Importing a Java Module \[page 1034\]](#)
- [Implementing Java Files \[page 1033\]](#)
- [Defining Dependencies \[page 1035\]](#)
- [Running and Testing Java Modules \[page 1036\]](#)

### Module Folder Structure

The default files of a new Java module (except Web application with OData support) already contain a basic, ready-to-run "Hello World" servlet. The module is structured following the Apache Maven standard directory layout.

### Using a settings.xml File

If you decide to use a `settings.xml` file to provide project-specific Apache Maven settings, be aware that this file overrides default settings. During an update, changes to defaults do not overrule your `settings.xml`. Make sure you read the release notes to identify possible gaps between your `settings.xml` and the recommended default settings.

If needed, create the `settings.xml` next to the `pom.xml` on the same level. For more information about this optional step, see [Defining Dependencies \[page 1035\]](#)

### Implementing Java Files

In the `src/main/java` folder, create and implement the required Java files.

## Related Information

[Apache Maven - Introduction to the Standard Directory Layout](#) ↗

[The SAP HANA XS Advanced Java Run Time \[page 589\]](#)

[Developing Multi-Target Applications \[page 983\]](#)

### 14.4.6.1 Creating or Importing a Java Module

You can create a new Java module or import an archived Java module.

## Creating a New Module

### Procedure

1. From the project context menu, choose **New > Java Module**, and enter a module name.
2. Define your basic module settings. Depending on your application needs, choose one of the Java module templates:

Option	Description
<a href="#">Simple web application</a>	Produces a *.war file and contains a basic, ready-to-run "Hello World" servlet.
<a href="#">Web application with OData support</a>	Produces a *.war file and provides an OData endpoint.
<a href="#">Multi-Module web application</a>	Produces *.war and *.jar files with dependencies.
<a href="#">SpringBoot application</a>	Produces a *.jar file, that brings its own runtime.

In addition, you can enable OData support to automatically expose annotated CDS models of your MTA project's HDB module as OData services.

3. Customize the template. If the default values don't fit your needs, change the Maven POM settings.

## Related Information

[Maintaining OData Services in XS Advanced \[page 407\]](#)

# Importing a Module from an Archive

## Prerequisites

The .zip module archive that you want to import, which was exported from another MTA project, is available in the file system.

## Procedure

1. From the root folder of the project, choose **File** **Import** **From File System**.
2. Click **Browse** to locate and select your archive, and choose **Open**. The file name appears in the **File** field. The destination folder is displayed in the **Import to** field. To change this folder, choose **Select Folder**, and browse to the required folder, or create a new folder.  
The specified folder, containing the artifacts extracted from the archive, is created in the project.
3. To make the imported folder a proper module in your project, you need to convert it into a module of the matching type. From the folder context menu, choose **Convert To**, and then the type of the target module.

### Note

The conversion process does not check whether the imported folder structure matches the selected module type, and does not generate the module artifacts according to the selected type.

The imported module becomes a part of your MTA project, and the module entry is added to the MTA descriptor.

## 14.4.6.2 Defining Dependencies

You can define optional dependencies of a Java module after creating it.

## Procedure

1. Modify `pom.xml` to add dependencies as found on the Maven website.
2. If your SAP Web IDE server cannot access the Internet, in particular `https://repo1.maven.org/maven2/`, you can set up your own Maven repository manager. Such custom repository setups can be configured by adding a `settings.xml` file to your Java module.
3. To identify outdated dependencies select your Java module and go to **File** **Show dependency updates**. Use the **Problems** view to see outdated dependencies.

## Related Information

[Creating or Importing a Java Module \[page 1034\]](#)

[Running and Testing Java Modules \[page 1036\]](#)

[Inside an MTA Descriptor \[page 985\]](#)

<http://search.maven.org/>

<https://maven.apache.org/settings.html>

<https://maven.apache.org/repository-management.html>

<https://repo1.maven.org/maven2/>

### 14.4.6.3 Running and Testing Java Modules

You can run and test your Java modules using the tools provided by SAP Web IDE.

#### Prerequisites

Before running a module, make sure that:

- All the relevant dependencies of the module are defined in the MTA descriptor.
- All the modules and resources on which the module is dependent are implemented and available (HDB modules are built).

**i Note**

You don't need to build your module explicitly before running, because it will be built automatically during the run process. The result of that build is then available in your workspace in the `target` folder. The result is either a `.war` or `.jar` file depending on the type of your Java module.

#### Running a Module

To create a running application in XS advanced and resolve all dependencies defined in `mta.yaml` run each Java module in your project. From the module context menu, choose [Run](#), and one of the following options:

Table 374:

Option	Description
<a href="#">Run Configurations</a>	Create a new or modify an existing run configuration. On the <i>Browser Window</i> tab, you can optionally choose a service path for your module to open in a new browser window, for instance, <code>/hello</code> for the Hello World sample servlet generated by the wizard.
 <a href="#">Run as Java Application</a>	Run the <code>.war</code> or <code>.jar</code> archive that was last built by Maven.

## Using the Run Console

Progress and status messages, generated during the run process, as well as the log records, are displayed in the Run console, which opens automatically in the main browser tab. You can show and hide the console by clicking the  (*run console*) icon in the bottom pane's toolbar.

The left pane of the console displays a list of all running modules in the current MTA project, with the corresponding run configurations, accompanied by a status icon with a tooltip. Click the run configuration name below a module name to view its most recent run log.

The controls displayed at the top of the console enable you to perform the following tasks:

- If the application URL is visible, click it to access the running application in the browser.
- To view the detailed server log of a run process, click [Logs](#).
- To clear the log in the console, click  (*clear the log*).
- To stop running the module, click  (*stop*).  
This causes the module to be removed from XS Advanced, so that if you run the module again, the run process starts from the beginning.
- To run the module using a different configuration, select the configuration and choose the  (*run*) button in the task bar.

## Testing a Module

Build and test your Java application with unit tests.

- Add unit tests in the `test` folder structure.
- Execute your tests. In the context menu from your module select [Build and Run Tests](#)
- Review the test results.

Each Java module must provide own its JUnit test. The Java module template comes with a sample JUnit test. Use this test to get familiar with the test functionality and the  (*test results*) pane. By default, the  (*test results*) pane shows *Test Results*, *Stack Trace* and *Coverage*. Also by default, the code coverage is shown in the editor, which enables you to write targeted unit tests for uncovered code. To disable this function edit your  (*settings*).

You can also use the [\(test results\)](#) pane to [\(export\)](#) or [\(delete\)](#) all test results for the selected module. Your workspace is connected to your [\(test results\)](#) pane. The results you see, are for the currently selected module, which is also referenced in the pane as the path to your current module (/<projectname>/<modulename>). If you miss the latest test result select [\(refresh\)](#). Also check the logs to see if your test execution failed. If it has, no test result is available.

## Debugging a Module

If you have enabled debugging in the run configuration, and set breakpoints in your .java files, the [\(debugger\)](#) panel will open as soon as the module is up and running. When it reaches a breakpoint you are notified. In this pane, you can perform the regular debugging tasks, such as viewing the call stack, examining the variables, stepping in and out of the functions, and so on.

### Note

Beginning with SAP HANA XS Advanced 2.0 SP02, you don't have to explicitly run Java modules in debug mode. Instead, you can [\(attach\)](#) to any running Java module in the [\(debugger\)](#) pane. Your Java module automatically switches to debug mode.

## Breakpoints

Table 375:

Breakpoint Status	Description
Enabled standard breakpoint	Shape outline: blue Shape fill: light blue
Disabled standard breakpoint	Shape outline: blue Shape fill: transparent  Breakpoints have been explicitly disabled. Disable breakpoints in the <i>Breakpoints</i> section by selecting the corresponding checkbox or deactivate globally by pressing the <a href="#">(deactivate breakpoints)</a> button.
Checkmark	Shape outline and fill are not affected.  You set the breakpoint successfully on a running debugging session. The checkmark indicates that the corresponding source file has been loaded. As a consequence the application will suspend when it reaches the breakpoint.  A breakpoint without a checkmark means, there is no debug session running, the source file has not been loaded, or the breakpoint could not be successfully resolved. An unresolved breakpoint may occur when the workspace resource version and the currently executed resource version don't match.

## Related Information

[Running and Testing Node.js Modules \[page 1028\]](#)

[Running and Testing HTML5 Modules \[page 1078\]](#)

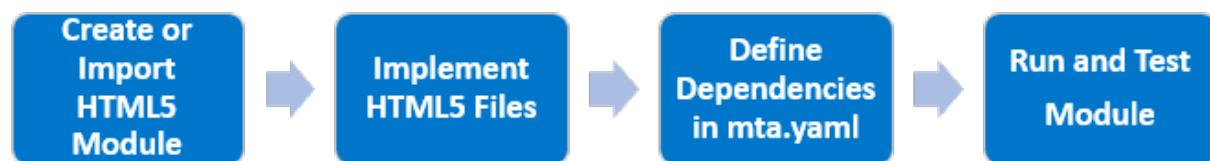
### 14.4.7 Developing HTML5 Modules

An HTML5 module is a collection of related HTML5 files that implement the user interface of your application.

SAP Web IDE supports the following HTML5 module types:

- Basic HTML5 module - a very basic UI app
- SAPUI5 HTML5 module - an SAPUI5-based UI app with an optional view but without a predefined layout
- SAP Fiori master-detail module - an SAPUI5-based UI app with the split-screen SAP Fiori layout

To develop HTML5 modules, perform the following steps:



- [Creating or Importing an HTML5 Module \[page 1040\]](#)
- [Implementing HTML5 Files \[page 1040\]](#)
- [Inside an MTA Descriptor \[page 985\]](#)
- [Running and Testing HTML5 Modules \[page 1078\]](#)

## Module Folder Structure

The following figure depicts a sample basic HTML5 module folder structure alongside the corresponding entry in the `mta.yaml`.

The diagram shows a sample basic HTML5 module folder structure on the left and its corresponding `mta.yaml` descriptor on the right. Both are highlighted with red boxes.

**Folder Structure:**

- `tinyui` (highlighted)
- `resources` (highlighted)
- `index.html`
- `package.json`
- `xs-app.json`
- `mta.yaml`

**mta.yaml Content:**

```
17   - name: hdi-container
18 #----- exposes SERVICE URL to consumers
19 provides:
20 - name: tinyjs_api
21 properties:
22   | service_url: ${default-url}
23 #
24   name: tinyui
25   type: html5
26   path: tinyui
27 # -- requires tinyjs service URL
28 requires:
29   - name: tinyjs_api
30   group: destinations
31 properties:
32   | name: tinyjs_url
33   | url: ~{service_url}
```

Table 376:

Folder	Description
<code>&lt;module name&gt;</code>	Contains the <code>xs-app.json</code> (application router configuration), and <code>package.json</code> (application router details and dependencies) files.
<code>resources</code>	Contains the HTML5 resource files, including the default <code>index.html</code> file.

 **Caution**

Do not change the version of `approuter` in the `package.json` of the module.

## Implementing HTML5 Files

In the `\resources` subfolder, create and implement the required HTML5 files. You can design the XML views using the SAP Web IDE layout editor.

For information about developing HTML5 apps with SAPUI5, see [SAPUI5: UI Development Toolkit for HTML5](#).

## Related Information

[Layout Editor \[page 1043\]](#)

[Developing Multi-Target Applications \[page 983\]](#)

### 14.4.7.1 Creating or Importing an HTML5 Module

You can create a new or import an archived HTML5 module.

## Creating a New Module

### Procedure

- From the project context menu, choose  [New > More](#).
- Depending on your application needs, choose one of the HTML5 module templates:

Option	Description
<a href="#">Basic HTML5 Module</a>	To create a very basic UI app.

Option	Description
<i>SAPUI5 HTML5 Module</i>	To create an SAPUI5-based UI app with an optional view but without a predefined layout.
<i>SAP Fiori Master-Detail Module</i>	To create an SAPUI5-based UI app with the split-screen SAP Fiori layout.  <b>i Note</b> As a prerequisite for creating this module, you need an OData service definition ( <code>metadata.xml</code> ) file that defines the data source for the app.

3. Enter the module name, and set the module properties as follows:

Table 377: SAPUI5 HTML5 Module

Property	Description
<i>Namespace</i>	Optional. Enter a value if you want to override the default, which is the module path in the project.
<i>View Type</i>	Select one of the available view types: None, XML, JSON, JavaScript, HTML.
<i>View Name</i>	Name of the view.

Table 378: SAP Fiori Master-Detail Module

Property	Description
<i>Sources</i>	Data source for the module. Select an existing OData service from one of the following: <ul style="list-style-type: none"> <li>o <i>Current project</i>: Select an OData service from the list of available services exposed by the Java and Node.js modules in the current project. If the module is not running, run it. Note that the exposed service must allow anonymous access.</li> <li>o <i>Workspace</i>: select a <code>metadata.xml</code> file from one of the projects in your workspace.</li> <li>o <i>File system</i>: select a <code>metadata.xml</code> file from the file system.</li> </ul>
Other properties:  <i>Application Settings</i>  <i>SAP Fiori Settings</i>	Click the image on the right to enlarge it, and see the detailed explanations of the properties.

A new HTML5 module with the specified name is created in your project, and a corresponding section is added to the MTA descriptor (`mta.yaml`). For example:

```

3  version: 0.0.1
4
5  modules:
6    - name: tinydb
7      type: hdb
8      path: tinydb
9      requires:
10        - name: hdi-container
11
12    - name: tinyjs
13      type: nodejs
14      path: tinyjs
15
16
17    - name: tinyui5
18      type: html5
19      path: tinyui5

```

#### Note

The default `index.html` file of the new module implements a basic Hello World application.

## Importing a Module from an Archive

### Prerequisites

The `.zip` module archive that you want to import, which was exported from another MTA project, is available in the file system.

### Procedure

- From the root folder of the project, choose **File** **Import** **From File System**.
- Click **Browse** to locate and select your archive, and choose **Open**. The file name appears in the **File** field. The destination folder is displayed in the **Import to** field. To change this folder, choose **Select Folder**, and browse to the required folder, or create a new folder.
- The specified folder, containing the artifacts extracted from the archive, is created in the project.
- To make the imported folder a proper module in your project, you need to convert it into a module of the matching type. From the folder context menu, choose **Convert To**, and then the type of the target module.

#### Note

The conversion process does not check whether the imported folder structure matches the selected module type, and does not generate the module artifacts according to the selected type.

The imported module becomes a part of your MTA project, and the module entry is added to the MTA descriptor.

## 14.4.7.2 Layout Editor

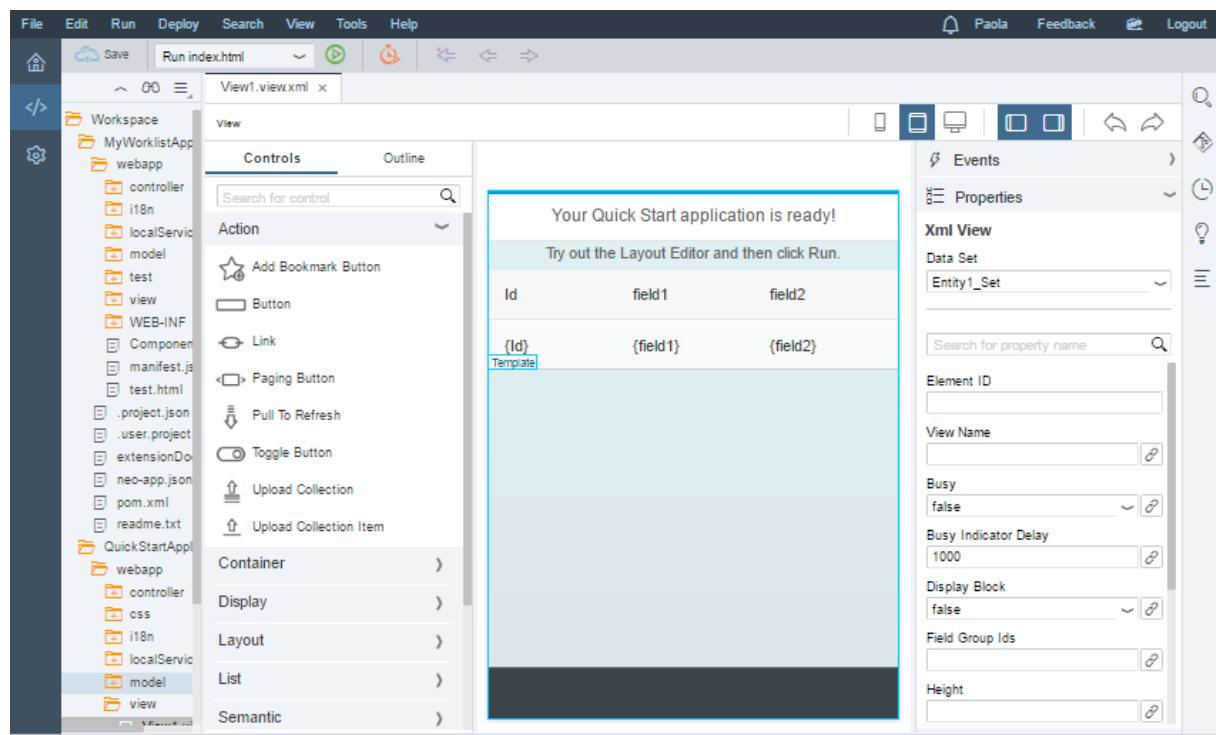
Display the content of an XML view in the SAP Web IDE layout editor to see it in a way that closely corresponds to how it will appear in your finished application.

### i Note

The layout editor is not supported in the Safari browser.

## Layout Editor Landscape

The layout editor is composed of a canvas, a pane on the left that includes the *Controls* and *Outline* tabs, and a pane on the right that includes the *Events* and *Properties* panes.

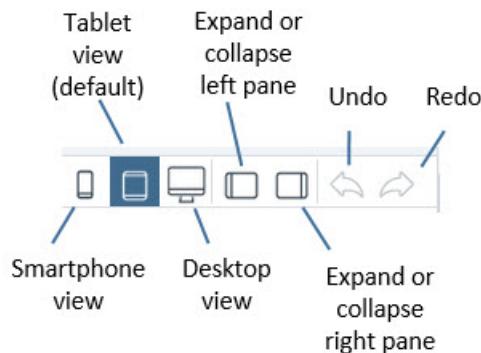


## Toolbar

The buttons on the layout editor toolbar allow you to:

- Change the device format of the canvas to smartphone, tablet, or desktop view.
- Change the application's SAPUI5 version.
- Expand and collapse the panes to the right and left of the canvas.
  - The pane on the left side includes the *Controls* and *Outline* tabs.

- The pane on the right side includes the *Properties* and *Events* panes.
- Undo and redo actions.



## Controls Tab

You can expand or collapse each section by clicking the arrow on each section header. You can also search for controls by entering the control name in the search field at the top of the *Controls* tab. The relevant sections expand to display the controls that match the search criteria.

**i Note**

Make sure to delete the search criteria if you want to expand other sections.

You can drag and drop controls from the *Controls* tab onto the canvas. For more information, see [Drag and Drop \[page 1051\]](#).

Controls	Outline
Search for control <input type="text"/>	
Action	›
Container	›
Display	›
Layout	›
List	›
Tile	›
User Input	›

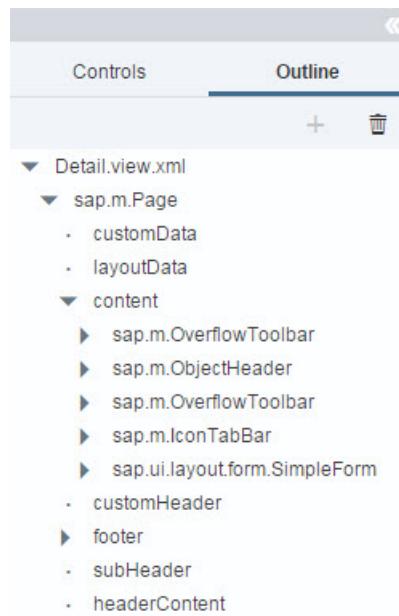
You can find the list of available controls in [SAPUI5 Controls Supported in the Layout Editor \[page 1058\]](#).

## Outline Tab

Controls that are selected on the *Outline* tab are automatically selected on the canvas and vice versa.

You can use the *Outline* tab to see the hierarchy of controls on the canvas. In addition, you can add and remove controls from the canvas using the *Outline* tab.

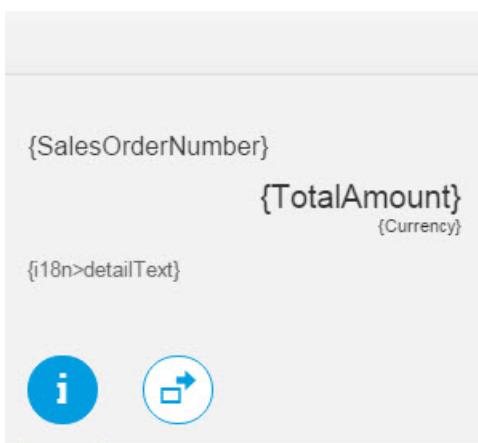
For more information, see [Adding Controls from the Outline Tab \[page 1049\]](#).



## Canvas

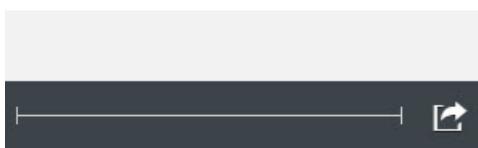
The canvas in the middle of the layout editor area provides a graphical display of the selected XML view.

Click a control on the canvas to select it. Click again to select its parent control. You can keep clicking until you reach the highest control in the hierarchy and then the focus will return to the original control. Click outside the canvas to undo the selection.



NetPriceAmount:  
{NetPriceAmount}

TaxAmount:  
{TaxAmount}  
{OrderDate}



### Events and Properties Pane

On the right side of the canvas is a pane that displays the following panes:

## Events Pane

The *Events* pane allows you to select an existing event handler from the controller for an event of the selected control. The icon next to each event opens the code editor to display the relevant controller in the XML code.

Events

Object Header

Search for event name

Title Selector Press

Icon Press

Intro Press

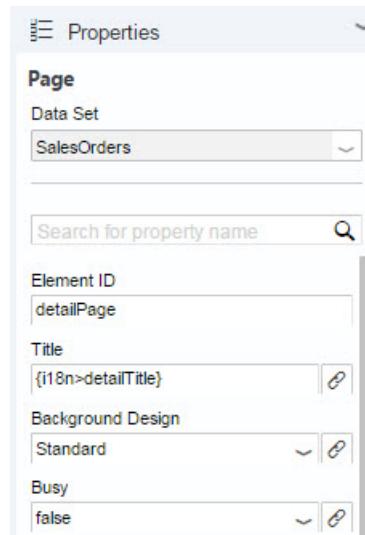
Title Press

Validate Field Group

## Properties Pane

The *Properties* pane shows the properties of the control that is currently selected in the canvas and allows you to modify its property values. The most commonly used properties for each control are displayed at the top of the list. The  icon next to each property opens the *Data Binding* dialog box.

For more information, see [Binding Data \[page 1053\]](#), [Binding Data to a Simple Control \[page 1054\]](#), and [Binding Data to an Aggregate-Type Control \[page 1055\]](#).



### Note

Deprecated properties or aggregations are marked with the label *deprecated* (also in the *Outline* tab). For more information, see *SAP Library for User Interface Add-On 1.0 for SAP NetWeaver* on SAP Help Portal at <http://help.sap.com/nw-uiaddon>. Under *Application Help*, open *SAP Library*, and search for *deprecation*.

### [Working with the Layout Editor \[page 1048\]](#)

An overview of the steps required to edit a project using the layout editor.

### [SAPUI5 Controls Supported in the Layout Editor \[page 1058\]](#)

Provides a list of SAPUI5 controls that are supported in the layout editor.

### [Try It: Build an Application with the Layout Editor \[page 1071\]](#)

Get an overview of the features that are available with the layout editor by following this tutorial for building an application.

## Related Information

### [Working with the Layout Editor \[page 1048\]](#)

### [SAPUI5 Controls Supported in the Layout Editor \[page 1058\]](#)

## 14.4.7.2.1 Working with the Layout Editor

An overview of the steps required to edit a project using the layout editor.

### Prerequisites

- In your MTA project, you have an HTML5 module with an XML view.
- In the `view` subfolder of the module folder, you have chosen the XML file.

### Procedure

1. From the context menu of the XML view, choose  [Open with Layout Editor](#).
2. Edit views of your application as follows:
  - Add controls to your view using drag and drop functionality.
  - Delete controls from your view.
  - Rearrange controls in your view using drag and drop functionality.
  - Use the keyboard to navigate within the canvas. Click the same control twice to move the selection to its parent.
  - Bind controls in the layout editor to elements from the OData service.
  - Extract a control to a fragment using the context menu.

#### [Adding Controls from the Outline Tab \[page 1049\]](#)

You can add controls to the canvas from the *Outline* tab.

#### [Deleting Controls from the Outline Tab \[page 1049\]](#)

You can remove controls from the *Outline* tab.

#### [Creating a New Function \[page 1050\]](#)

You can create a new function for a controller from the *Events* pane.

#### [Drag and Drop \[page 1051\]](#)

Add controls to the canvas by using the drag and drop functionality.

#### [Keyboard Support \[page 1052\]](#)

Use the keyboard to move selected controls or navigate within the view that you opened with the layout editor.

#### [Layout Editor Binding Capabilities \[page 1053\]](#)

In the layout editor, you can bind properties of controls or control aggregations to data fields, to i18n models, and to label annotations.

## 14.4.7.2.1.1 Adding Controls from the Outline Tab

You can add controls to the canvas from the *Outline* tab.

### Procedure

1. On the *Outline* tab, select a control to which you want to add another control.
2. At the top of the *Outline* tab, click the  **Add** button, then in the popup menu, select the control you want to add.

The control is added on the *Outline* tab and appears on the canvas.

 **Note**

The information bar at the top of the canvas shows you where you are about to drop the control.

**Task overview:** [Working with the Layout Editor \[page 1048\]](#)

### Related Information

[Deleting Controls from the Outline Tab \[page 1049\]](#)

[Creating a New Function \[page 1050\]](#)

[Drag and Drop \[page 1051\]](#)

[Keyboard Support \[page 1052\]](#)

[Layout Editor Binding Capabilities \[page 1053\]](#)

[Deleting Controls from the Outline Tab \[page 1049\]](#)

[SAPUI5 Controls Supported in the Layout Editor \[page 1058\]](#)

## 14.4.7.2.1.2 Deleting Controls from the Outline Tab

You can remove controls from the *Outline* tab.

### Procedure

1. On the *Outline* tab, select a control that you want to delete.
2. At the top of the *Outline* tab, click the  **Delete** button.

The control is removed from the view.

**Task overview:** [Working with the Layout Editor \[page 1048\]](#)

## Related Information

[Adding Controls from the Outline Tab \[page 1049\]](#)

[Creating a New Function \[page 1050\]](#)

[Drag and Drop \[page 1051\]](#)

[Keyboard Support \[page 1052\]](#)

[Layout Editor Binding Capabilities \[page 1053\]](#)

[Adding Controls from the Outline Tab \[page 1049\]](#)

[SAPUI5 Controls Supported in the Layout Editor \[page 1058\]](#)

### 14.4.7.2.1.3 Creating a New Function

You can create a new function for a controller from the *Events* pane.

#### Procedure

1. On the canvas, select a control.
2. In the *Events* pane, open the dropdown list under the event for whose controller you want to create a function.
3. In the dropdown list, select *New Function*.
4. In the *New Function* dialog box, enter a function name and click *OK*.

#### i Note

The function name you enter must be a valid JavaScript function name.

After entering the name for your new function, the code is created by the layout editor in the code editor.

To go directly to that code, click the *Go to code*  icon next to the new function name in the *Events* pane.

**Task overview:** [Working with the Layout Editor \[page 1048\]](#)

## Related Information

[Adding Controls from the Outline Tab \[page 1049\]](#)

[Deleting Controls from the Outline Tab \[page 1049\]](#)

[Drag and Drop \[page 1051\]](#)

[Keyboard Support \[page 1052\]](#)

[Layout Editor Binding Capabilities \[page 1053\]](#)

[Layout Editor \[page 1043\]](#)

## 14.4.7.2.1.4 Drag and Drop

Add controls to the canvas by using the drag and drop functionality.

Select the control that you want to add to the canvas from the *Controls* tab on the left side of the layout editor area and drag it to the drop target on the canvas. A tooltip displays the drop targets as you drag the control around the canvas.

### Example

If you want to add an *HBox* container with a *Button* control to your view, do the following:

1. Go to the *Controls* tab.
2. Open the *Container* section by clicking the arrow to the right of the section title.
3. Select the *HBox* control and drag it to the canvas. Drop it at the position where you want it to appear.
4. Open the *Action* section by clicking the arrow to the right of the section title.
5. Select the *Button* control and drag it to the canvas.
6. Drop the *Button* control onto the *HBox* container.

**Parent topic:** [Working with the Layout Editor \[page 1048\]](#)

## Related Information

[Adding Controls from the Outline Tab \[page 1049\]](#)

[Deleting Controls from the Outline Tab \[page 1049\]](#)

[Creating a New Function \[page 1050\]](#)

[Keyboard Support \[page 1052\]](#)

[Layout Editor Binding Capabilities \[page 1053\]](#)

## 14.4.7.2.1.5 Keyboard Support

Use the keyboard to move selected controls or navigate within the view that you opened with the layout editor.

### Selecting Controls

If you selected a control in the canvas of the layout editor, you can move the selection using the arrow keys:

- **[UP ARROW]**: moves the selection to the parent of the selected control
- **[DOWN ARROW]**: moves the selection to the child of the selected control
- **[LEFT ARROW]**: moves the selection to the control that is up/to the right of the selected control (within the same aggregation)
- **[RIGHT ARROW]**: moves the selection to the control that is down/to the left of the selected control (within the same aggregation)
- **[CTRL]+ control**: moves the selection to the parent of the selected control

### Moving Selected Controls

If you have selected a control in the canvas of the layout editor, you can change its position within the aggregation:

- **[SHIFT]+[LEFT ARROW]**: moves the control up/to the right
- **[SHIFT]+[RIGHT ARROW]**: moves the control down/to the left

### Changing the Drop Target

When you drag and drop a control from the palette to the canvas or from one position within the canvas to another, you can use the keyboard to define the drop position of the dragged control:

- Use **[SHIFT]** or **[ALT]** to define the drop position of the dragged control within an aggregation:
  - **[SHIFT]**: moves the drop position up/to the right
  - **[ALT]**: moves the drop position down/to the left

**Parent topic:** [Working with the Layout Editor \[page 1048\]](#)

### Related Information

[Adding Controls from the Outline Tab \[page 1049\]](#)

[Deleting Controls from the Outline Tab \[page 1049\]](#)

---

[Creating a New Function \[page 1050\]](#)  
[Drag and Drop \[page 1051\]](#)  
[Layout Editor Binding Capabilities \[page 1053\]](#)

## 14.4.7.2.1.6 Layout Editor Binding Capabilities

In the layout editor, you can bind properties of controls or control aggregations to data fields, to i18n models, and to label annotations.

**Parent topic:** [Working with the Layout Editor \[page 1048\]](#)

### Related Information

[Adding Controls from the Outline Tab \[page 1049\]](#)  
[Deleting Controls from the Outline Tab \[page 1049\]](#)  
[Creating a New Function \[page 1050\]](#)  
[Drag and Drop \[page 1051\]](#)  
[Keyboard Support \[page 1052\]](#)

## 14.4.7.2.1.6.1 Binding Data

In the layout editor, you can bind properties of controls or control aggregations to an artifact in the OData service.

### Prerequisites

You have defined a data set for the view that you are working on, by doing one of the following:

- If you are opening a view that has no data set defined for it, the *Data Binding* dialog box opens, where you can define a data set.
- Select the view, and define the data set from the dropdown list.

#### Caution

If you change the data set that is defined for the current view, the existing data bindings might become invalid.

## Overview

The following types of bindings are possible:

- Properties of controls
- Aggregations of controls

### Note

To bind properties of models that are not OData models, you must work from the source code files and not from the layout editor. Alternatively, if you do not want to work with the source files in the XML editor, you can enter free text for properties in the *Properties* pane.

### Note

If your application does not consume an OData service, you can add the *OData Service* component to it.

#### [Binding Data to a Simple Control \[page 1054\]](#)

You can bind data to a simple control.

#### [Binding Data to an Aggregate-Type Control \[page 1055\]](#)

You can bind data to an aggregate-type control, which creates a template.

#### [Unbinding Data from a Simple Control Property \[page 1056\]](#)

You can unbind data from a simple control property.

#### [Unbinding Data from a Template \[page 1057\]](#)

You can unbind data from a template (aggregate-type control).

## Related Information

[SAPUI5 API Reference](#)

## 14.4.7.2.1.6.1.1 Binding Data to a Simple Control

You can bind data to a simple control.

## Procedure

1. On the canvas, select the desired control for which you want to define data binding.
2. In the *Properties* pane to the right of the canvas, do one of the following:

- To the right of the property to which you want to bind data, click the *Binding*  button, and then in the *Data Binding* dialog box:

1. In the *Data Fields* list, double-click one or more data fields that you want to add to the expression. The data fields are automatically concatenated to the string in the *Expression* box.
  2. Click *OK* or first manually edit the expression string and then click *OK*.
- In the *Properties* pane, for a field or dropdown list, manually enter the required expression within curly brackets {...} according to the data set that you selected.

**Task overview:** [Binding Data \[page 1053\]](#)

## Related Information

[Binding Data to an Aggregate-Type Control \[page 1055\]](#)

[Unbinding Data from a Simple Control Property \[page 1056\]](#)

[Unbinding Data from a Template \[page 1057\]](#)

## 14.4.7.2.1.6.1.2 Binding Data to an Aggregate-Type Control

You can bind data to an aggregate-type control, which creates a template.

### Procedure

1. In the canvas or on the *Outline* tab to the left of the canvas, choose an aggregate-type control that you want to turn into a template, such as a *List Item* control.
2. In the properties pane to the right of the canvas, under the *Data Set* dropdown list, check the *Set as template* checkbox.  
This control becomes a template.
3. In the *Confirmation Needed* dialog box, confirm the removal of any existing controls on the same level by clicking *OK*.

### Results

The template item is now marked *Template* in the *Outline* tab.

**Task overview:** [Binding Data \[page 1053\]](#)

---

## Related Information

[Binding Data to a Simple Control \[page 1054\]](#)

[Unbinding Data from a Simple Control Property \[page 1056\]](#)

[Unbinding Data from a Template \[page 1057\]](#)

[Aggregation Binding](#)

### 14.4.7.2.1.6.1.3 Unbinding Data from a Simple Control Property

You can unbind data from a simple control property.

#### Procedure

1. In the canvas, select the control whose data you want to unbind.
2. In the *Properties* pane, do one of the following:
  - In the property field or dropdown list, delete the string.
  - Click the *Data Binding*  button to open the *Data Binding* dialog box. Click *Clear* and then click *OK*.

Task overview: [Binding Data \[page 1053\]](#)

## Related Information

[Binding Data to a Simple Control \[page 1054\]](#)

[Binding Data to an Aggregate-Type Control \[page 1055\]](#)

[Unbinding Data from a Template \[page 1057\]](#)

## 14.4.7.2.1.6.1.4 Unbinding Data from a Template

You can unbind data from a template (aggregate-type control).

### Procedure

1. In the canvas, select the aggregate-type control whose data you want to unbind.
2. In the properties pane to the right of the canvas, do one of the following:
  - Clear the *Set as template* checkbox.
  - In the dropdown list of the data set, select *Unbind*.

**Task overview:** [Binding Data \[page 1053\]](#)

### Related Information

[Binding Data to a Simple Control \[page 1054\]](#)

[Binding Data to an Aggregate-Type Control \[page 1055\]](#)

[Unbinding Data from a Simple Control Property \[page 1056\]](#)

## 14.4.7.2.1.6.2 Binding to the i18n Model

You can bind a control property to the i18n model or create a new i18n entry.

### Procedure

1. On the canvas, select the desired control for which you want to define i18n model binding.
2. In the *Properties* pane to the right of the canvas, do one of the following:
  - To the right of the property to which you want to bind data, click the *Binding*  button, and then in the dialog box that appears:
    1. From the drop down list, select *i18n*, double-click one or more entries that you want to add to the expression.  
The entries are automatically concatenated to the string in the *Expression* box.
    2. Click *OK* or first manually edit the expression string and then click *OK*.
  - In the *Properties* pane, for a field or dropdown list, manually enter the required expression within curly brackets {...}.

- Click + to add a new i18n entry.

## 14.4.7.2.1.6.3 Binding to a Label Annotation

You can bind a control property to a label annotation.

### Context

You can bind a control property to a label annotation that resides in the Odata metadata file. Other annotation files are not supported.

### Procedure

1. On the canvas, select the desired control for which you want to define a label annotation binding.
2. In the *Properties* pane to the right of the canvas, do one of the following:
  - To the right of the property to which you want to bind data, click the *Binding*  button, and then in the dialog box that appears:
    1. From the drop down list, select *Labels*, double-click one or more annotation that you want to add to the expression.  
The annotations are automatically concatenated to the string in the *Expression* box.
    2. Click *OK* or first manually edit the expression string and then click *OK*.
  - In the *Properties* pane, for a field or dropdown list, manually enter the required expression within curly brackets {...}.

## 14.4.7.2.2 SAPUI5 Controls Supported in the Layout Editor

Provides a list of SAPUI5 controls that are supported in the layout editor.

### Controls Tab

The SAPUI5 controls listed below can be dragged and dropped from the *Controls* tab onto the canvas.

#### Note

The controls on the *Controls* tab are also available from the *Outline* tab. For more information, see [Adding Controls from the Outline Tab \[page 1049\]](#).

**i** Note

For more information about SAPUI5 controls, see [UI development toolkit for HTML5 - Demo Kit](#).

Table 379: SAPUI5 Controls Available on the Controls Tab

SAPUI5 Control Name	Description
Action List Items sap.m.ActionListItem	Button that is used to fire actions when pressed.
Action Select sap.m.ActionSelect	Provides a list of predefined items that allows end users to choose options and additionally trigger some actions.
Add Bookmark Button sap.ushell.ui.footerbar.AddBookmarkButton	Button that is displayed in the application footer. Clicking the button opens a dialog box that allows the user to save the app state, so that the app can be launched in this state directly from the launchpad.
Analytic Map sap.ui.vbm.AnalyticMap	Renders a map based on a GeoJSON source.
App sap.m.App	The root element of an SAPUI5 mobile application. It inherits from the <code>NavContainer</code> control and thus provides its navigation capabilities. App provides certain header tags to the HTML page that are relevant for mobile apps.
Bar sap.m.Bar	Centers a control like a title while having other controls on its left and right.
Busy Indicator sap.ui.core.BusyIndicator	Provides methods to show or hide a waiting animation that covers the whole page and blocks user interaction.
Button sap.m.Button	Allows users to trigger actions.
Calendar sap.ui.unified.Calendar	Basic calendar that is used for DatePickers.
Calendar Legend sap.ui.unified.CalendarLegend	Legend for the <code>Calendar</code> control. Displays special date colors with their corresponding description.
Carousel sap.m.Carousel	Navigates through a list of controls by swiping right or left.
Check Box sap.m.CheckBox	Allows the user to select one or multiple items from a list.

SAPUI5 Control Name	Description
Column sap.m.Column	Allows definition of column-specific properties that are applied when rendering a List control.
Column List Item sap.m.ColumnListItem	Used with cell aggregation to create rows for the sap.m.Table control.
Combo Box sap.m.ComboBox	Combines a dropdown list with items and a text field with a button allowing the user to either type a value directly or choose from a list of predefined items.
Custom List Item sap.m.CustomListItem	With content aggregation, can be used to customize standard list items that are not provided by SAPUI5. ListItem type is applied to CustomListItem as well.  <b>i Note</b> Content aggregation allows any control. Complex responsive layout controls (such as Table and Form) should not be aggregated as content.
Custom Tile sap.m.CustomTile	Displays application-specific content in the Tile control.
Date Picker sap.m.DatePicker	Date input control with a calendar used as a date picker.
Detail Page sap.m.semantic.DetailPage	An sap.m.semantic.ShareMenuPage control that supports certain semantic buttons that have default semantic-specific properties and are eligible for content aggregation.
Display List Item sap.m.DisplayListItem	Used to represent a label and a value.
Feed Input sap.m.FeedInput	Allows the user to enter text for a new feed entry and then post it.
Feed List Item sap.m.FeedListItem	Provides a set of properties for text, sender information, and time stamp.
Flex Box sap.m.FlexBox	Builds the container for a flexible box layout.
Fullscreen Page sap.m.semantic.FullscreenPage	An sap.m.semantic.ShareMenuPage control that supports certain semantic buttons that have default semantic-specific properties and are eligible for content aggregation.

SAPUI5 Control Name	Description
Geo Map sap.ui.vbm.GeoMap	A map control that allows the user to position multiple visual objects on top of a map.
Grid sap.ui.layout.Grid	Layout that positions its child controls in a 12-column flow layout.
Group Header List Item sap.m.GroupHeaderListItem	Used to display the title of a group and act as a separator between groups in sap.m.List and sap.m.Table.
HBox sap.m.HBox	Builds the container for a horizontal flexible box layout.
Horizontal Layout sap.ui.layout.HorizontalLayout	Provides support for horizontal alignment of controls.
Icon sap.ui.core.Icon	Uses an embedded font instead of a pixel image.
Icon Tab Bar sap.m.IconTabBar	Represents a collection of tabs with associated content.
Icon Tab Filter sap.m.IconTabFilter	Represents a selectable item inside an Icon Tab Bar control.
Icon Tab Header sap.m.IconTabHeader	Displays a number of Icon Tab Filter and Icon Tab Separator controls.
Icon Tab Separator sap.m.IconTabSeparator	Icon used to separate two Icon Tab Filter controls.
Image sap.m.Image	Wrapper around the IMG tag.
Input sap.m.Input	Allows users to input data.
Input List Item sap.m.InputListIItem	List item used for a label and an input field.
Invisible Text sap.ui.core.InvisibleText	Used to bring hidden texts to the UI for screen reader support.

SAPUI5 Control Name	Description
Item sap.ui.core.Item	Control base type.
Label sap.m.Label	Used in SAPUI5 mobile applications to provide label text for other controls.
Link sap.m.Link	Used to trigger actions or to navigate to other applications or web pages.
List sap.m.List	Provides a container for all types of list items.
List Item sap.ui.core.ListItem	Used in lists or list-like controls, such as <code>DropdownBox</code> .
Master Page sap.m.semantic.MasterPage	An <code>sap.m.semantic.SemanticPage</code> control that supports certain semantic buttons that have default semantic-specific properties and are eligible for content aggregation.
Message Strip sap.m.MessageStrip	Allows the embedding of application-related messages in the application.
Nav Container sap.m.NavContainer	Handles hierarchical navigation between Page controls or other fullscreen controls.
Object Attribute sap.m.ObjectAttribute	Displays a text field that can be normal or active.
Object Header sap.m.ObjectHeader	Allows the user to easily identify a special object.
Object Identifier sap.m.ObjectIdentifier	Display control that allows the user to easily identify a specific object.
Object List Item sap.m.ObjectListItem	Display control that provides summary information about an object as an item in a list.
Object Number sap.m.ObjectNumber	Displays number and number unit properties for an object.
Object Status sap.m.ObjectStatus	Status information that can be either text with a value state, or an icon.

SAPUI5 Control Name	Description
Overflow Toolbar sap.m.OverflowToolbar	Container based on sap.m.Toolbar that provides overflow when its content does not fit in the visible area.
Page sap.m.Page	Basic container for a mobile application screen.
Paging Button sap.m.PagingButton	Allows users to navigate between items and entities.
Panel sap.m.Panel	Container for controls that has a header and content.
Progress Indicator sap.m.ProgressIndicator	Shows the progress of a process in a graphical way.
Pull To Refresh sap.m.PullToRefresh	Triggers the refresh event.
Radio Button sap.m.RadioButton	Control similar to CheckBox, but it allows the user to choose only one of a predefined set of options.
Radio Button Group sap.m.RadioButtonGroup	Used as a wrapper for a group of sap.m.RadioButton controls, which then can be used as a single UI element.
Rating Indicator sap.m.RatingIndicator	Used to rate content.
Search Field sap.m.SearchField	Allows users to input a search string.
Segmented Button sap.m.SegmentedButton	Horizontal control made of multiple buttons, which can display a title or an image.
Select sap.m.Select	Provides a list of items that allows the user to select an item.
Select List sap.m.SelectList	Displays a list of items that allows the user to select an item.
Semantic Page sap.m.semantic.SemanticPage	An enhanced sap.m.Page control that can contain controls with semantic meaning. Content specified in sap.m.semantic.SemanticPage semantic control aggregations are automatically positioned in dedicated sections of the footer or the header of the page, depending on the control's semantics.

SAPUI5 Control Name	Description
Share Menu Page sap.m.semantic.ShareMenuPage	An sap.m.semantic.SemanticPage control that supports a Share menu in the footer.
Simple Form sap.ui.layout.form.SimpleForm	Provides an API for creating simple forms. Inside a SimpleForm control, a Form control is created along with its FormContainers control and FormElements control, but the complexity in the API is removed.
Slider sap.m.Slider	User interface control that allows the user to adjust values within a specified numerical range.
Standard List Item sap.m.StandardListItem	List item that provides the most common use cases, such as image, title, and description.
Standard Tile sap.m.StandardTile	Displayed in the Tile container.
Switch sap.m.Switch	User interface control on mobile devices that is used for switching between binary states.
Table sap.m.Table	Provides a set of sophisticated and convenient functions for responsive table design.
Text sap.m.Text	Used for embedding longer text paragraphs that need text wrapping into your application.
Text Area sap.m.TextArea	Allows multiline text input.
Tile Container sap.m.TileContainer	Container that arranges same-size tiles on carousel pages.
Title sap.ui.core.Title	Used to aggregate other controls.
Toggle Button sap.m.ToggleButton	Control that toggles between pressed and normal state.
Toolbar sap.m.Toolbar	Horizontal container that is usually used to display buttons, labels, selects, and other input controls.
Toolbar Separator sap.m.ToolbarSeparator	Creates a visual separator between toolbar items.

SAPUI5 Control Name	Description
Toolbar Spacer sap.m.ToolbarSpacer	Adds horizontal space between toolbar items.
Upload Collection sap.m.UploadCollection	Allows users to upload single or multiple files.
Upload Collection Item sap.m.UploadCollectionItem	Provides information about uploaded files.
VBox sap.m.VBox	Builds the container for a vertical flexible box layout.
Vertical Layout sap.ui.layout.VerticalLayout	Layout in which the content controls are rendered one below the other.

## Outline Tab

The SAPUI5 controls listed below are available only from the *Outline* tab in the layout editor.

### i Note

For more information about SAPUI5 controls, see [UI development toolkit for HTML5 - Demo Kit](#).

Table 380: SAPUI5 Controls Available on the Outline Tab

SAPUI5 Control Name	Description
sap.m.semantic.AddAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an sap.m.semantic.SemanticPage control.
sap.m.semantic.CancelAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an sap.m.semantic.SemanticPage control.
sap.m.semantic.DiscussInJamAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an sap.m.semantic.SemanticPage.
sap.m.semantic.EditAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an sap.m.semantic.SemanticPage control.

SAPUI5 Control Name	Description
sap.m.semantic.FavoriteAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an sap.m.semantic.SemanticPage control.
sap.m.semantic.FilterAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an sap.m.semantic.SemanticPage control.
sap.m.semantic.FlagAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an sap.m.semantic.SemanticPage control.
sap.m.semantic.ForwardAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an sap.m.semantic.SemanticPage control.
sap.m.semantic.GroupAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an sap.m.semantic.SemanticPage control.
sap.m.semantic.MainAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an sap.m.semantic.SemanticPage control.
sap.m.semantic.MessagesIndicator	Has default semantic-specific properties and is eligible to be included in the aggregation content of an sap.m.semantic.SemanticPage control.
sap.m.semantic.MultiSelectAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an sap.m.semantic.SemanticPage control.
sap.m.semantic.NegativeAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an sap.m.semantic.SemanticPage control.
sap.m.semantic.OpenInAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an sap.m.semantic.SemanticPage control.
sap.m.semantic.PositiveAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an sap.m.semantic.SemanticPage control.
sap.m.semantic.PrintAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an sap.m.semantic.SemanticPage control.

SAPUI5 Control Name	Description
sap.m.semantic.SaveAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an sap.m.semantic.SemanticPage control.
sap.m.semantic.SendEmailAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an sap.m.semantic.SemanticPage control.
sap.m.semantic.SendMessageAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an sap.m.semantic.SemanticPage.
sap.m.semantic.ShareInJamAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an sap.m.semantic.SemanticPage control.
sap.m.semantic.SortAction	Has default semantic-specific properties and is eligible to be included in the aggregation content of an sap.m.semantic.SemanticPage control.
sap.m.Title	Used for header texts and title.
sap.m.Token	Renders a token containing text and an optional <i>Delete</i> icon.
sap.m.Tokenizer	Displays multiple tokens.
sap.suite.ui.commons.BusinessCard	Allows displaying of business card information, including an image, first title (either URL link or text), second title, and multiple text lines.
sap.suite.ui.commons.ComparisonChart	Displays a comparison chart.
sap.suite.ui.commons.ComparisonData	Comparison tile value holder.
sap.suite.ui.commons.CountingNavigationItem	Extends the sap.ui.ux3.NavigationItem control. This control displays the number of items in a corresponding content area. It also provides a rich tooltip that appears and disappears after a certain delay.
sap.suite.ui.commons.DateRangeScroller	Provides a method to scroll through a series of time periods, each of which is represented by a start date and an end date, known as the date range.
sap.suite.ui.commons.DateRangeSliderInternal	Provides the user with a RangeSlider control that is optimized for use with dates.
sap.suite.ui.commons.DeltaMicroChart	Displays a delta of two values as a chart.

SAPUI5 Control Name	Description
sap.suite.ui.commons.DynamicContainer	Displays multiple GenericTile controls as changing slides.
sap.suite.ui.commons.FacetOverview	Used in UnifiedThingInspector to display a preview of facet content.
sap.suite.ui.commons.GenericTile	Tile control that displays a title, description, and customizable main area.
sap.suite.ui.commons.HarveyBallMicroChart	Chart that shows a comparative part to a total.
sap.suite.ui.commons.HarveyBallMicroChartItem	Configuration of a slice on a pie chart.
sap.suite.ui.commons.HeaderCell	Contains four cells (West, North, East, South). It can display one or more controls in different layouts. Each aggregation must contain only one instance of HeaderCellItem.
sap.suite.ui.commons.HeaderCellItem	Object that contains an instance of a control and information about its height. It should be used inside sap.suite.ui.commons.HeaderCell.
sap.suite.ui.commons.HeaderContainer	Container that provides a horizontal layout. On mobile devices, it provides a horizontal scroll. On desktops, it provides scroll left and scroll right buttons. This control supports keyboard navigation. You can use left and right arrow keys to navigate through the inner content. The [Home] key puts focus on the first control and the [End] key puts focus on the last control. Use the [Enter] key or [Spacebar] key to choose the control.
sap.suite.ui.commons.JamContent	Displays SAP Jam content text, subheader, and numeric value in a tile.
sap.suite.ui.commons.KpiTile	Used in UnifiedThingInspector to display object-related KPIs in a factsheet.
sap.suite.ui.commons.NewsContent	Displays news content text and subheader in a tile.
sap.suite.ui.commons.NoteTaker	Allows creation and storage of notes for further reference.
sap.suite.ui.commons.NoteTakerCard	Allows storage of NoteTaker card header and body text.
sap.suite.ui.commons.NoteTakerFeeder	Allows entering quick notes and note cards.
sap.suite.ui.commons.NumericContent	Numeric content to be used in a tile or other place where it is needed to show numeric values with semantic colors and deviations.

SAPUI5 Control Name	Description
sap.suite.ui.commons.TileContent	Serves as a universal container for different types of content and footer.
sap.ui.commons.ApplicationHeader	Located at the top of an application page and consists of four areas.
sap.ui.commons.Button	Allows users to trigger actions such as save or print. For the button UI, you can define text or an icon, or both.
sap.ui.commons.CheckBox	Provides a box that can be flagged and has a label. A checkbox can either stand alone, or be in a group with other checkboxes.
sap.ui.commons.ColorPicker	Allows the user to choose a color. The color can be defined using HEX, RGB, or HSV values, or a CSS colorname.
sap.ui.commons.FileUploader	Framework that generates an input field and a button with the text <i>Browse....</i>
sap.ui.commons.MenuBar	Represents a user interface area that is the entry point for menus with their menu items.
sap.ui.commons.MenuButton	Common button control that opens a menu when chosen by the user. The control provides an API for configuring the docking position of the menu.
sap.ui.commons.Paginator	Provides navigation between pages within a list of numbered pages.
sap.ui.commons.Panel	Represents a container with scroll functionality that can be used for text and controls.
sap.ui.commons.PasswordField	Text field with masked characters that borrows its properties and methods from the <code>TextField</code> control.
sap.ui.commons.ProgressIndicator	Shows the progress of a process in a graphical way.
sap.ui.commons.RadioButton	Consists of a round element and descriptive text.
sap.ui.commons.RangeSlider	Interactive control that is displayed either as a horizontal or vertical line with two pointers and units of measurement.
sap.ui.commons.RatingIndicator	Allows the user to rate a certain topic.
sap.ui.commons.SegmentedButton	Provides a group of buttons.
sap.ui.commons.Slider	Interactive control that is displayed either as a horizontal or vertical line with a pointer and units of measurement.
sap.ui.commons.Splitter	Allows splitting the screen into two areas.

SAPUI5 Control Name	Description
sap.ui.commons.TextArea	Control for entering or displaying multiple rows of text.
sap.ui.commons.TextField	Renders an input field for text input.
sap.ui.commons.Toolbar	Horizontal row of items where in many cases the single toolbar items are buttons that contain icons.
sap.ui.commons.Tree	Simple tree for displaying an item in a hierarchical way.
sap.ui.commonsTreeNode	Tree node element.
sap.ui.layout.FixFlex	Builds the container for a layout with a fixed and a flexible part.
sap.ui.layout.form.Form	Structured into <code>FormContainer</code> controls, each of which consists of <code>FormElement</code> controls.
sap.ui.layout.form.FormContainer	Group inside a <code>Form</code> .
sap.ui.layout.form.FormElement	Row in a <code>FormContainer</code> control.
sap.ui.layout.form.FormLayout	Base layout used to render a <code>Form</code> control.
sap.ui.layout.form.GridLayout	Renders a <code>Form</code> control using an HTML table-based grid.
sap.ui.layout.form.ResponsiveGridLayout	Renders a <code>Form</code> control using a responsive grid.
sap.ui.layout.form.ResponsiveLayout	Renders a <code>Form</code> control with a responsive layout.
sap.ui.unified.Currency	Text view that displays currency values and aligns them at the separator.
sap.ui.unified.FileUploader	Framework that generates an input field and a button with the text <code>Browse ....</code>
sap.ui.unified.FileUploaderParameter	Represents a parameter for the <code>FileUploader</code> , which is rendered as a hidden input field.
sap.ui.unified.Menu	Interactive element that provides a choice of different actions to the user.
sap.ui.unified.MenuItem	Standard item used inside a menu. Represents an action that can be selected by a user in the menu or that can be used as a submenu that organizes the actions hierarchically.
sap.ui.unified.ShellOverlay	Opened in front of an <code>sap.ui.unified.Shell</code> control.
sap.ui.unified.SplitContainer	Provides a main content and a secondary content area.

SAPUI5 Control Name	Description
sap.ui.ux3.ExactArea	Consists of two sections: a toolbar and a content area where arbitrary controls can be added.
sap.ui.ux3.FeedChunk	Unit that is embedded, standalone or multiple, into a Feed control.
sap.ui.ux3.Feeder	Lean common feed, or a comment feed, with a text commit function.
sap.uxap.ObjectPageHeader	Static part of an Object page header.
sap.uxap.ObjectPageHeaderContent	Dynamic part of an Object page header.

### 14.4.7.2.3 Try It: Build an Application with the Layout Editor

Get an overview of the features that are available with the layout editor by following this tutorial for building an application.

#### Related Information

[Prerequisites \[page 1071\]](#)

[Create an OData Model File \[page 1072\]](#)

[Create a Project for Your New Application \[page 1073\]](#)

[Add Controls to Your New Application \[page 1074\]](#)

### 14.4.7.2.3.1 Prerequisites

Prerequisite steps that you must complete before following the steps in the tutorial topic "Add Controls to Your New Application."

1. You have created an OData model file according to the instructions in the topic [Create an OData Model File \[page 1072\]](#).
2. You have created a new project from a template according to the instructions in the topic [Create a Project for Your New Application \[page 1073\]](#).

#### Related Information

[Create an OData Model File \[page 1072\]](#)

[Create a Project for Your New Application \[page 1073\]](#)

[Add Controls to Your New Application \[page 1074\]](#)

## 14.4.7.2.3.1.1 Create an OData Model File

This task is a prerequisite for the tutorial on building an application with the layout editor.

### Procedure

1. Open a new text file.
2. Copy and paste the XML code provided below into the text file.

```
<edmx:Edmx Version="1.0" xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx" xmlns:sap="http://www.sap.com/Protocols/SAPData">
<edmx:DataServices xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata" m:DataServiceVersion="2.0">
<Schema xmlns="http://schemas.microsoft.com/ado/2008/09/edm" Namespace="MySalesOrders">
<EntityType Name="SalesOrder">
  <Key>
    <PropertyRef Name="SalesOrderNumber" />
  </Key>
  <Property Name="SalesOrderNumber" Type="Edm.String" sap:label="Sales Order Number" Nullable="false" MaxLength="10" />
  <Property Name="TotalAmount" Type="Edm.Decimal" sap:label="Total Amount" Precision="16" Scale="3" sap:unit="Currency" MaxLength="10" />
  <Property Name="Currency" Type="Edm.String" sap:label="Currency" MaxLength="5" sap:semantics="currency-code" />
  <Property Name="CustomerID" Type="Edm.String" sap:label="Customer ID" MaxLength="10" />
  <Property Name="CustomerName" Type="Edm.String" sap:label="Customer Name" MaxLength="35" />
  <Property Name="NetPriceAmount" Type="Edm.Decimal" sap:label="Net Price Amount" Precision="16" Scale="3" sap:unit="Currency" />
  <Property Name="TaxAmount" Type="Edm.Decimal" sap:label="Tax Amount" Precision="16" Scale="3" sap:unit="Currency" />
  <Property Name="OrderDate" Type="Edm.String" sap:label="Order Date" />
  <Property Name="RequestedDate" Type="Edm.String" sap:label="Requested Date" />
  <Property Name="Status" Type="Edm.String" MaxLength="1" sap:label="Status" />
  <Property Name="SalesOrganization" Type="Edm.String" MaxLength="4" sap:label="Sales Organization" />
  <Property Name="SalesOrganizationName" Type="Edm.String" MaxLength="20" sap:label="SalesOrganizationName" sap:creatable="false" sap:updatable="false" sap:sortable="false" sap:filterable="false" />
  <Property Name="DistributionChannel" Type="Edm.String" MaxLength="4" sap:label="Distribution Channel" />
  <Property Name="Division" Type="Edm.String" MaxLength="2" sap:label="Division" />
  <Property Name="DistributionChannelName" Type="Edm.String" MaxLength="20" sap:label="DistributionChannelName" sap:creatable="false" sap:updatable="false" sap:sortable="false" sap:filterable="false" />
  <Property Name="DivisionName" Type="Edm.String" MaxLength="20" sap:label="DivisionName" sap:creatable="false" sap:updatable="false" sap:sortable="false" sap:filterable="false" />
</EntityType>
```

```
<EntityContainer Name="MySalesOrders_Entities"
m:IsDefaultEntityContainer="false">
<EntitySet Name="SalesOrders" EntityType="MySalesOrders.SalesOrder"
sap:searchable="true" sap:requires-filter="true" />
</EntityContainer>
</Schema>
</edmx:DataServices>
</edmx:Edmx>
```

3. Save the file to your computer with the file name `SalesOrderService_metadata.xml`.

## 14.4.7.2.3.1.2 Create a Project for Your New Application

This task is a prerequisite for the tutorial on building an application with the layout editor.

### Context

Create a new project for a sales order tracking application using a template.

### Procedure

1. Open SAP Web IDE in the Google Chrome browser.
2. Open a new project from a template by using one of the following options:
  - On the SAP Web IDE Welcome page, choose *New Project from Template*.
  - In the *File* menu, choose *New > Project from Template*.

The *New Project* wizard opens.

3. In the *Template Selection* wizard step, select the *SAP Fiori Master Detail Application* tile and then choose the *Next* button.
4. In the *Basic Information* wizard step, enter a project name. Choose the *Next* button.

#### Note

The project name must start with a letter or an underscore and may contain alphanumeric characters, periods, and underscores. It may not end with a period.

5. In the *Data Connection* wizard step, select the *File System* source. Then choose the *Browse* button and navigate to the file `SalesOrderService_metadata.xml` that you created in the topic [Create an OData Model File \[page 1072\]](#). Choose the *Next* button.
6. In the *Template Customization* wizard step, enter or select the mapping data in the fields and dropdown lists according to the tables below.

#### *Application Settings Section*

Field or Dropdown List	Value
Title	Sales Orders
Project Namespace	<project_namespace>
	<p><b>i Note</b></p> <p>The namespace must start with a letter or an underscore and may contain alphanumeric characters, periods, and underscores. It may not end with a period.</p>
OData Collection	SalesOrders
Item Title	SalesOrderNumber
Numeric Attribute	TotalAmount
Unit of Measure	Currency

#### Detail Section

Field or Dropdown List	Value
Title	Sales Order
Detail Text	Sales Order Details
Status Attribute	Leave it blank
Attribute 1	NetPriceAmount
Attribute 2	TaxAmount
Attribute 3	OrderDate

7. Choose the *Next* button.
8. In the *Confirmation* wizard step, choose the *Finish* button.

Your new project is now created in your workspace.

### 14.4.7.2.3.2 Add Controls to Your New Application

Steps for adding controls to your new sales order tracking application.

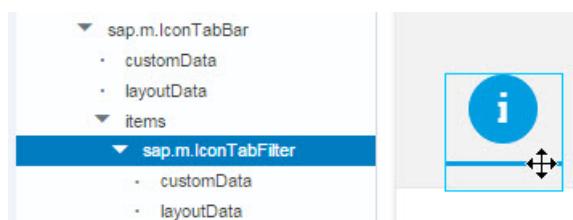
#### Procedure

1. In the workspace, expand the folder with name of the project that you created in the topic [Create a Project for Your New Application \[page 1073\]](#), then expand the view folder and right-click the Detail.view.xml view.
2. From the context menu, choose  The content of the XML view is displayed on the canvas in a way that corresponds to how it will appear in your finished application.

3. Change the icons of the *Icon Tab Filter* controls:

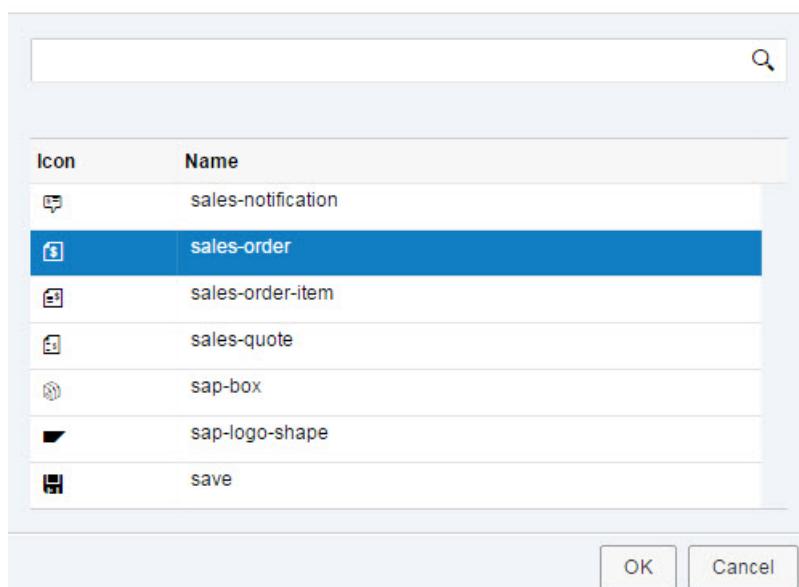
- a. On the canvas, select the first *Icon Tab Filter* control on the left side of the *Icon Tab Bar* control.

In the *Outline* tab on the left side of the canvas, `sap.m.IconTabFilter` is selected.



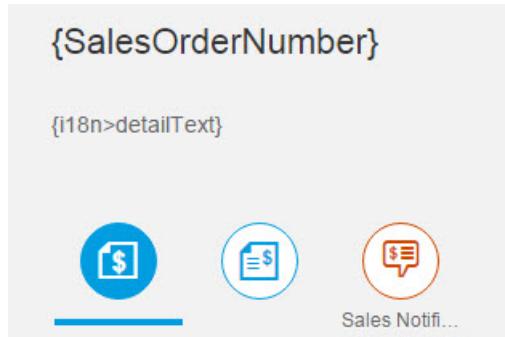
- b. In the *Properties* pane to the right of the canvas, in the *Icon* field, choose the *Select Icon* button to open the *Select Icon* dialog box. Scroll down and select the `sales-order` icon and then choose *OK*.

Select Icon



- c. On the canvas, select the second *Icon Tab Filter* control.
- d. In the *Properties* pane, in the *Icon* field, choose the *Select Icon* button to open the *Select Icon* dialog box. Scroll down and select the `sales-order-item` icon and then choose *OK*.
4. Add a new *Icon Tab Filter* control to the view:
- On the *Controls* tab to the left of the canvas, expand the *Container* section or use the search field to search for the *Icon Tab Filter* control.
  - From the *Controls* tab, drag the *Icon Tab Filter* control to the canvas and drop it on the *Icon Tab Bar* control.

- c. Change its  icon to the *sales-notification*  icon in the same way as you changed the icons in step 3.
- d. In the *Properties* pane, in the *Count* field, clear the value by deleting it.
- e. In the *Properties* pane, change the value in the *Text* field to *Sales Notifications*.



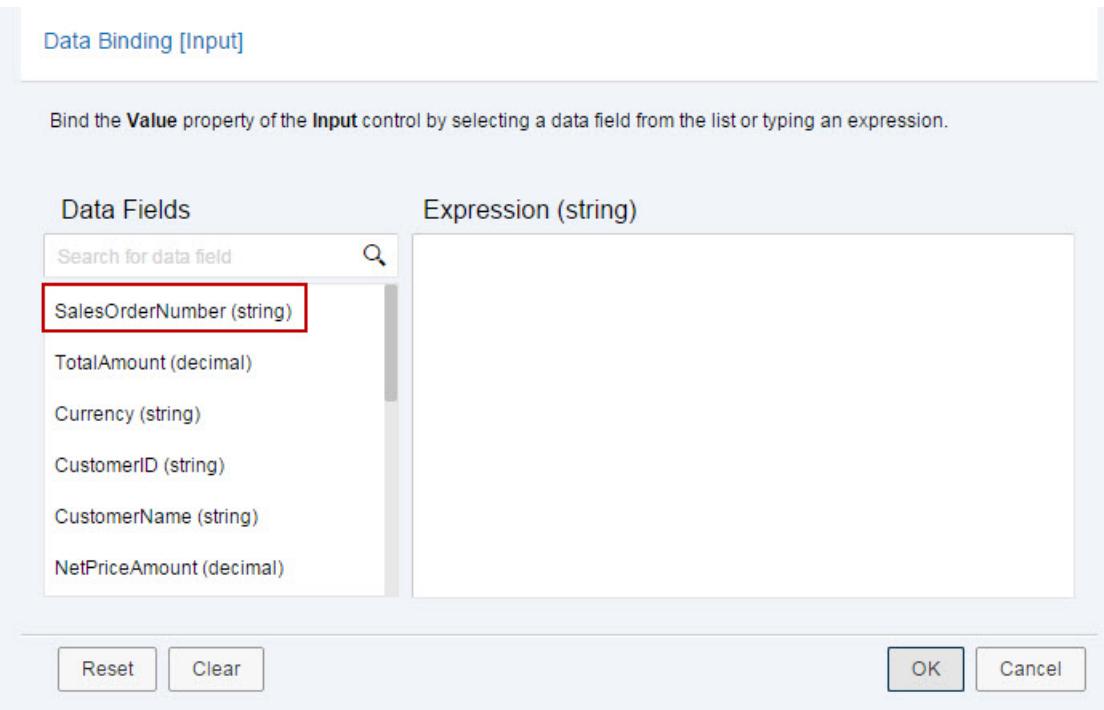
5. Add a *Simple Form* control to the new sales notification *Icon Tab Filter* control:
- On the *Controls* tab to the left of the canvas, expand the *Layout* section.
  - In the canvas, select the *Sales Notification Icon Tab Filter*.
  - From the *Controls* tab on the left, drag the *Simple Form* control to the canvas and drop it on the space below the *Icon Tab Filter* control.
6. Change the properties of the new *Simple Form* control:
- In the canvas, select the title in the *Simple Form* control.
  - In the *Properties* pane, change the value of the *Text* property to **My Sales Notifications**.
  - In the canvas, in the *Simple Form* control, select *Label 1*.
  - In the *Properties* pane, change the value of the *Text* property to **Sales Order Number**.
  - On the *Controls* tab to the left of the canvas, search for the *Label* control.
  - Drag a new label and drop it above the second input field.

- g. In the canvas, in the *Simple Form* control, select the label that you just created and then in the properties pane to the right of the canvas, change its text to **Order Date**.
- h. Change the text of the *Label 2* control of the third input field in the same way to **Status**.

## My Sales Notifications

Sales Order Number:	<input type="text" value="{\$SalesOrderNumber}"/>
Order Date:	<input type="text" value="{\$OrderDate}"/>
Status:	<input type="text" value="{\$Status}"/>

7. Bind the *Value* property of the input fields to elements from the OData service that you created:
  - a. In the canvas, select the *Sales Order Number* field in the *SimpleForm* control.
  - b. In the *Properties* pane, click the *Data Binding*  button next to the *Value* property.
  - c. In the *Data Binding [Input]* dialog box, select *SalesOrderNumber* and choose *OK*.



- d. In the canvas, select the *Order Date* field in the *Simple Form* control.
- e. In the *Properties* pane, click the *Data Binding*  button next to the *Value* property.
- f. In the *Data Binding* dialog box, select *OrderDate* and choose *OK*.
- g. Select the *Status* field and bind it to **Status** in the same way as the previous two input fields.
8. Save your changes.  
Your changes are saved for both the layout editor and the code editor.
9. Run your application to test the result.

## Related Information

[Running and Testing HTML5 Modules \[page 1078\]](#)

### 14.4.7.3 Running and Testing HTML5 Modules

You can run and test your HTML5 modules using the tools provided by SAP Web IDE and the browser.

#### Prerequisites

Before running a module, make sure that:

- All the relevant dependencies of the module are defined in the MTA descriptor (mta.yaml).
- All the modules and resources on which the module is dependent are implemented and available .

#### i Note

You don't need to build your module explicitly before running, because it will be built automatically during the run process.

#### Running a Module

To run an HTML5 module, from the module context menu, choose *Run*, and one of the following options:

Table 381:

Option	Description
<a href="#">Run Configurations...</a>	Create a new or modify an existing run configuration. For more information, see <a href="#">Configuring How to Run HTML5 Modules [page 1079]</a> .
 <a href="#">Run as &gt; Web Application</a>	Run the module as a Web application starting with its HTML file.
 <a href="#">Run as &gt; SAP Fiori Component on Sandbox</a>	Run the module in an SAP Fiori launchpad sandbox environment.

#### i Note

When you run a module for the first time, it is pushed to XS Advanced and bound to the required XS Advanced services that are defined in the MTA descriptor, which can take some time. After this, the run starts more quickly. If you modify only the source files, or `xs-app.json` in HTML5 modules, the module remains in XS Advanced, and the changes are injected directly into the running module.

## Using the Run Console

Progress and status messages, generated during the run process, as well as the log records, are displayed in the Run console, which opens automatically in the main browser tab. You can show and hide the console by clicking the  (*run console*) icon in the bottom pane's toolbar.

The left pane of the console displays a list of all running modules in the current MTA project, with the corresponding run configurations, accompanied by a status icon with a tooltip. Click the run configuration name below a module name to view its most recent run log.

The controls displayed at the top of the console enable you to perform the following tasks:

- If the application URL is visible, click it to access the running application in the browser.
- To view the detailed server log of a run process, click [Logs](#).
- To clear the log in the console, click  (*clear the log*) .  
This causes the module to be removed from XS Advanced, so that if you run the module again, the run process starts from the beginning.
- To stop running the module, click  (*stop*) .

## Debugging a Module

Debug the module using the browser developer tools.

### 14.4.7.3.1 Configuring How to Run HTML5 Modules

You can configure how to run HTML5 modules in your project.

#### Procedure

1. From the context menu of an HTML5 module in your project, choose  [Run > Run Configurations](#).
2. To create a new run configuration for a module, click  and the type of run configuration, *Web Application* or *SAP Fiori Component in Sandbox*.  
A new configuration with a default name appears under the selected category in the left-side panel.
3. To edit a newly created or existing run configuration, click its name in the panel, and edit its properties as required.

Field	Description
General tab	

Field	Description
<a href="#">Name</a>	Modify the default name.
<a href="#">Run Application File</a>	If the type is <i>Web Application</i> , then enter the name of the main HTML file of your module.  If the type is <i>SAP Fiori Component in Sandbox</i> , then enter the SAP Fiori component to start the application, such as the <i>Component.js</i> file.
<a href="#">Preview Mode</a>	Select how to open the application: <ul style="list-style-type: none"> <li>○ <i>Without Frame</i>: in a full window (default)</li> <li>○ <i>With Frame</i>: in a frame with configurable viewing options</li> </ul>
<b>Mock Data</b>	Select <i>Run with mock data</i> to use mock data in your application.  If you use mock data for your application, make sure that: <ul style="list-style-type: none"> <li>○ You have configured settings for the mock server. For more information, see <a href="#">Configuring Mock Data Usage [page 1095]</a>.</li> <li>○ The application that you want to run uses the SAPUI5 OData model with JSON format.</li> </ul>
<b>URL Components tab</b>	
<a href="#">URL Parameters</a>	You can define additional URL parameters as name-value pairs to pass to the application. <ol style="list-style-type: none"> <li>1. To add a new row for an additional parameter, choose <a href="#">Add Parameter</a>.</li> <li>2. In the new row, enter a parameter name and its value.</li> </ol>
<a href="#">URL Hash Fragment</a>	Enter the URL hash fragment (also known as a fragment identifier) without the hash delimiter (#).

4. To save the configuration and run your module using this configuration, choose [Save and Run](#). To save all configurations without running, choose [OK](#). To discard all your changes, choose [Cancel](#).

These settings are persistent and are used every time that you run your application with the selected run configuration, until you edit or delete the settings.

## 14.4.8 Developing SAP Fiori Launchpad Modules

SAP Fiori launchpad modules are used to create a SAP Fiori launchpad site that runs on the XS Advanced Model.

For information about the runtime capabilities of SAP Fiori launchpad, see the *User Guide* under [SAP Fiori Launchpad](#).

## Related Information

[Creating a New Launchpad Site \[page 1081\]](#)

[Adding an App to an Existing Launchpad Site \[page 1088\]](#)

[Roles and Authorizations \[page 1091\]](#)

### 14.4.8.1 Creating a New Launchpad Site

Create a Multi-Target Application (MTA) project to contain your launchpad site.

#### Context

The following modules are created:

- MySite – the Web entry point to the SAP Fiori launchpad. This module contains an `xs-app.json` file, which is configured like an approuter component. For more information, see [Configure the XS Advanced Application Router](#).
- MySite-Content – the configuration files of the site, which contain a sample group and sample tiles/apps.

#### Procedure

1. In SAP Web IDE, create a new Multi-Target Application project.
2. Select a space for it. For more information, see [Setting Up Application Projects](#).
3. From the context menu of the project root folder, select [New](#) [SAP Fiori Launchpad Site Modules](#).
4. In the wizard, enter the site's module name, for example, **MySite** and click **Next**.
5. Specify a name for the site's content module (optional). The default name in this example is **MySite-Content**.
6. Click **Finish**, and the following modules are created: MySite and MySite-Content.
7. From the context menu of the project root folder, select **Build**, to build an `.mtar` file for your project.
8. Download the `.mtar` file from `mta_archives`, by selecting **Export** from the file's context menu.
9. Use the SAP HANA XS Advanced Model command line tool to deploy this file to SAP HANA XS Advanced Model server. At the end of the deployment process, you receive the URL of the launchpad site in the command line tool.
10. Run the URL in your browser.

You have now created a launchpad site that runs on SAP HANA XS Advanced Model with sample out-of-the-box content.

## Related Information

- [Adding a SAPUI5 App to the Launchpad Site \[page 1082\]](#)
- [Adding Content to the Launchpad Site \[page 1083\]](#)
- [Assigning Apps to Scopes \[page 1084\]](#)
- [Applying Translations to Your Launchpad Site \[page 1085\]](#)
- [Applying a Custom Theme to Your Launchpad Site \[page 1086\]](#)

### 14.4.8.1.1 Adding a SAPUI5 App to the Launchpad Site

Add a custom SAPUI5 app to the launchpad.

#### Prerequisites

You already have a SAPUI5 application that you want to add to the launchpad site.

#### Procedure

1. Copy your SAPUI5 application resources under the site's module ('MySite').
2. To the `xs-app.json` file, add the route to the application's static resources. For example:

#### Sample Code

```
{  
  "source": "^/appl/sap/demo/(.*)",  
  "target": "$1",  
  "localDir": "appl/sap/demo/",  
  "cacheControl": "public, max-age=31536000,must-revalidate"  
}
```

3. Under the `applications` folder, add an application configuration to your site as follows:
  - a. Create a folder under the `applications` folder. For example, `MySAPUI5App`.
  - b. Copy the `manifest.json` file of the application to the new folder. For example:

#### Sample Code

```
<site-content-deployer-folder-name>/  
| - package.json  
| - applications/  
|   \ - MySAPUI5App/  
|     | - manifest.json  
|     \ - i18n/
```

```
\ - src/
  | - site-content.json
  \ - i18n/
<MySite>/
```

- c. Add to the `manifest.json` the location of your SAPUI5 component. For example:

#### Sample Code

```
...
"sap.platform.cf": {
  "uri": "<Location of your SAPUI5 component under your MySite module>"
}
...
```

## Related Information

[Adding Content to the Launchpad Site \[page 1083\]](#)

### 14.4.8.1.2 Adding Content to the Launchpad Site

You can add a group to the launchpad, assign it to the Everyone role, and add a tile to the group.

## Procedure

1. In the `site-content.json`, add to the `groups` property, a group with the same structure as the `sample` group. Under the `identification.id` property, provide the new group with an id that is different from the id of the sample group.
2. To enable all users to see this group, assign it to the `Everyone` role, by adding the group id to the `id` property. For example:

#### Sample Code

```
"roles": {
  "Everyone": {
    "catalogs": [],
    "groups": [
      {
        "id": "<id>"
      }
    ]
  }
},
```

3. To add a tile to the group, add a new tile to the array of tiles under the group's `tiles` property. The link between the tile in the group and the app to which it navigates, is done using the `semanticObject` and `action` properties.

### Note

The values of the `semanticObject` and `action` properties must match the corresponding properties that are defined in the `manifest.json` of your SAPUI5 application.

4. To see the results in runtime, build, deploy, and run your launchpad site again.

## Related Information

[Assigning Apps to Scopes \[page 1084\]](#)

### 14.4.8.1.3 Assigning Apps to Scopes

You need to assign an app to scopes in the `site-content.json`, to determine which users see this app tile. Otherwise, all users will be able to see this tile/app.

To assign an app to a scope, add the `oAuthScopes` property to the `manifest.json` of your app. For example:

#### Sample Code

```
... "sap.platform.cf": {  
    "oAuthScopes": ["$XSAPPNAME.Display", "$XSAPPNAME.Create"]  
} ...
```

Only users assigned to these scopes will see the tile/app.

### Note

The scopes you assign must be the scopes in the `xs-security.json` file, which contains the configuration for the User Authentication and Authorization system (UAA). For more information, see [The Application Security Descriptor \[page 764\]](#).

## 14.4.8.1.4 Applying Translations to Your Launchpad Site

You can apply translations to your launchpad site. Translations are handled according to SAPUI5 conventions.

### Context

To apply translations, you need to perform the following tasks, for both the `manifest.json` file (in the `applications` folder), and also for the group, catalog, and application entities in the `site-content.json` file.

For more information about the `manifest.json` file, see the SAPUI5 Toolkit: [Descriptor for Applications, Components, and Libraries](#).

### Procedure

1. Modify the properties of the entity to be translated, so that the value of the property is in the SAPUI5 format. For example:

#### Sample Code

```
{  
    "identification" :  
        "title" :  "{{<key_for_title_in_property_file>}}"  
}
```

#### Note

Only properties defined by the spec as translatable are translated.

2. Modify the `i18n` property of the entity, in the `manifest.json` file, and in the `identification` sections of the group, catalog and application entities in the `site-content.json` file, to point to the master language properties file.

#### Note

The master language properties file must be located in the same folder as the `manifest.json` or `site-content.json`, or in one of their subfolders.

Backward relativity is not supported. For example, this is not supported:

```
"i18n" : "../../otherFolder/bundle.properties"
```

The following is an example of a group:

Sample Code

```
{  
    "version": "1.0",  
    "identification": {  
        "id": "123",  
        "namespace": "",  
        "title": "{{group_title}}",  
        "entityType": "group",  
        "i18n": "i18n/MyBundle.properties"  
    },  
    "payload": {...}
```

3. Create property files for translation:

- For the master language, provide any file name with the .properties extension. For example, MyBundle.properties:

Sample Code

```
<key_for_title_in_property_file> = My Group Title
```

- For other languages, provide a file name that ends with the \_<locale> string. For example, MyBundle\_fr.properties:

Sample Code

```
<key_for_title_in_property_file> = My Group Title In French
```

### 14.4.8.1.5 Applying a Custom Theme to Your Launchpad Site

You can apply a custom theme to your launchpad site.

#### Prerequisites

- You have created a custom theme in the SAP UI theme designer tool.
- You have downloaded the custom theme zip file to your computer, and unzipped it.

#### Procedure

- In your SAP Web IDE project, under the site module, *MySite* for example, under the themes folder, place the folder that contains the unzipped custom theme files.

2. In the `site-content.json` file, under the site's payload section, "sap.cloud.portal", add the name of the custom theme that is active, and the list of themes from which the user can select a theme. For example:

#### Sample Code

```
"sap.cloud.portal": {  
    "config": {  
        "theme.id": "my_company_theme",  
        "theme.active": "[\"sap_belize\", \"sap_hcb\", \"my_company_theme\"]"  
    }  
}
```

In addition, under "siteThemes", below the default themes, add an entry for each custom theme. For example:

#### Sample Code

```
"siteThemes": {  
    "sap_hcb": {  
        "description": "SAPHighContrastBlack",  
        "name": "sap_hcb",  
        "path": "sap_hcb"  
    },  
    "sap_belize": {  
        "description": "SAPBelizel",  
        "name": "sap_belize",  
        "path": "sap_belize"  
    },  
    "my_company_theme": {  
        "description": "MyCompanyTheme",  
        "name": "my_company_theme",  
        "path": "my_company_theme"  
    }  
}
```

3. In the `xs-app.json` file, add the route to the `themes` folder. For example:

#### Sample Code

```
{  
    "source": "/themes/(.*)",  
    "target": "$1",  
    "localDir": "themes"  
}
```

## 14.4.8.2 Adding an App to an Existing Launchpad Site

You can add an app to an existing launchpad site.

### Prerequisites

- You have a project in SAP Web IDE that contains a SAP Fiori app or a SAPUI5 app
- You have the site ID of the SAP Fiori Launchpad site to which you want to add the SAP Fiori app
- If your SAP Fiori app is configured for a User Authentication and Authorization system (UAA) that is different from the one used by the site, you need to assign your users with permissions for this UAA

### Context

The workflow involves the following steps:

1. Add the `site-content.json` file to your app project.  
This file defines the group in which the app's tile will appear, and the semantic object and action properties that are used to identify the app's tile. The same semanticObject and action pair needs to be configured also in the app's `manifest.json` file.  
You can also assign the app to a catalog in this file, using the application ID.
2. In the `manifest.json` file, you also define the scopes required for viewing the app's tile. These same scopes must be defined in the `xs-security.json` of the relevant UAA as well.
3. Configure the `package.json` file to use the site-app-server which is required for running your SAP Fiori application in the SAP Fiori launchpad.
4. In the `mtad.yaml` file, you add the binding to the portal service and specify the site ID of the launchpad site to which to add the app.

The following sections provide codes samples for each step.

### Procedure

1. Add the `site-content.json` file to your app project. In the following sample, you see how to the group, `MyGroup`, to the predefined role `Everyone`, which represents all logged-in users.

#### Sample Code

```
{  
  "roles": {  
    "Everyone": {  
      "_version": "1.0",  
      "identification": {  
        "id": "Everyone",  
        "namespace": "",  
        "entityType": "role"  
      }  
    }  
  }  
}
```

```

        },
        "payload": {
            "catalogs": [],
            "groups": [
                {
                    "id": "MyGroupId"
                }
            ]
        }
    },
    "applications": {
    },
    "groups": {
        "MyGroupId": {
            "identification": {
                "id": "MyGroupId",
                "i18n": "i18n/i18n.properties",
                "namespace": "",
                "title": "{{ MyGroupId_GROUP_TITLE }}"
            },
            "payload": {
                "tiles": [
                    {
                        "id": "SampleTile",
                        "subTitle": "{{ SampleTile_SUB_TITLE }}",
                        "target": {
                            "semanticObject": "myApp",
                            "action": "show",
                            "parameters": []
                        }
                    }
                ]
            }
        }
    }
}

```

2. In the `manifest.json` file, which is the SAPUI5 application descriptor, you define the scopes that determine the users who can see this app tile.

### Sample Code

```
{
...
    "sap.app": {
        "version": "1.3.0",
        "id": "MyUniqueAppID",
        "title": "My App",
        ...
        "crossNavigation": {
            "inbounds": {
                "appShow": {
                    "title": "{{TITLE}}",
                    "icon": "sap-icon://person-placeholder",
                    "semanticObject": "myApp",
                    "action": "show",
                }
            }
        }
    },
    ...
    "sap.ui5": {
        "componentName": "<SAPUI5 component name>"
    },
    ...
    "sap.platform.cf": {
        "uri": "/resources/webapp",
        "oAuthScopes": ["$XSAPPNAME.Manager"]
    }
}
```

The same scopes must also be defined in the `xs-security.json` of the relevant UAA:

#### Sample Code

```
...
{
  "xsappname": "backend",
  "scopes": [
    {
      "name": "$XSAPPNAME.Manager",
      "description": "Manager"
    },
    {
      "name": "$XSAPPNAME.User",
      "description": "User"
    }
  ],
  "role-templates": [
    {
      "name": "MyAppRoleTemplate",
      "description": "Sample role template",
      "scope-references": [
        "$XSAPPNAME.Manager"
      ]
    }
  ]
}
```

3. Configure the `package.json` file to use the `site-app-server`, which is required for running your SAP Fiori application in the SAP Fiori launchpad:

#### Sample Code

```
"dependencies": {
  "@sap/site-app-server": "^1.9.1"
},
"scripts": {
  "start": "node node_modules/@sap/site-app-server/index.js"
}
```

4. In the `mtad.yaml` file, you add the binding to the portal service and specify the site ID of the launchpad site to which to add the app:

#### Sample Code

```
- name: sample-app
  type: javascript.nodejs
  path: sample-app/
  parameters:
    memory: 32M
  requires:
    - name: MySampleBackendApp
      group: destinations
      properties:
        name: MySampleBackendApp
        url: ~{backendurl}
        forwardAuthToken: true
    - name: sap-portal-services-sample-app-content
    - name: MySampleBackendAppUAA
```

```

- name: backend
  type: javascript.nodejs
  path: backend/
  requires:
    - name: independent-backend-uaa
  provides:
    - name: backend
      properties:
        backendurl: ${default-url}
  parameters:
    memory: 32M
resources:
- name: sap-portal-services-sample-app-content
  type: com.sap.portal.site-content
  parameters:
    config:
      siteId : TargetSiteId
- name: independent-backend-uaa
  type: com.sap.xs.uaa-space
  parameters:
    config-path: security/xs-security.json

```

### 14.4.8.3 Roles and Authorizations

The application developer defines roles and authorizations by providing an `xs-security.json` file that contains a role template, one or more scopes, and attributes for each application.

Table 382: Terminology

Term	Description
Scopes	A list of the authorization scopes define in the application's <code>xs-security.json</code> file.
Attributes	A list of the attributes defined in the application's <code>xs-security.json</code> file, which are applied to a selected role. Depending on the value of the attributes defined, access to resources is either granted or restricted. For example, in a sales scenario, the attribute <code>region=emea</code> could be used to restricts access to the sales orders for the geographical region "EMEA".
Role Templates	A list of the role templates defined in the application's <code>xs-security.json</code> file. A role template defines the type of access permitted for an application. For example, the authorization scope, and any attributes that need to be applied.
Role	An instance of a role template. You build a role based on a role template and assign the role to a role collection.

Term	Description
Role Collection	Roles are assigned to role collections, which, in turn, are assigned, to SAP HANA users or SAML2 groups.

#### Guidelines

- Roles are reflected at the application level.
- Catalogs are relevant for personalization only. They are used to add more apps to the launchpad.
- Catalogs are assigned by default to the *Everyone* role and are not be filtered by roles.
- Applications are filtered by scopes. Each application can have one or more scopes assigned. An application can declare several permissions such as read, write, or a semantic type such as payment view.
- Groups are filtered by attributes.

## 14.4.9 Packaging and Deploying Applications

At the last stage of multi-target application development, you need to package your application and deploy it to the SAP HANA XS Advanced Model (XS Advanced) production system.

### Building and Packaging an Application for Deployment

#### Prerequisites

All the modules in your application are implemented and tested, and the MTA descriptor contains all the correct definitions and dependencies.

#### Procedure

Build the whole application by choosing *Build* from the context menu of the project root.

The application is packaged into an MTA archive named `<ID>_<version>.mtar`, which appears in the project under *mta\_archives* `<ID>`. where `ID` and `version` are the properties of your application defined in the MTA descriptor.

#### Related Information

[Inside an MTA Descriptor \[page 985\]](#)

# Deploying an Application

## Procedure

1. Right-click the .mtar file that you have created, and choose ► *Deploy* ► *To Cloud Foundry* ▶ or ► *Deploy* ► *To XS Advanced* ▶.
2. In the dialog box that opens, enter the properties of the space into which you want to deploy your application, and choose *Deploy*.

## 14.5 Customizing Your Project

Customize the developer experience for your SAP Web IDE project.

Any project customization you make in one session persists to the next. Customizations can vary from project to project.

### [Customizing JavaScript Beautifier Properties \[page 1093\]](#)

Beautify JavaScript files to reformat the source code to make it more readable.

### [Customizing Code Checking Rules \[page 1094\]](#)

SAP Web IDE provides validators to check your code. You can customize code checking for each project.

### [Configuring Mock Data Usage \[page 1095\]](#)

Configure settings to run an application using a client mock server.

### 14.5.1 Customizing JavaScript Beautifier Properties

Beautify JavaScript files to reformat the source code to make it more readable.

## Context

The beautifier configuration applies to all JavaScript, .js, .json, .xsjs, and .xsjslib files in your project.

## Procedure

1. From the context menu of any file in your project, select ► *Project Settings* ► *Beautifier* ► *JavaScript* ▶.
2. Determine how you want lines to be broken:

Option	Description
<i>Break lines on chained methods</i>	Select this option to add a line break when you add a chained method. By default, this option is cleared.
<i>New lines</i>	Set the maximum number of new lines between tokens. Choose from: <i>No New Lines</i> , <i>1</i> , <i>2</i> , <i>5</i> , <i>10</i> , <i>Unlimited New Lines</i> . For example, if there are two \ns between tokens, but you set this value to <i>1</i> , the second \n is removed. The default is <i>2</i> .
<i>Wrapping</i>	Determine the number of characters that triggers a line wrap. A line break is inserted before the first word that reaches the limit that you set. Choose from <i>Do not wrap lines</i> , <i>40</i> , <i>70</i> , <i>80</i> , <i>110</i> , <i>120</i> , or <i>140</i> (default).

3. Determine how you want indents to be handled:

Option	Description
<i>Keep array indentation</i>	Select this option to keep the indentation of arrays as is. Clear this option to remove array indentation. By default, this option is cleared.
<i>Indent with</i>	Select the length of each code indent: <i>a Tab</i> (default), <i>2 Spaces</i> , <i>3 Spaces</i> , <i>4 Spaces</i> , or <i>8 Spaces</i> .

4. Determine additional code formatting options:

Option	Description
<i>Space before conditional</i>	Select this option to insert spaces before conditional if/then statements. By default, this option is selected.
<i>Unescape printable chars encoded</i>	Select this option to decode printable characters that are encoded. By default, this option is cleared.
<i>Braces</i>	Determine the positioning of braces: <i>Braces with control statement</i> (default), <i>Braces on own line</i> , or <i>End braces on own line</i> .

5. Choose *OK*.

You can now beautify any open JavaScript file according to these customized properties, by choosing *Beautify* from the context menu.

## 14.5.2 Customizing Code Checking Rules

SAP Web IDE provides validators to check your code. You can customize code checking for each project.

You can customize and use the SAP Web IDE JavaScript validators and their rule configurations, or you can customize code checking for your project:

- Customize the SAP Web IDE JavaScript validator configuration with your own settings. For more information, see [Customizing JavaScript Validator Configuration \[page 1109\]](#).
- Configure your own JavaScript code checking rules and use them for your project. For more information, see [Creating JavaScript Code Checking Rules \[page 1108\]](#).
- Create your own code checking plugin, then choose the plugin in the corresponding code checking configuration pane.

## 14.5.3 Configuring Mock Data Usage

Configure settings to run an application using a client mock server.

### Context

#### i Note

If your application uses the application descriptor, the *Root URI* and the *Metadata File Path* are taken from the file and are not editable in the settings. If you need to make changes, they should be made in the application descriptor file.

### Procedure

1. From the project context menu, select *Project Settings*.
2. From the *Project Settings* options, select *Mock Data*.
3. In the *Root URI* field, enter the URL of the service used in your project.
  - If you selected a service when you created your project, SAP Web IDE automatically populates this field.
  - If the service URL changes, edit this field accordingly.
  - If there is no service URL, leave this field empty.
4. In the *Metadata File Path* field, enter the path to the service `metadata.xml` file to fetch mock data to use in your project.

#### i Note

If you do not specify the path, SAP Web IDE looks for the mock data in the `model/metadata.xml` file. If this path changes, edit this field accordingly.

5. Select one of the following options as your *Mock Data Source*:
  - Generated data (default)
  - JSON files
6. If you have provided an additional file containing custom mock requests extending the mock server, do the following:
  - a. Select the *Add custom mock requests* checkbox.
  - b. In the *Extension File Path* field, enter the path to the `mockRequests.js` file.
7. *Save* your changes.

## 14.6 Using Code Editors

SAP Web IDE provides customizable editors in which you edit the code for your applications.

There are various editors to support different types of files, including:

- JavaScript
- XML
- xsodata
- CDS
- hdbprocedure

When you open a file, it is displayed in the appropriate editor for the file type. All these editors support code completion.

### [Configuring the Code Editor \[page 1096\]](#)

Define the appearance and behavior of the code editor, and whether to autosave all changes in SAP Web IDE.

### [Working in the Code Editor \[page 1097\]](#)

Use keyboard shortcuts and context menus to easily edit and navigate through your code and code comments.

### [Generating JSDoc Comment Snippets \[page 1101\]](#)

You can generate a snippet for JavaScript function declaration that creates a template for documenting the function.

### [Using Code Completion \[page 1102\]](#)

The code completion feature assists you when you are writing your code by preventing typos and other common mistakes, and providing API reference information for SAPUI5 objects.

### [Checking Code \[page 1105\]](#)

SAP Web IDE performs code checking, also known as validation, and displays errors as annotations.

### [Locating Objects in Code \[page 1111\]](#)

The code editor allows you to locate objects or definitions of objects in code.

## 14.6.1 Configuring the Code Editor

Define the appearance and behavior of the code editor, and whether to autosave all changes in SAP Web IDE.

### Procedure



1. In the left sidebar, choose (Preferences), then choose *Code Editor*.
2. Customize the appearance and behavior.

Option	Description
<b>Appearance</b>	
<b>Code Editor Theme</b>	The theme of the code editor determines the background color and color of text.
<b>Font</b>	The monospace font for text in the code editor
<b>Font Size</b>	The font size of text in the code editor.
<b>Code Folding</b>	Hides code from a marked begin point to a marked end point or to the end of the file if end points are not used. Click in the gutter to mark begin and end points.
<b>Full line selection</b>	Line selection extends to the end of the line instead of to the end of the text in the line.
<b>Highlight selected word</b>	Highlights all occurrences of the word at the cursor position.
<b>Show invisible characters</b>	Shows white-space characters such as spaces, tabs, and new lines.
<b>Input Behavior</b>	
<b>Auto-pair characters</b>	Auto-pairs characters, such as quotation marks, parentheses, brackets, and so on.
<b>Use spaces for tab indentation</b>	Uses spaces for tab indentation, also known as soft tabs.

Test your settings in the preview.

3. Change the [Autosave](#) setting, if required.
4. Click [Save](#).

**Task overview:** [Using Code Editors \[page 1096\]](#)

## Related Information

- [Working in the Code Editor \[page 1097\]](#)
- [Generating JSDoc Comment Snippets \[page 1101\]](#)
- [Using Code Completion \[page 1102\]](#)
- [Checking Code \[page 1105\]](#)
- [Locating Objects in Code \[page 1111\]](#)

## 14.6.2 Working in the Code Editor

Use keyboard shortcuts and context menus to easily edit and navigate through your code and code comments.

Feature support depends on file types.

All edit actions are also available from the [Edit](#) menu.

**i** Note

The following keyboard shortcuts are for Microsoft Windows. For Mac OS, see [Keyboard Shortcuts \[page 972\]](#).

## Basic Navigating and Editing

Table 383:

Action	Keyboard Shortcut	Context Menu
Move one tab to the right	[Alt] + [R]	-
Move one tab to the left	[Alt] + [Q]	-
Close all tabs to the right of the open tab		<i>Close Tabs to the Right</i>
Undo edit	[Ctrl] + [Z]	<i>Undo</i>
Redo edit	[Ctrl] + [Y]	<i>Redo</i>
Show and hide control characters, such as space, tab, newline, and paragraph	[Ctrl] + [I] (uppercase i)	-
Move a line up	[Alt] + [Up arrow]	-
Move a line down	[Alt] + [Down arrow]	-
Move to a specific line	[Ctrl] + [L]	<i>Go to Line</i>
Copy a line to the line above or below	[Alt] + [Shift] + [Up arrow] [Alt] + [Shift] + [Down arrow]	-
Expand the entire hierarchy of file elements	[Alt] + [Shift] + [F2]	Gutter context menu: <i>Expand All</i>
Collapse the entire hierarchy of file elements	[Ctrl] + [Alt] + [F2]	Gutter context menu: <i>Collapse All</i>
Search for a string	[Ctrl] + [F]	<i>Find</i>
Search and replace a string	[Ctrl] + [H]	<i>Find and Replace</i>

## Commenting

Table 384:

Action	Keyboard Shortcut	Context Menu
<p>Comment out a line and restore to code.</p> <p>Comment syntax is appended to the code fragment automatically as a line. Each comment line is prefixed with <code>//</code>. For example:</p> <pre>//Use the comment style for short comments.</pre>	<code>Ctrl</code> + <code>/</code>	<i>Toggle Line Comment</i>
<p>Comment out a block and restore to code.</p> <p>Comment syntax is appended to the code fragment as a block using <code>/*</code> and <code>*/</code> to wrap the comment block. For example:</p> <pre>/* This is a comment block. Use this block comment style when comments span multiple lines.*/</pre> <pre>//Use the comment style for short comments.</pre>	<code>Ctrl</code> + <code>Shift</code> + <code>/</code>	<i>Toggle Block Comment</i>
<p>Flag JavaScript code with a TODO comment.</p> <p>A <code>//TODO</code> comment is added at the cursor location. If the line already contains a <code>//TODO</code> comment, the action is ignored.</p>	<code>Ctrl</code> + <code>Alt</code> + <code>T</code>	<i>Add TODO Comment</i>

## Beautify File Formatting

Beautify file formatting for JavaScript, JSON, XML, and CSS files using the context menu *Beautify* option or `Ctrl` + `Alt` + `B`.

- The XML beautifier formats code with line wrapping at 140 characters, and an indentation of one tab space. The beautifier is not configurable.
- The CSS beautifier formats indentation with one tab space and is not configurable.
- You can customize JavaScript beautifier settings for your project. For more information, see [Customizing JavaScript Beautifier Properties \[page 1093\]](#).

## Using Multiple Cursors

You can use multiple cursors to rename several variables at once or to insert the same text in multiple locations.

To add cursors in your file, press `Ctrl` and click at the required locations. You can then type text, which will appear in all the cursor locations.

To remove multiple cursors click anywhere in the file.

## Refactoring

You can change the name of a JavaScript function or variable, by using the context menu *Refactor* or `Alt` + `J`. Enter a valid new name and click *Rename*, and all references to the function or variable are automatically updated.

## Navigating from View to Controller

While editing an XML view, you can navigate to the view's controller by using the context menu *Open Controller*. The controller JavaScript file is opened in a new tab.

If you right-click the name of an event handler and select *Open Controller*, the controller is opened to the event handler function (if it is defined in the controller).

## Managing i18n Strings

In the context menu of an XML or JavaScript file, select *Open i18n* to open the i18n file defined in the `manifest.json`.

You can easily create a new string without having to open the i18n file by selecting *Create i18n String*. Add the string and its key, and they are added to the i18n file.

To edit a string, right-click on the i18n binding (for example, `{i18n>fileType_title}`) and select *Edit i18n String*.

You can get code completion when you need to select a string to add to an XML or JavaScript file. Put your cursor inside quotation marks and press `Ctrl` + `Space`, and get a list of strings defined in your i18n properties files. The properties files must be defined as i18n models in your manifest.

**Parent topic:** [Using Code Editors \[page 1096\]](#)

---

## Related Information

[Configuring the Code Editor \[page 1096\]](#)  
[Generating JSDoc Comment Snippets \[page 1101\]](#)  
[Using Code Completion \[page 1102\]](#)  
[Checking Code \[page 1105\]](#)  
[Locating Objects in Code \[page 1111\]](#)

### 14.6.3 Generating JSDoc Comment Snippets

You can generate a snippet for JavaScript function declaration that creates a template for documenting the function.

#### Context

The JSDoc comment snippet provides a template for documenting the function that you create. You can use code completion for variables within the snippet.

#### Procedure

1. Select a function, and press `Ctrl` + `Alt` + `J`, or choose *Generate JSDoc Comment* from the context menu.  
The snippet is displayed above the function code.
2. Enter information about the function variables in the corresponding placeholders in the comment.

**Task overview:** [Using Code Editors \[page 1096\]](#)

## Related Information

[Configuring the Code Editor \[page 1096\]](#)  
[Working in the Code Editor \[page 1097\]](#)  
[Using Code Completion \[page 1102\]](#)  
[Checking Code \[page 1105\]](#)  
[Locating Objects in Code \[page 1111\]](#)

## 14.6.4 Using Code Completion

The code completion feature assists you when you are writing your code by preventing typos and other common mistakes, and providing API reference information for SAPUI5 objects.

### Context

Code completion is triggered in one of the following ways:

- Auto hint code completion for JavaScript and XML files  
Based on your cursor location, SAP Web IDE establishes the context and displays a constrained list of suggestions. Use the corresponding icon to visually identify the code type being completed (that is, XML or JavaScript). This option is configurable; see [Enabling Code Completion \[page 1104\]](#).
- Manual code completion  
Supported for various file types including:`i18n` and `messagebundle` property files.
  - `i18n` and `messagebundle`
  - JavaScript
  - XML
  - `xsodata`
  - CDS
  - `hdbprocedure`No configuration is required.

#### Cross-file completion for JavaScript

The dependencies can be defined using a dependency declaration in either of the following ways: SAP Web IDE can provide suggestions for functions (methods) that are defined in other modules on which the source file depends. Suggestions include methods declared explicitly in the dependency file and generated methods for metadata properties, associations, aggregations, and events of SAPUI5 controls.

- Use `SAP Web IDE` can provide suggestions for functions (methods) that are defined in `jQuery.sap.require(<moduleId1>, <moduleId2>, ...)`. The dependency (target) file should contain the module declaration `jQuery.sap.declare(<moduleId>)`.
- Use `sap.ui.define([<dependencyFile1>, <dependencyFile2>, ...], function(d1, d2, ...) { })`; where `dependencyFile` is the relative file path in the current project or the logical path to the library module. The dependency (target) files should contain a module declaration using `sap.ui.define` or Asynchronous Module Definition (AMD).

#### **i** Note

Suggestions can be provided only if the dependency file is visible in the current project

### Code completion for SAPUI5

The code editor provides code completion suggestions for:

- Properties in SAPUI5 methods. For example, suggestions for `Op5.createPageObjects`, include all the properties for `baseClass`, `namespace`, `actions`, and `assertions`.

- SAPUI5 static and instance methods when you use dependency declarations with Asynchronous Module Definition (AMD).
- SAPUI5 module dependencies in AMD. Proposed suggestions are for the element type of module according to the element name entered.

API reference information for SAPUI5 objects is shown in a tooltip when you hover over a suggestion. If more information is available, a [More Details](#) button is provided that links to the full SAPUI5 documentation for the current element.

Code completion for SAPUI5 is dependent on the SAPUI5 version selected in the project settings.

## Procedure

To use auto hint code completion:

1. Create or open an XML or JavaScript file.
2. Place your cursor in the context for which you require assistance, and press **Ctrl** + **Space**.

A list of suggestions is displayed. Deprecated APIs are indicated with strikethrough text. Hover over a suggestion and the tooltip also indicates whether an API is deprecated or experimental.

3. Scroll through the list, select the appropriate fragment, and press **Enter**.  
The code fragment is added at the cursor location.

### Example

This example of embedded type code completion shows how you can attach a type to your variable definition to enable presentation of appropriate code completion suggestions for the variable.

Define the variable type in a JSDoc comment before the variable definition, and then press **Ctrl** + **Space** for code assistance. When you hover over a suggestion (e.g., `setIconDensityAware`), API reference information is shown in a tooltip.

The screenshot shows a code editor with the following snippet:

```
formatConcatenateNameId : function(sName, sID) {
    /** @type sap.m.Button */
    var x = sap.ui.getCore().byId("btnBack");
    x.s
}
```

The cursor is positioned after the letter 's' in 'x.s'. A code completion dropdown is open, listing several methods for the `sap.m.Button` type:

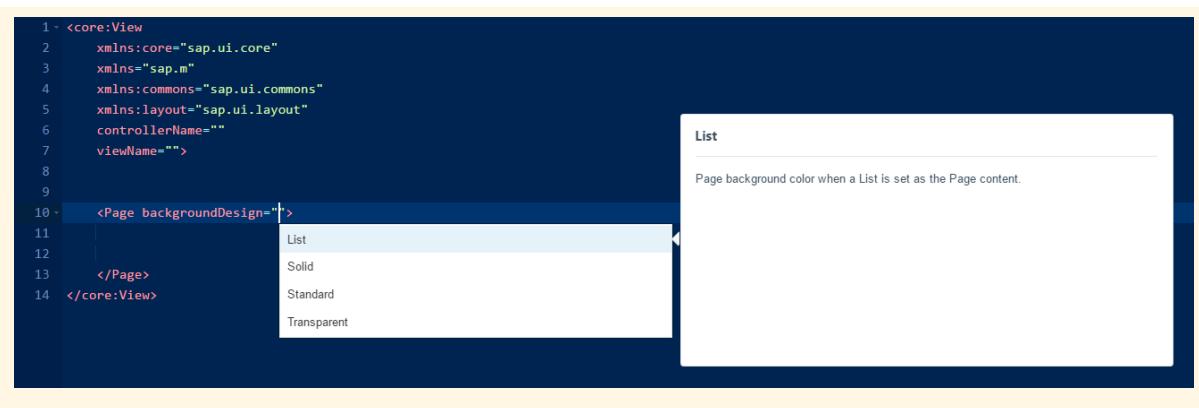
- `M setActiveIcon(sActiveIcon) : sap.m.Button`
- `M setEnabled(bEnabled) : sap.m.Button`
- `M setIcon(sIcon) : sap.m.Button`
- `M setIconDensityAware(bIconDensityAware) : sap.m.Button` (highlighted)
- `M setIconFirst(bIconFirst) : sap.m.Button`
- `M setText(sText) : sap.m.Button`
- `M setTextDirection(sTextDirection) : sap.m.Button`

On the right, a tooltip for the `setIconDensityAware` method is displayed:

**setIconDensityAware(bIconDensityAware) : sap.m.Button**

Sets a new value for property `iconDensityAware`. By default, this is set to true but then one or more requests are sent trying to get the density perfect version of image if this version of image doesn't exist on the server. If only one version of image is provided, set this value to false to avoid the attempt of fetching density perfect image. When called with a value of `null` or `undefined`, the default value of the property will be restored. Default value is `true`.

In the next example, to determine which values you can use for `<Page backgroundDesign="">`, move the cursor between the quotation marks as shown below, and launch code completion. The options for the attribute are listed. Hover over one of the options to get a tooltip with API reference information about that option.



**Task overview:** [Using Code Editors \[page 1096\]](#)

## Related Information

- [Configuring the Code Editor \[page 1096\]](#)
- [Working in the Code Editor \[page 1097\]](#)
- [Generating JSDoc Comment Snippets \[page 1101\]](#)
- [Checking Code \[page 1105\]](#)
- [Locating Objects in Code \[page 1111\]](#)

### 14.6.4.1 Configuring Code Completion

You can enable code completion as you type (auto hint) for JavaScript and XML files.

#### Procedure

1. To open the *Preferences* perspective, in the left sidebar, choose (Preferences).
2. Click *Code Completion* and select the *Enable* checkbox.
3. Click *Save*.

## 14.6.5 Checking Code

SAP Web IDE performs code checking, also known as validation, and displays errors as annotations.

By default, when you open a JavaScript, JSON, or XML file, code checking is triggered and the detected code issues are displayed as annotations within the editor. Understand how to configure and use code checking in your project.

### [Configuring Code Checking \[page 1105\]](#)

Configure when to trigger code checking, also known as code validation, and the level of messages to display.

### [Code Checking Annotations \[page 1106\]](#)

If a syntax error is found during editing, the relevant lines are annotated with flags indicating the error severity. Understand the severity level of these flags so that when you open a file, you know how to interpret these annotations.

### [JavaScript Validation \[page 1107\]](#)

Review the default JavaScript validator configuration. Customizations always override these defaults.

### [YAML Validation \[page 110\]](#)

A built-in client-side YAML validator is integrated with the code editor.

### [Using the Problems View \[page 110\]](#)

View information about problems in the projects in your workspace.

**Parent topic:** [Using Code Editors \[page 1096\]](#)

## Related Information

[Configuring the Code Editor \[page 1096\]](#)

[Working in the Code Editor \[page 1097\]](#)

[Generating JSDoc Comment Snippets \[page 1101\]](#)

[Using Code Completion \[page 1102\]](#)

[Locating Objects in Code \[page 1111\]](#)

## 14.6.5.1 Configuring Code Checking

Configure when to trigger code checking, also known as code validation, and the level of messages to display.

## Context

By default, code checking is enabled when you make changes to your code, and all messages are displayed. You can change these defaults.

### Note

You can customize code checking rules for each project. For more information about customizing and using code checking, see [Checking Code \[page 1105\]](#).

## Procedure

1. To open the *Preferences* perspective, in the left sidebar, choose  (Preferences).
2. Select *Code Check*.
3. In the *Run Code Check* section, select when to display code checking annotations:
  - Choose *On Save* to display annotations only when you save your file.
  - Choose *On Change* to display annotations every time you make a change to your code.
4. In the *Code Check Level* section, select which messages to display:
  - All* displays all errors, warnings, and information messages
  - Error* displays error messages only
  - Error and Warning* displays error and warning messages
  - Disable* suppresses message display
5. Choose *Save*.

**Task overview:** [Checking Code \[page 1105\]](#)

## Related Information

[Code Checking Annotations \[page 1106\]](#)

[JavaScript Validation \[page 1107\]](#)

[YAML Validation \[page 1110\]](#)

[Using the Problems View \[page 1110\]](#)

## 14.6.5.2 Code Checking Annotations

If a syntax error is found during editing, the relevant lines are annotated with flags indicating the error severity. Understand the severity level of these flags so that when you open a file, you know how to interpret these annotations.

- All syntax errors are annotated on code line and tab levels and each annotation is colored according to its severity.

Color	Severity
Red	Error
Yellow	Warning
Blue	Information

- When you hover over an annotation, a tooltip displays one or more detected issues and possible resolutions for the annotated line.

The detected issues of an annotated line are categorized (for easy identification), identified, and described, so that you can determine how best to resolve an issue.

**Categories** Used to classify the issue. For example, possible error, best practice, stylistic issue, and others.

**Rule IDs** Used to define the logic for detection or list known issue exceptions. For example, `semi` is a rule ID for ESLint.

**Messages** Detail the issue or suggest a possible resolution.

You can resolve the issue and continue development with iterative fixes.

**Parent topic:** [Checking Code \[page 1105\]](#)

## Related Information

[Configuring Code Checking \[page 1105\]](#)

[JavaScript Validation \[page 1107\]](#)

[YAML Validation \[page 1110\]](#)

[Using the Problems View \[page 1110\]](#)

## 14.6.5.3 JavaScript Validation

Review the default JavaScript validator configuration. Customizations always override these defaults.

The SAP Web IDE default JavaScript validator uses ESLint code checking.

- |                                      |   |
|--------------------------------------|---|
| <b>ESLint Rule</b>                   | See these resources for additional support:   |
| <b>Execution Defaults</b>            | <ul style="list-style-type: none"> <li>For ESLint configuration details, see <a href="#">Configuring ESLint</a>.</li> <li>For ESLint rule information, see <a href="#">Rules</a>.</li> </ul>  |
| <b>ESLint Rule Metadata Defaults</b> | <p>Attributes can have multiple supported values. Use the following:</p> <ul style="list-style-type: none"> <li><code>severity</code> attribute to define whether an issue renders as <code>error</code>, <code>warning</code> (default), or <code>information</code>.</li> </ul> |

- category attribute for a better semantic classification: possible error, best practice, stylistic issue, and others.
- help attribute (optional) to override the default help links listed.

**Parent topic:** [Checking Code \[page 1105\]](#)

## Related Information

[Configuring Code Checking \[page 1105\]](#)

[Code Checking Annotations \[page 1106\]](#)

[YAML Validation \[page 1110\]](#)

[Using the Problems View \[page 1110\]](#)

### 14.6.5.3.1 Creating JavaScript Code Checking Rules

You can replace the SAP Web IDE default rule configurations in the JavaScript validator by configuring your own code checking rule configurations.

#### Context

The SAP Web IDE default JavaScript validator uses ESLint code checking. You can override the default rule configuration by creating a folder containing your custom rules.

#### Procedure

1. Choose .
2. Name the new folder and add your customized code checking rules to this folder.

##### Note

The rule files in the folder must be JavaScript files.

3. From the context menu of any file in your project, choose *Project Settings*.
4. From the *Project Settings* options, select .
5. Next to the *Custom Rules Folder* field, choose *Browse*. A popup window displays the projects in the workspace.
6. Open your project and select the rules folder that you created.

7. Choose *OK*.

Your custom rules appear in the *Rules* table instead of the default rules. All the rules are disabled by default.

8. To implement your custom code checking rules, in the *Rules* table, enable each one.
9. Choose *Save* to save the new rule configuration for the project.

### 14.6.5.3.2 Customizing JavaScript Validator Configuration

You can customize the configuration of SAP Web IDE JavaScript validators.

#### Context

You can customize the configuration and rules of SAP Web IDE JavaScript validators directly in the SAP Web IDE UI.

If you change the default settings and turn on or off specific rules, this information is stored in a `.eslintrc` file in the root folder of the project. If you have already configured ESLint rules in another development environment, you can import your `.eslintrc` file into the root folder, instead of using the UI. Additional SAP Web IDE-specific information about each rule – such as category, severity and help URL – is stored in an automatically generated `.eslintrc.ext` file, also in the root folder.

#### Procedure

1. From the context menu of any file in your project, choose *Project Settings*.
  2. From the *Project Settings* options, choose *Code Checking* *JavaScript*.
- The default JavaScript validator opens displaying the SAP Web IDE default rules.
3. In the *Validator Configuration* field, you can define the validator configuration for globals and environments for the selected validator. The configuration should conform to `.json` file structure.
  4. In the *Rules* table, configure the rules for the selected validator as follows:
    - a. Enable each rule that you want to use to check your code by selecting the checkbox by the rule name.
    - b. Configure the error level of the rules by setting the severity and category.
    - c. Use the help link for each rule to access detailed rule information about how you can fix the detected issue.
  5. Choose *Save*. The enabled rules will be implemented when you write your code.

#### Note

You can restore the default validator configuration and rules by clicking the *Reset* button next to the *Validator* field.

## 14.6.5.4 YAML Validation

A built-in client-side YAML validator is integrated with the code editor.

YAML files, such as MTA descriptors (`mta.yaml`), are automatically validated as you edit. Errors and warnings are indicated in the file tab and next to the line numbers with red and yellow markers respectively. When you hover over a marker, an error or warning message is displayed.

 Note

Only one error is indicated at a time. Once you fix it, the next one appears.

The client-side validator only checks the file syntax according to the YAML 1.2 specification. It does not check the MTA schema.

The syntax and schema of the `mta.yaml` files are validated during the build process on the server side according to the YAML 1.1 specification. The server-side validation messages appear in the Build console and logs.

[Parent topic: Checking Code \[page 1105\]](#)

## Related Information

[Configuring Code Checking \[page 1105\]](#)

[Code Checking Annotations \[page 1106\]](#)

[JavaScript Validation \[page 1107\]](#)

[Using the Problems View \[page 1110\]](#)

## 14.6.5.5 Using the Problems View

View information about problems in the projects in your workspace.

The *Problems* view displays information about problems in the modules in your projects.

The problem analysis is triggered by one of the following actions:

- Automatically whenever you select a project or a module when the *Problems* view is open.
- Manually by clicking the *Analyze and display problems* icon in the *Problems* view.
- When you build an HDB module or an entire project.

The *Problems* view displays the following information:

- The total error count and the error count for each severity for all the analyzed files. The error count does not change when you filter the display or change the severity selection.
- The severity of each problem. You can filter the list according to severity.
- The problem description with a link to more information.

- A link to the file that contains the problem. Click the link to open the file in the workspace at the location of the problem.
- The full path to the folder that contains the file with the problem.
- The problem category.

For HTML5 and Node.js modules, the [Problems](#) view is dynamically updated when you edit, add, and delete files or delete a project within the scope of the analysis.

For JavaScript, you can exclude specific files or folders from the analysis:

1. Create a text file with the suffix `eslintignore` and put it under the root project folder.
2. In the file, enter the names of all the files and folders to ignore. Each entry should be on a separate line.  
The next time that you trigger an analysis, these files and folders will be excluded from the analysis.

**Parent topic:** [Checking Code \[page 1105\]](#)

## Related Information

[Configuring Code Checking \[page 1105\]](#)

[Code Checking Annotations \[page 1106\]](#)

[JavaScript Validation \[page 1107\]](#)

[YAML Validation \[page 1110\]](#)

## 14.6.6 Locating Objects in Code

The code editor allows you to locate objects or definitions of objects in code.

### Context

Methods can be defined in other modules on which the source file depends. You can use [Goto Definition](#) to navigate to methods declared explicitly in the dependency file and generated methods for metadata properties, associations, aggregations, and events of SAPUI5 controls.

The dependencies can be defined using a dependency declaration in either of the following ways:

- Use `jQuery.sap.require(<moduleId1>, <moduleId2>, ...)`. The dependency (target) file should contain the module declaration `jQuery.sap.declare(<moduleId>)`.
- Use `sap.ui.define([<dependencyFile1>, <dependencyFile2>, ...], function(d1, d2, ...) { })`; where `dependencyFile` is the relative file path in the current project or the logical path to the library module. The dependency (target) files should contain a module declaration using `sap.ui.define` or Asynchronous Module Definition (AMD).

Suggestions can be provided only if the dependency file is visible in the current project.

You can:

- Use goto services to locate the definition action of a user-defined object (for example, a variable, function, object, or property) in JavaScript files for in a project. Goto searches the active file as well as all files in the same project.

#### **i** Note

Goto is not supported for JavaScript native keywords (for example, `var`, `window`, `JSON`, or SAPUI5 library objects (for example, `sap`, `sap.ui`, `sap.ui.core`), as they are not user-defined objects. If you attempt to use goto with a restricted object, a message appears in the console pane to indicate that the definition was not found.

- Highlight all instances of the selected object in the active file.

## Procedure

1. To locate the definition using goto services:

- a. Validate whether the object is user-defined by placing the cursor over the user-defined object and pressing **CTRL** + **ALT**.  
If the object has a user-defined definition, it appears underlined in blue. Otherwise, the console reports that the definition cannot be found.
- b. To go to the location, press **CTRL** + **ALT** and click the underlined object.  
If the definition is in the current file, the definition is highlighted. Otherwise, the correct file is first opened and the definition is highlighted.

#### **i** Note

You can also conflate both of these steps into a single action by selecting the object and pressing **CTRL** + **ALT** + **G**, or selecting the object and choosing ► *Edit* ► *JavaScript* ► *Goto Definition* ▾.

2. To locate all instances of an object in an open file, double-click the object.

All instances of the object are highlighted in a blue box.

For example, if you select the first instance of the string `content`, all remaining instances are identified in a blue box.



```
index.html
1 <!DOCTYPE HTML>
2 <html>
3 <head>
4   <meta http-equiv="X-UA-Compatible" content="IE=edge">
5
6   <script src="resources/sap-ui-core.js">
7     id="sap-ui-bootstrap"
8     data-sap-ui-libs="sap.ui.commons"
9     data-sap-ui-theme="sap_bluecrystal">
10  </script>
11  <!--add sap.ui.table,sap.ui.ux3 and/or other libraries to 'data-sap-ui-libs' if required-->
12
13  <script>
14    sap.ui.localResources("view");
15    var view = sap.ui.view({id:"dfsd", viewName:"view.dfsd", type:sap.ui.core.mvc.ViewType.XML});
16    view.placeAt("content");
17  </script>
18
19</head>
20<body class="sapUiBody" role="application">
21  <div id="content"></div>
22</body>
23</html>
```

**Task overview:** [Using Code Editors \[page 1096\]](#)

## Related Information

- [Configuring the Code Editor \[page 1096\]](#)
- [Working in the Code Editor \[page 1097\]](#)
- [Generating JSDoc Comment Snippets \[page 1101\]](#)
- [Using Code Completion \[page 1102\]](#)
- [Checking Code \[page 1105\]](#)

## 14.7 Using Source Control (Git)

The SAP Web IDE includes the Git source control system, letting you connect and interact with remote Git repositories.

### Git Tools

SAP Web IDE provides a graphical user interface for executing Git commands and managing your source control and versioning. The following are the main tools for working with Git:

- **Git Menu:** Access the menu from  . The menu includes the ability to clone a repository, as well as other Git commands for working with cloned repository.

- **Git Pane:** The Git pane provides a graphic user interface for executing Git commands on a specific Git repository, as well as a status table that lists all the uncommitted changes you've made to your project.

The screenshot shows the SAP Web IDE's Git pane. At the top, it displays the repository name "myTest" and the current branch "testBranch". Below the repository information are several circular icons representing Git commands: Pull, Fetch, Rebase, Merge, Show Stash, Reset, and Fetch from Gerrit. A "Commit" section follows, containing "Stage All" and "Discard All Changes" buttons. The main area is a "Status table" with a red border, listing file changes. The table has columns for Status, Name, Stage, and Discard. Two files are listed: "M webapp/Component.js" and "M webapp/index.html", each with a "Stage" button and a "Discard" button. Below the table is a "Commit Description" input field with placeholder text "Insert commit description". At the bottom, a note says "A Change ID for Gerrit will not be added". There are three buttons: "Commit and Push", "Commit", "Push", and "Stash".

Status	Name	Stage	Discard
M	webapp/Component.js	<input type="checkbox"/>	<input checked="" type="checkbox"/>
M	webapp/index.html	<input type="checkbox"/>	<input checked="" type="checkbox"/>

Status table

To open the *Git* pane:

1. From the workspace, select a Git repository

2. From the right sidebar, choose (Git pane).

- **Git History Pane:** Lets you view the commits for different branches.

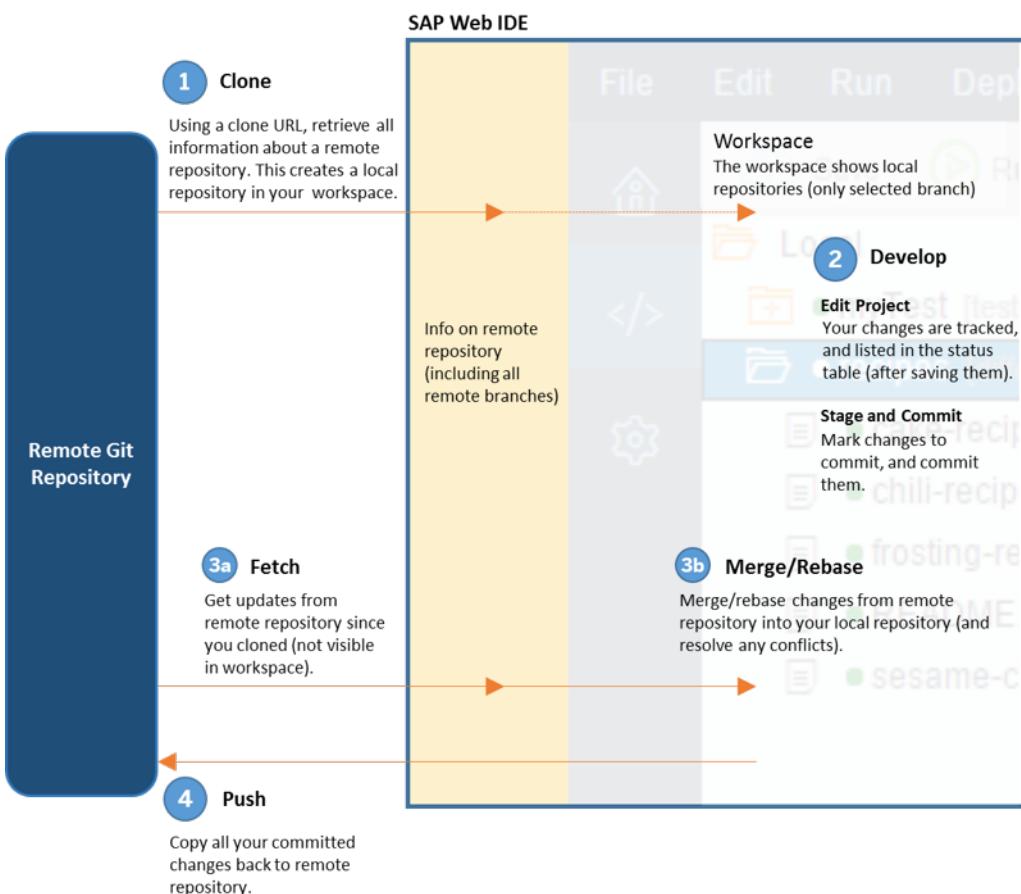
To open the *Git History* pane:

1. From the workspace, select a Git repository
2. From the right sidebar, choose  (Git History pane).

## Workflow

Using Git with SAP Web IDE is easy. The basic workflow is as follows:

1. **Clone:** Clone a repository from a remote Git source control system. All the information about the repository is copied, and a local master branch is created and is visible in your workspace. If the remote repository has several branches, you can create additional local branches based on those remote branches.  
You can also create a new repository by creating a new project in your SAP Web IDE, and choosing  *Git*  *Initialize Local Repository*  You can then connect the local repository to a remote repository by choosing  *Set Remote* 
2. **Develop:** Once you have the code, you can develop – add files, delete files, modify files. Your changes are visible in the status table of the *Git* pane. When you are ready, you can stage your changes and commit them.
3. **Fetch and Merge/Rebase:** (Optional) Before sending back your changes to the remote repository, you can fetch all the changes made by others. Then you can merge or rebase the changes into your changes to make sure there are no conflicts. If there are conflicts, you can adjust your code.
4. **Push:** Add your changes to the remote repository.



### [Setting Up Git \[page 1117\]](#)

To use source control in your SAP Web IDE project, your user name and email address must be set for your Git account.

### [Cloning Repositories \[page 1119\]](#)

You can clone an existing Git repository into your workspace.

### [Initializing a Local Git Repository \[page 1120\]](#)

You can initialize a local repository for any project that is not already connected to a Git repository.

### [Setting a Remote Repository \[page 1120\]](#)

After initializing a local repository for your project, you likely will want to set a remote repository for your project, so you can push your work to a central Git repository.

### [Fetching Changes \[page 1121\]](#)

Fetching enables you to download objects and references from another repository into your local repository. You can then merge or rebase the changes into your project.

### [Rebasing Changes \[page 1122\]](#)

Rebasing enables you to take all the committed changes from one branch and incorporate them into a different branch.

### [Merging Changes \[page 1123\]](#)

You can incorporate all the changes from one branch into another in a single commit.

### [Pulling Changes \[page 1124\]](#)

Pulling is the same as fetching and merging. Pulling enables you to download objects and references from another repository into your local repository, and then merge the changes into your project.

#### [Staging Files \[page 1124\]](#)

The status table shows changed files, and lets you select files to stage.

#### [Committing Changes \[page 1126\]](#)

You can commit changes to the repository locally.

#### [Pushing Changes \[page 1127\]](#)

The Push option incorporates all unsynced committed changes into the remote branch of the currently checked-out local branch. The number of unsynced committed changes is displayed next to the repository name. All tags created within the open repository are pushed.

#### [Using Multiple Local Branches \[page 1127\]](#)

From the Git pane, you can check out a local branch, add a new local branch, and remove a local branch.

#### [Creating a Remote Branch \[page 1129\]](#)

You can create a new branch in the remote repository.

#### [Viewing Git History \[page 1130\]](#)

From the Git *History* pane, you can explore the history of committed changes that were made for repositories, folders, and files in a specific project.

#### [Setting Up Git to Work with Gerrit \[page 1131\]](#)

Gerrit is a web-based software code review tool for reviewing, approving, or rejecting changes to the source code developed by your colleagues. Gerrit works as an intermediate environment for source control between the local environment and the remote Git repository.

## Related Information

### [Git Troubleshooting \[page 1134\]](#)

## 14.7.1 Setting Up Git

To use source control in your SAP Web IDE project, your user name and email address must be set for your Git account.

## Context

### **i** Note

If your Git server uses certificate authentication, you need to upload the server certificate to the SAP Web IDE server. For more information, see [Managing SSL Certificates \[page 966\]](#).

By carrying out the following steps, you can either set your Git user settings or check whether they are already set correctly.

**i Note**

Your user name and email address will be stored on the remote Git server. This is mandatory for Git operations and cannot be undone.

## Procedure

1. Open SAP Web IDE in one of the supported browsers using the subscription URL.



2. Choose **Preferences** (Preferences) and select *Git Settings*.
3. Enter your email address and name.

**i Note**

The email address field is case-sensitive.

**i Note**

If you have not set your Git user name and email address, SAP Web IDE extracts this information from the identity provider defined in your account and pre-populates these fields in the *Git Settings* page.

4. Choose **Save**.

### 14.7.1.1 Git Decorations

Any change in a file's status is reflected in the workspace by decorations.

The table below shows the meaning of these decorations:

Table 385:

Decoration	Meaning
	Committed file
	Modified file that has not been staged
	Modified file that has been staged
	New file
	Folder containing deleted files

Decoration	Meaning
	File with merge conflicts

## 14.7.2 Cloning Repositories

You can clone an existing Git repository into your workspace.

### Procedure

1. From the *File* menu, choose  *Git* .
2. In the *URL* field, enter the Git repository URL and press .
3. Enter your Git repository user name and password.
4. If your remote Git system works with Gerrit, select *Add configuration for Gerrit*.
5. Choose  . The cloning starts. When the process is finished, the content of the repository appears in the workspace.

## 14.7.2.1 Configuring Git Repositories

You can configure the Git repository for your project by creating new entries or deleting and editing existing entries.

### Procedure

1. Right-click a Git repository, and choose  .
2. Configure the Git repository as follows:
  - To create a new entry, choose *Add Entry* and type the relevant values in the *Key* and *Value* fields.

#### Note

Use the following format for the *Key* entry: <section>.<name>. For example, `user.name`

#### Note

The *Key* field is mandatory and cannot be duplicated.

- To edit an entry, choose .

- To delete an entry, choose  (Delete).
3. Choose  (Save).

### 14.7.3 Initializing a Local Git Repository

You can initialize a local repository for any project that is not already connected to a Git repository.

#### Context

You can create an empty local repository for your project. This local repository can then be connected to a remote repository.

#### Procedure

1. Select the desired project.
2. Right-click and select  *Git* .

#### Related Information

[Setting a Remote Repository \[page 1120\]](#)

### 14.7.4 Setting a Remote Repository

After initializing a local repository for your project, you likely will want to set a remote repository for your project, so you can push your work to a central Git repository.

#### Procedure

1. Select a project that has been initialized as a local Git repository (with   .
2. Right-click and select  *Git* .
3. Enter the remote repository URL, and a name for the remote repository.

### **i** Note

Once you have set a remote repository, you can change it from the project settings.

1. Right-click the desired project and select *Project Settings*.
2. Select *Git Repository Configuration*.
3. Click *Add Entry* and add the following:

Table 386:

Key	Value
remote.origin.fetch	<code>+refs/heads/*:refs/remotes/origin/*</code>
remote.origin.url	<Remote Git repository URL>
branch.master.merge	<code>refs/heads/master</code>
branch.master.remote	<code>origin</code>

### **i** Note

When you connect your local repository to a remote repository, before you push your changes you need to first perform a fetch. Otherwise you will not be able to see the remote branches, such as `origin/master`.

4. If your remote Git system works with Gerrit, select *Add configuration for Gerrit*.
5. Choose *OK*.

## Related Information

[Initializing a Local Git Repository \[page 1120\]](#)

### 14.7.5 Fetching Changes

Fetching enables you to download objects and references from another repository into your local repository. You can then merge or rebase the changes into your project.

## Procedure

1. In the *Git Pane*, choose *Fetch*.
2. Enter your Git repository user name and password.

3. Choose *Remember Me* to avoid being asked for credentials again in this session..
4. Choose *OK*.
5. Choose *OK*.

## 14.7.6 Rebasing Changes

Rebasing enables you to take all the committed changes from one branch and incorporate them into a different branch.

### Procedure

1. In the *Git Pane*, choose *Rebase*.
2. Select the branch from which you want to obtain the changes.

**i** Note

The branch that is currently checked out is automatically disabled. It cannot be selected. By default, the corresponding remote branch is selected.

3. Choose *OK*. The latest changes are integrated and shown in your workspace.

**i** Note

Rebase can fail due to conflicts between the current branch and the branch whose changes you want to incorporate. When conflicts are identified, the Git pane switches to *Rebase Interactive* mode and different actions are available to enable conflict resolution.

### Related Information

[Rebase Interactive Mode \[page 1122\]](#)

## 14.7.6.1 Rebase Interactive Mode

When conflicts occur while rebasing a branch, rebase interactive mode is triggered.

When rebase interactive mode is enabled, the caption [`rebase in progress`] is displayed next to the repository name in the Git pane.

The following actions can be executed in the rebase interactive state:

- Continue the rebase process.

1. Fix the conflicts that are visible in the status table and save the fixed files.
  2. Stage the fixed files.
  3. Choose *Continue*.
  4. If all conflicts are resolved, the repository returns to its normal mode. If errors still exist, repeat the procedure.
- Abort the rebase process.  
Choose *Abort*.
  - Skip a specific conflicting commit.  
Choose *Skip Patch*.
  - Reset changes. Resetting changes discards all staged and unstaged changes on the current local branch, so that it is identical to the remote branch.  
Choose *Reset*.

## 14.7.7 Merging Changes

You can incorporate all the changes from one branch into another in a single commit.

### Procedure

1. In the *Git Pane*, choose *Merge*.
2. Select the branch from which you want to obtain the changes.

**i** Note

The branch that is currently checked out is automatically disabled. It cannot be selected. By default, the corresponding remote branch is selected.

3. Choose *OK*. The latest changes are integrated and shown in your SAP Web IDE workspace.

**i** Note

Merge operations can fail due to conflicts between the current branch and the branch you chose from which to incorporate the changes.

## 14.7.8 Pulling Changes

Pulling is the same as fetching and merging. Pulling enables you to download objects and references from another repository into your local repository, and then merge the changes into your project.

### Procedure

1. In the *Git Pane*, choose *Pull*.
2. Enter your Git repository user name and password.
3. Choose *Remember Me* to avoid being asked for credentials again in this session.
4. Choose *OK*. The changes are fetched from the specific branch and merged into your local checked-out branch.

## 14.7.9 Staging Files

The status table shows changed files, and lets you select files to stage.

### Context

Whenever a file is updated, added, or deleted, it appears in the status table but is not staged. After staging, you can commit the staged files.

### Procedure

In the status table of the *Git Pane*, choose the *Stage* checkbox in the row that contains the change that you want to stage.

**i** Note

To stage all files in the status table, select the *Stage All* checkbox at the top of the table.

### Related Information

[Discarding Changes \[page 1126\]](#)

## 14.7.9.1 Comparing Code

You can compare different versions of your code.

### Context

Use the SAP Web IDE compare editor to compare a modified version of your code with a version from the staging tables in the *Git* pane.

### Procedure

1. In the status table in the *Git Pane*, double-click the file in the staging table row or right-click it and choose *Compare* from the context menu.

The title of each pane indicates whether it contains the latest editable version or the previous read-only version. The read-only file is indicated by background shading.

#### *i* Note

You can only compare files that are modified (status M) or that have conflicts (status C).

2. Compare your modified code with the original code as follows:
  - Choose *Next* and *Previous* to navigate between the highlighted differences in the code of the original and modified versions. As you navigate through the changes in the file, the color of the changed code deepens to indicate your cursor position.
  - Choose *Copy from right to left* to move selected code from the original version to the modified version. The highlighted lines on the right side will replace the highlighted lines on the left side.
3. Edit your code and resolve any conflicts that may have occurred, especially after a rebase or merge operation.

#### *i* Note

If there is a conflict, decide which version of code to continue with. You may want to use a combination of both versions. The modified version is editable and you will eventually push this version.

## 14.7.9.2 Discarding Changes

Discarding removes all changes from an existing file in the local environment. For example, discarding a new file deletes the file from the branch.

### Context

**i** Note

Only unstaged files can be discarded.

### Procedure

In the status table of the *Git Pane*, choose in the row that contains the change that you want to discard.

All changes that you made to the file are removed.

**i** Note

To discard all files, choose *Discard All* at the top of the table. All unstaged files in the table in the *Commit* section are discarded.

## 14.7.10 Committing Changes

You can commit changes to the repository locally.

### Procedure

1. In the *Git Pane* status table, select the *Stage* checkbox for the files you want to stage (or click *Stage All* above the table).
2. Enter a description of the change in *Commit Description*.
3. If you want to add the current changes to the last commit, select the *Amend Changes* checkbox. The commit description of the last committed change appears in *Commit Description*, which you can modify.
4. Choose *Commit*. The changes are committed locally, and one is added to the counter for unsynced commits at the top of the *Git Pane*, next to the repository name.

## Results

The user name and email listed for your commits are the user name and email from your SAP HANA system, not the user name and email from your Git system.

### 14.7.11 Pushing Changes

The Push option incorporates all unsynced committed changes into the remote branch of the currently checked-out local branch. The number of unsynced committed changes is displayed next to the repository name. All tags created within the open repository are pushed.

## Procedure

1. In the *Git Pane*, choose *Push*.

For your convenience, you can instead use *Commit and Push* to commit the currently staged changes and then immediately push them to a remote branch. Before choosing *Commit and Push*, remember to stage your changes and to add a description for the commit.

2. Choose one of the following from the dropdown list:
  - *origin/<remote branch>* if your local branch is based on a specific remote branch.
  - *Remote Branch* to select a different remote branch.
3. Enter your Git repository user name and password.
4. Choose *Remember Me* to avoid being asked for credentials again in this session.

### 14.7.12 Using Multiple Local Branches

From the Git pane, you can check out a local branch, add a new local branch, and remove a local branch.

## Checking Out a Local Branch

## Procedure

In the *Git Pane*, select the desired local branch.

### i Note

If you have uncommitted changes in your workspace, a dialog box containing the list of conflicting files opens. Choose *Cancel* to abort, or *Reset and Checkout* to remove all uncommitted changes.

The selected branch is checked out. The name of the selected branch is shown in the workspace next to the name of the project.

## Creating a New Local Branch

### Context

You can create a new local branch referencing any available remote or local branch.

### Procedure

1. In the *Git Pane*, choose  (Add Branch).  
The *Create a New Branch* dialog box appears.
2. From the *Source Branch* dropdown list, select the desired local or remote branch.
3. Enter a name for the new local branch.
4. Choose *OK*.

#### Note

If you have uncommitted changes in your workspace, a dialog box containing the list of conflicting files opens. Choose *Cancel* to abort, or *Reset and Checkout* to remove all uncommitted changes.

The new local branch is created and checked out.

## Deleting a Local Branch

### Context

If there is only one branch available, it cannot be deleted.

### Procedure

1. In the *Git Pane*, choose  (Delete Branch). The *Delete Branch* dialog box appears showing all the branches of the selected repository.
2. Select one or more branches that are not checked out.
3. Choose *Delete*.

The selected branches are deleted.

## 14.7.12.1 Resetting a Local Branch

You can delete all new objects and references that were added to an existing local branch to make it identical to its remote branch.

### Context

When you reset a branch, all unsynced committed changes are removed, and all staged and unstaged files are reverted to their original state in the local copy of the respective remote branch.

### Procedure

1. In the *Git Pane*, choose *Reset*.
2. Select the branch that you want to revert back to.

**i** Note

The branch that is currently checked out is automatically disabled. It cannot be selected. By default, the corresponding remote branch is selected.

3. Choose a *Reset Type*.

**i** Note

If you choose a *Hard* reset, all changes are removed.

4. Choose *OK* to reset the branch.

## 14.7.13 Creating a Remote Branch

You can create a new branch in the remote repository.

### Context

To work on the remote branch you just created, you still must check out the branch by creating a new local branch for this new remote branch.

**i** Note

To create a local branch, you can choose the menu ► *Git* ▶ *Create Local Branch* ▶ or click the plus sign in the *Git* pane.

## Procedure

1. Select your project.
2. From the menu, select **Git > Create Remote Branch**.

## Related Information

[Using Multiple Local Branches \[page 1127\]](#)

### 14.7.14 Viewing Git History

From the Git *History* pane, you can explore the history of committed changes that were made for repositories, folders, and files in a specific project.

Git history is located in a dedicated pane that you can access in one of the following ways:

- Using the main menu: **File > Git > History**
- Using the context menu of the selected Git repository
- Using the dedicated icon (Git History pane) from the right sidebar

Once you have selected a Git repository in the workspace, use the Git *History* pane to:

- View a list of commits (descriptions of the commits) in the central area of the pane. A commit graph on the left gives you a visual representation of the commit history.
- Search commits by:
  - The author of the commit
  - Person who committed the change
  - Commit ID
  - DateThe results of your search are highlighted and you can toggle between them using the arrows on the right of your filter.
- Select a commit in the list to view details of that commit in the area below.
- View a list of all the files packaged in this commit in the *File* column in the area below. From the *Status* column, you can see one of the following statuses of the file:

Table 387:

Status	Description
N	New
D	Deleted
M	Modified
C	Contains conflicts

## 14.7.15 Setting Up Git to Work with Gerrit

Gerrit is a web-based software code review tool for reviewing, approving, or rejecting changes to the source code developed by your colleagues. Gerrit works as an intermediate environment for source control between the local environment and the remote Git repository.

### Procedure

1. From the  *File*  menu, choose *Clone Repository* or *Set Remote*.

If you select *Set Remote*, your project must have been initialized as a local repository with   *File*  *Initialize Local Repository*.

2. Select the *Add configuration for Gerrit* checkbox.

### Results

Anytime you push, the changes will be sent to Gerrit for code review.

 **Note**

Make sure to only select the checkbox if your Git uses Gerrit.

### Related Information

[Cloning Repositories \[page 1119\]](#)

[Initializing a Local Git Repository \[page 1120\]](#)

[Setting a Remote Repository \[page 1120\]](#)

[Fetching Changes from Gerrit \[page 1132\]](#)

## Manual Setup

You can still set up a local Git repository to work with Gerrit even if you did not specify the Gerrit configuration when cloning or setting a remote repository.

### Procedure

1. Right-click your project and choose  [Project Settings](#)  [Git Repository Configuration](#).
2. Choose [Add Entry](#).
3. In the [Key](#) field, enter `gerrit.createchangeid`
4. In the [Value](#) field, enter `true..`
5. Choose [OK](#).

### 14.7.15.1 Fetching Changes from Gerrit

When your repository is set up to work with Gerrit, you work with Git as normal, and review code changes in your Gerrit system. You can also fetch a change from Gerrit and create a local branch from the change, and then collaborate with a colleague using Gerrit before merging the change in Git.

### Procedure

1. From the workspace, select your project.
2. Open the [Git Pane](#).
3. Choose [Fetch from Gerrit](#).
4. Enter the change that you want to download, and choose [OK](#).

Use the ref specification for the change, in the form of `refs/changes/79/2565079/3`. This is available under [Download](#) in the standard Gerrit UI.

The changes are fetched into a new local branch, and the branch is checked out.

#### Note

If you have uncommitted changes in your workspace, a dialog box containing the list of conflicting files opens. Choose [Cancel](#) to abort, or [Reset and Checkout](#) to remove all uncommitted changes.

## Results

When you change or update your code in a repository associated with Gerrit, and then stage, commit and push the changes to Gerrit, a notification confirms that the push operation was successful. A link to the committed change in the Gerrit tool is displayed.

## 14.8 Troubleshooting

Here are some common troubleshooting issues in SAP Web IDE.

### Related Information

[Archive Import Troubleshooting \[page 1133\]](#)

[Git Troubleshooting \[page 1134\]](#)

### 14.8.1 Archive Import Troubleshooting

If you are having trouble importing an archive (.zip file) into SAP Web IDE multi-cloud version, you can try the following possible solutions.

- **UTF-8 Encoding**  
Make sure that the archive files and folder names in your project contain only characters that are encoded with UTF-8 encoding. In other words, these names must not contain any special characters such as a question mark (?) or an ampersand (&). In addition, make sure to use only Latin characters. Non-Latin characters such as Chinese, Japanese, or Hebrew cannot be used.
- **Check File Size**  
The .zip file you want to import must be less than 20 MB.
- **Import .zip Files Only**  
Make sure that the archive you want to import is a .zip file. Archives with other extensions, such as .rar, cannot be imported.

### Multitarget Application (MTA) Project Troubleshooting

- Check the correctness of the mta.yaml file.  
For more information, see [Inside an MTA Descriptor \[page 985\]](#).
- Make sure that the path element of each module in the mta.yaml file points to a folder that actually exists in the archive.

## 14.8.2 Git Troubleshooting

Steps you can take if you have trouble using Git.

The following lists error messages you may receive when using Git, possible causes of the error, and possible solutions.

### Invalid committer

The email listed in the Git repository is not the same as the email assigned to you in Gerrit.

1. Right-click on your project, select  [Project Settings](#)  [Git Repository Configuration](#) and change the email address in the `user.email` field.
2. Commit your changes again, this time by selecting the [Amend Changes](#) checkbox and then selecting [Commit](#). You can then try to push your changes again.

### NON FAST FORWARD

Someone else pushed new changes to the remote repository.

Sync your repository (either [Fetch](#) and then [Rebase](#), or [Pull](#)).

### Prohibited by Gerrit

When you set up your local repository (either by cloning, or by initializing your repository and then setting the remote repository), you did not indicate that you want to work with Gerrit, by selecting [Add configuration for Gerrit](#), when in fact your remote repository does work with Gerrit.

Right-click on your project, select  [Project Settings](#)  [Git Repository Configuration](#) and add an entry with the key `gerrit.createchangeid` and set the value to `true`.

### Cannot upload review

You do not have permission to push changes to Gerrit. Request from your administrator permissions on Gerrit.

### Checkout failed

One of the files listed in the `.gitignore` file is preventing you from checking out a different branch.

1. Open `.gitignore` file (original branch).
2. Remove lines containing files blocking checkout.
3. Click *Discard* for the file if it appears in the staging table in the Git pane.
4. Stage the `.gitignore` file and commit it.
5. Check out your branch.
6. Create `.gitignore` file and save it. (If the file already exists, then make a small change and save it.)
7. In staging table, right-click the files blocking checkout and select *Untrack and Ignore*.
8. Stage, commit and push your changes.
9. Merge your change in Gerrit.
10. Checkout the original branch.
11. Select *Reset*.

## Not authorized

You are not authorized in the Git system. This error may occur simply because the password was incorrect, for one of the following reasons:

- You entered the wrong password.
- The wrong password was cached in the browser. Clear the browser cache of passwords.

## Clone request failed

A clone request may fail for a variety of reasons. Check the error message for the specific reason.

- **Git repository not found:** You entered the wrong clone URL.
- **SAP Web IDE is not configured to trust the security certificate provided by the Git server:** If your Git server uses certificate authentication, you need to upload the server certificate to the SAP Web IDE server. For more information, see [Managing SSL Certificates \[page 966\]](#).

## Fetch request failed (wrong remote URL)

This error occurs after you initialize a local Git repository from a project ([Initialize Local Repository](#)) and then you set a remote repository with the wrong URL. When setting a remote repository, a fetch is automatically performed, and if the wrong URL is entered, the fetch will fail. The error message includes *Git repository not found*.

If you continue and do other actions with the remote repository, these actions will also fail.

To fix the URL, right-click your project and go to  [Project Settings](#)  [Git Repository Configuration](#), and then change the `remote.origin.url` field to the correct URL.

# **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 options is available in SAP Help Portal. 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**

© 2017 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. The information contained herein may be changed without prior notice.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

Please see <http://www.sap.com/corporate-en/legal/copyright/index.epx> for additional trademark information and notices.