PUBLIC
SAP HANA Platform 2.0 SPS 04
Document Version: 1.1 – 2019-10-31

SAP HANA Deployment Infrastructure (HDI) Reference

Content

1 The SAP HANA Deployment Infrastructure Reference

This guide explains how to set up and maintain the SAP HANA Deployment Infrastructure.

The *SAP HANA Deployment Infrastructure Reference* describes the tasks required to set up, maintain, grant access to, and use the SAP HANA Deployment Infrastructure (HDI). It also describes the roles required to provide access to the HDI at the various levels and how the roles fit together to provide a secure deployment infrastructure. The information in the guide is organized as follows:

- The SAP HANA Deployment Infrastructure
  This section provides an introduction to the SAP HANA Deployment Infrastructure, which is a service that enables you to deploy database development artifacts to so-called containers. These artifacts are modeled, staged (uploaded), built, and deployed into SAP HANA.
- SAP HDI administration
  This section describes the roles, personas, and most common tasks involved in the administration of SAP HDI. There is also important information about the security considerations to bear in mind when setting up and running SAP HDI. It is important to note that differences may apply to the various roles, personas, and procedures, depending on the underlying version of the SAP HANA database.
- SAP HDI content development
  This section describes how to develop content with the SAP HANA Deployment Infrastructure either in the context of XS advanced or using the SQL-based HDI application programming interface (API).
  - HDI container SQL API
    This section describes how to use the HDI container SQL API not only to configuring and controlling access to an HDI container but also to develop content in the HDI container. Developers creating content for HDI containers can use the content development API to maintain design-time artifacts in the container's virtual file systems and, on build or deployment, create database objects from them.
  - HDI with XS Advanced
    This section describes how you can make use of HDI in the context of XS advanced, for example with the HDI deployer. XS advanced developers creating content for HDI deployment can either log on to the XS advanced platform with the XS command-line interface (CLI) or use the tools included in the Browser-based SAP Web IDE.
- HDI artifact types and build plug-ins
  This section includes a list of the database artifact types supported by the SAP HANA deployment infrastructure, for example, tables, types, views. It also provides examples of the syntax required to use the artifacts as well as the plug-in configuration required to activate the corresponding catalog objects.

Related Information

SAP HANA Deployment Infrastructure [page 7]
SAP HDI Administration [page 14]
Content Development in SAP HDI [page 112]
SAP HDI Artifact Types and Build Plug-ins Reference [page 210]

2 SAP HANA Deployment Infrastructure

Understand the benefits of using the SAP HANA Deployment Infrastructure (HDI).

The SAP HANA Deployment Infrastructure, or HDI for short, 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 SAP HANA platform database features which describe the target (run-time) state of SAP HANA database artifacts, for example: tables, views, or procedures. These artifacts are modeled, staged (uploaded), built, and deployed into SAP HANA.

! Restriction

The HDI focuses strictly on deployment; HDI does not include any version-control tools, nor does it provide any tools for life-cycle management.

HDI provides its services using a separate database process named `diserver`. On systems where XS advanced is installed, HDI is already enabled; on other systems where XS advanced is not installed, the `diserver` process must usually be enabled by the database administrator before HDI can be used. If required by the usage scenario, other database processes may also need to be started as well.

In the context of XS advanced, the SAP HANA service broker is used to create and drop HDI containers. Outside of XS advanced, it is possible to maintain HDI containers with the SQL APIs. 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. 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 use HDI to deploy application-layer artifacts such as JavaScript programs or OData objects.

The System administrator is responsible for the one-time enabling of HDI, and HDI administrators are responsible for the subsequent regular maintenance tasks,which typically involve using the HDI SQL APIs. For example, an HDI administrator configures HDI, creates HDI container groups, and grants and revokes the access privileges required by HDI container-group administrators. The HDI container-group administrator can drop and create HDI containers, and grant or revoke container (and container-group) access privileges. HDI container administrators grant and revoke container-based access privileges, configure build plug-in libraries and general parameters, and grant and revoke access to an HDI container's schemas. Content developers with the required access permissions can deploy content to the HDI containers using the HDI container APIs, for example, `WRITE` and `MAKE`.

The configuration of HDI containers also 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 design-time database artifact types (for example, sequence, procedure, table, or view) and the corresponding plug in (and version) required to build and deploy the artifacts to the target run-time environment.

- Run-time container name-space rules (`.hdinamespace`)
  An **optional** JSON file containing a list of design-time file suffixes and the naming rules for the corresponding run-time locations. The name-space settings can be configured to **ignore** the folder name or **append** it to the name space. For more information about name-space configuration rules, see *Run-time Name Spaces in SAP HDI* in *Related Information* below.

→ Tip

The rules defined in either the `.hdiconfig` or `.hdinamespace` configuration file apply to the folder in which the configuration file resides and all subfolders unless another configuration file is found that defines new rules for the subfolder in which the additional configuration file is located and all its subfolders.

Related Information

SAP HDI Administration [page 14]
Content Development in SAP HDI [page 112]
The HDI Container Configuration File [page 118]
The HDI Name-Space Configuration File [page 127]
Run-Time Name Spaces in SAP HDI [page 126]
Configuring the HDI Deployer (Developer Guide)

3 The SQL API for SAP HANA Deployment Infrastructure (HDI)

An SQL application programming interface (API) is available to help maintain the SAP HANA Deployment Infrastructure (HDI).

In this topic, you can find information about the following components of the SQL API for HDI:

- SQL APIs for HDI [page 9]
- Parameters for API Procedures [page 10]
- Table Types [page 10]
- Predefined Parameter Tables [page 11]
- Result Sets [page 11]
- Return Codes [page 11]
- Request ID [page 12]
- Messages [page 12]

SQL APIs for HDI

The SAP HANA Deployment Infrastructure (HDI) can be seen as a layer on top of the SAP HANA database. The HDI includes an SQL-based API that is accessible by means of standard SAP HANA SQL connection data. HDI clients use the HDI SQL API to manage containers and their content. Examples of HDI clients include the following: the HDI command-line client, the development environment, and deployment to the XS Advanced run-time platform.

The HDI API is based on a combination of SQL and database procedures. For example, the system-wide container-management API contains procedures for creating and dropping HDI containers. Each HDI container has a container API consisting of container-specific procedures for uploading, listing, and retrieving design-time objects (write, read, list); for deploying uploaded design-time objects (make); and for granting or revoking access to (or from) the container's schemas and API (grant, revoke).

From HANA 2.0 SPS 00, HDI containers can be grouped into container groups, where a corresponding group-level management API restricts the management operations (for example, create, drop, configure, etc.) to the set of HDI containers that are assigned to the specific HDI container group.

The following SQL APIs are provided to help you manage and work with HDI:

SQL APIs for HDI

HDI SQL API	Description
HDI administration API	Used mainly for managing container groups and the administrative access to them. Use of the HDI administration API requires the privileges of an HDI administrator. i Note SYSTEM user privileges are required to create the first HDI administrator, who can then create other HDI administrators, if required. After creation of the first HDI administrator, the SYSTEM user can be deactivated.
HDI container group administration API	Used for managing a set of containers inside a container group and the administrative access to them. To use this API, a user needs to be granted the privileges of an HDI container group administrator by an HDI administrator.
HDI container administration API	Used for configuring a container and controlling the access to it. To use this API, a user needs to be granted the privileges of an HDI container administrator by an HDI container group administrator.
HDI container content development API:	Used for enabling applications to deploy or undeploy HDI artifacts within a container and manage access to the containers. To use this API, a user must be granted the privileges of an HDI container content developer by an HDI container administrator. i Note The HDI container content development API is not described in this HDI administration section.

Parameters of HDI API Procedures

HDI APIs usually expect a number of input and output parameters, where output parameters are always being the last parameters to be passed. Most of the HDI APIs return the following information as output parameters:

- A return code
- A request ID
- A messages table

Table Types

HDI's predefined table types can be found in the `_SYS_DI` schema, and are identified by the prefix "`TT_`. The privileges required for access to these table types are granted to a user by means of the grant APIs available to the HDI administrator, the HDI container-group administrator, or the HDI container administrator. To pass a parameter to an HDI API that expects an HDI table type, a temporary table of that type must be created before calling the API procedure. This temporary table must then be filled with the data that the API expects. For example:

Sample Code

```
CREATE LOCAL TEMPORARY COLUMN TABLE #CONFIG_PARAMETERS LIKE
_SYS_DI.TT_PARAMETERS;
INSERT INTO #CONFIG_PARAMETERS (KEY, VALUE) VALUES ('make.max_parallel_jobs',
'8');
CALL C#DI.CONFIGURE_CONTAINER_PARAMETERS(#CONFIG_PARAMETERS,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #CONFIG_PARAMETERS;
```

Predefined Parameter Tables

HDI's predefined tables can be found in the `_SYS_DI` schema; they can be identified by the prefix "`T_`". The privileges required to access these tables are granted to a user by means of the grant APIs available to the HDI administrator, the HDI container-group administrator, or the HDI container administrator. The predefined tables provide tables with default content that are needed more frequently. For example, when no special parameters need to be given to an HDI API, the `_SYS_DI.T_NO_PARAMETERS` table can be used.

Sample Code

```
CALL C#DI.CONFIGURE_CONTAINER_PARAMETERS(_SYS_DI.T_NO_PARAMETERS,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
```

Result Sets

The SAP HANA DI SQL API calls usually return the following result sets;

1. A result set that contains a result code (for example, 0 or 1) and a request ID (a unique ID for the API call).
2. A result set that includes a table of messages containing information about the execution of the HDI call.

Return Codes

The return code indicates if the API call was successful, or if there were problems during the execution of the call. The meaning of the possible return codes is described in the following table:

SQL API Return Codes for HDI

Return Code	Meaning	Details
0	Success	The call was successful, and the messages did not report any warnings or errors

Return Code	Meaning	Details
`1`	Warning	The messages contain warnings, but no errors.
`-1`	Error	The messages contain errors.
`-2`	Fatal Error	No messages could be logged. This indicates a problem with the database.

Request ID

The request ID is a unique ID generated for each API call. This ID is always the same for messages that originate from the same API call. It can, for example, be used by a container administrator to query the messages that were produced by a certain API call.

Messages

Most APIs return a table of messages with information about the execution of the HDI call. The message, content, and format is explained in the following table:

SQL API Message Tables for HDI

Message Table	Description
REQUEST_ID	The unique ID of the API call that produced this message. This ID is always the same for messages that originate from the same API call.
ROW_ID	An increasing number representing the line number of the message log.
LEVEL	The indentation level of the message (used for better visual representation).
TYPE	The type of message returned; the following list describes the possible message types: • SUMMARY A summary of the API call • HDI: The message from HDI • PLUGIN: The message from a plug-in
LIBRARY_ID	The ID of the library (for messages from a plug-in)
PLUGIN_ID	The ID of the plug-in (for messages from a plug-in)
PATH	The path to the artifact that is being processed
SEVERITY	The severity of the message, for example: INFO, WARNING, ERROR, ...

Message Table	Description
MESSAGE_CODE	A unique code corresponding to the MESSAGE field
MESSAGE	The message text
LOCATION	The Position (for example, "line:column") within the artifact that the message refers to
LOCATION_PATH	The XPath expression within the artifact that the message refers to
TIMESTAMP_UTC	The time stamp indicating when the message was created

Related Information

SAP HANA Deployment Infrastructure [page 7]

4 SAP HDI Administration

An overview of the scope of the tasks required to maintain the SAP HANA Deployment Infrastructure (HDI).

HDI administrators are responsible for enabling and maintaining the HDI. Maintaining the HDI, its HDI containers, and container groups, involves the following high-level tasks:

- Enabling the HDI
  A database administrator with SYSTEM privileges starts the HDI for the first time, creates the necessary administrator users, and assigns the new users the access privileges required to administrate the HDI.
- Maintaining the HDI
  An HDI administrator configures HDI, creates HDI container groups, and grants and revokes the access privileges required by the HDI container-group administrators.
- Maintaining HDI containers groups
  HDI container-group administrators drop and create HDI containers, grant and revoke container (and container-group) access privileges, and import and export containers (for support purposes).
- Maintaining HDI containers
  HDI container administrators grant and revoke container-based access privileges, configure libraries and parameters, and grant and revoke access to an HDI container's schemas.

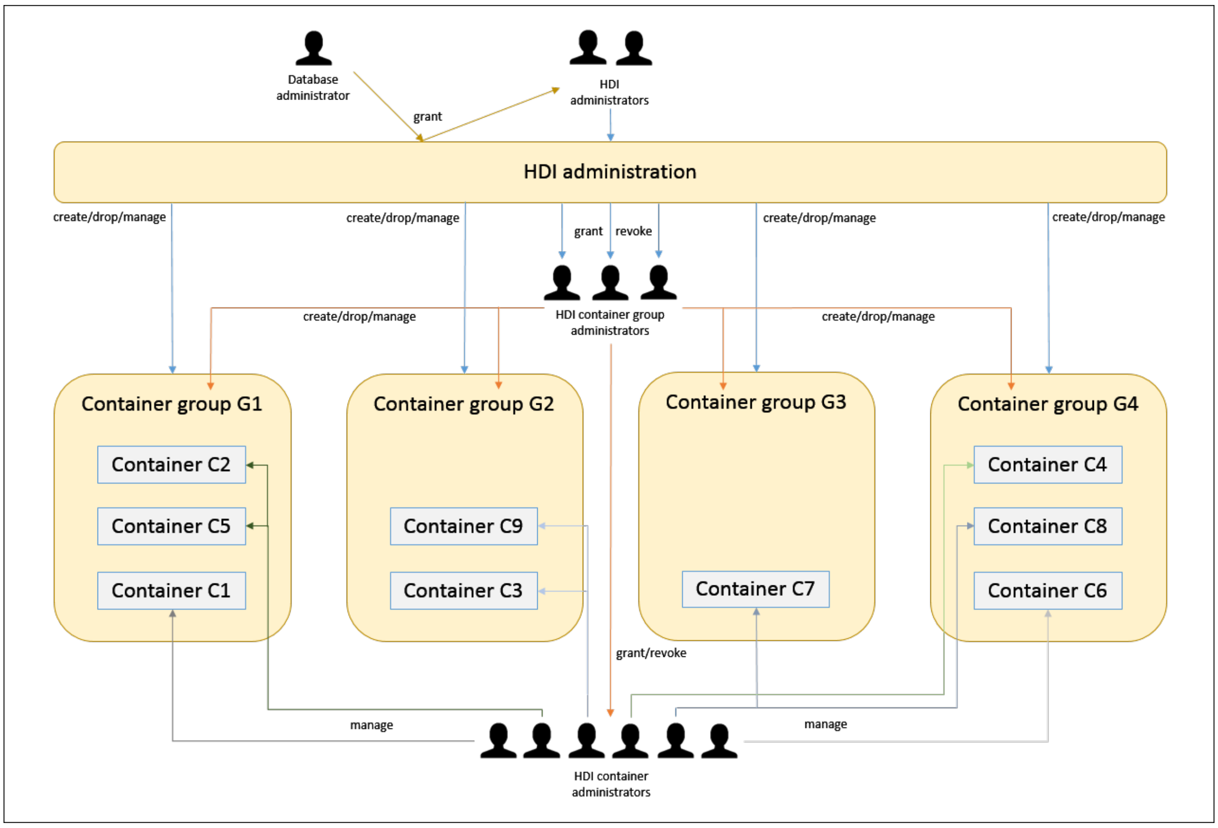
Related Information

SAP HDI Administration in Context [page 15]
SAP HDI Administrator Roles [page 18]
Enabling SAP HDI in the Database [page 40]

4.1 SAP HDI Administration in Context

An overview of the SAP HANA Deployment Infrastructure (HDI) administration process including the administrator users who set up and maintain HDI and its components.

The relationships between the various roles involved in the maintenance of SAP HDI containers and the corresponding scope of each role is shown in the following diagram:



SAP HDI Administration

SAP HDI provides its services using a separate database process named `diserver`. On systems where XS advanced is installed, SAP HDI is already enabled; on other systems where XS advanced is not installed, the `diserver` process must usually be enabled by the database administrator before HDI can be used. If required by the usage scenario, other database process may also need to be started as well.

The database administrator SYSTEM is needed for the first-time enabling of HDI in SAP HANA and for creating an HDI administrator, who then performs the tasks required to set up and maintain other HDI administrators, if required.

i Note

SYSTEM user privileges are required to create the first HDI administrator, who can then create other HDI administrators. After creation of the first HDI administrator, the SYSTEM user can be deactivated.

Created by the database administrator, the HDI administrator is responsible for the setup and overall maintenance of HDI. The role of the HDI administrator includes configuring general HDI parameters, maintaining containers and container groups (for example, by creating and dropping containers and container groups), and managing container-group administrator privileges, for example, by granting and revoking HDI container-group access permissions.

The HDI container-group administrator manages a set of containers in container groups assigned by the HDI administrator. Container-group management tasks include: granting and revoking container (and container-group) **administrator** access privileges; granting and revoking container **user** access privileges (for example, for temporary support purposes); maintaining containers and container groups.

→ Tip

The APIs of a container group “G” are in the `_SYS_DI#G` schema.

The HDI container administrator manages one or more containers assigned by the container-group administrator. The role of the container-manager focuses primarily on configuring and controlling access to the HDI containers used to store the database objects deployed by the SAP HANA Deployment Infrastructure deploy service and repairing any problems that occur with run-time objects in the assigned HDI containers. An HDI container administrator can manage one or more containers in one HDI container group or multiple containers distributed across multiple container groups.

→ Tip

The APIs of a container “C” are in the `C#DI` schema.

SAP HDI 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), which is an isolated environment used to store design-time files, and a run-time container (RTC), which is used to store deployed objects built according to the specification stored in the corresponding design-time artifacts.

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.

The configuration of HDI containers also 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 plug-in (and version).
- Run-time container name-space rules (`.hdinamespace`)
  A JSON file containing a list of design-time file suffixes and the naming rules for the corresponding run-time 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.

SAP HDI Container Groups

SAP HDI container groups are logical collections of the HDI containers used to store the objects deployed by the SAP HANA Deployment Infrastructure deploy service. After creation, an HDI container group can be assigned to a dedicated container-group administrator, who must be granted the privileges required to perform the typical tasks associated with the administration of container group, for example: granting and revoking container (and container-group) access privileges; and maintaining container groups (and the containers assigned to the groups).

> → Tip
>
> Container groups are intended to make life easier when multiple administrators require access to containers from different contexts, for example, XS advanced, ABAP, or other development groups working in native SAP HANA contexts.

The HDI administrator can create container groups for a target audience whose container-related requirements are unique or where there is an obvious benefit for a logical separation, for example, between ABAP and applications running in the XS advanced model run-time environment.

Related Information

Enabling SAP HDI in the Database [page 40]
Maintaining the SAP HDI [page 46]
Maintaining SAP HDI Container Groups [page 77]
Maintaining SAP HDI Containers [page 90]
SAP HDI Artifact Types and Build Plug-ins Reference [page 210]

4.2 SAP HDI Administrator Roles

The administration of SAP HANA Deployment Infrastructure (HDI) involves a number of tasks that must be performed by different administrator roles.

The following table describes the scope of the various SAP HDI administrator roles and lists the most common tasks the administrators are expected to perform.

SAP HDI Administrator Scope, Roles, and Tasks

SAP HDI Role	Description	Common Tasks
Database administrator	The database administrator SYSTEM is needed for initially enabling SAP HDI and for creating an HDI administrator. **Note** SYSTEM user privileges are required to create the first HDI administrator, who can then create other HDI administrators. After creation of the first HDI administrator, the SYSTEM user can be deactivated.	Enable SAP HDI Create an HDI administrator Grant and revoke HDI administrator privileges
HDI administrator	Configures general SAP HDI parameters, maintains containers and container groups, and manages container group administrator privileges.	Configure SAP HDI Create and drop containers and container groups Grant and revoke required access privileges Maintain containers and container groups Move containers between container groups
HDI container-group administrator	Manages the container groups assigned by the SAP HDI administrator. The APIs of a container group “G” are in the `_SYS_DI#G` schema.	Grant and revoke container (and container-group) **administrator** access privileges Import and export containers (for support purposes) Grant and revoke container **user** access privileges (for support purposes) Maintain containers and container groups

SAP HDI Role	Description	Common Tasks
HDI container administrator	Configures and controls access to a container and manages run-time objects in the assigned containers. The APIs of a container "C" are in the `C#DI` schema.	Grant and revoke container **administrator** access privileges Configure libraries and parameters Grant and revoke roles from schemas to users Grant and revoke user access to container schemas Cancel an asynchronous make operation

Related Information

SAP HANA Deployment Infrastructure (Administration Guide) [page 7]

4.3 SAP HDI Security

An overview of the tools used to configure and ensure security in the SAP HANA Deployment Infrastructure (HDI).

This section of the *The SAP HANA Deployment Infrastructure Reference* describes the aspects of security that you need to consider when setting up and using the SAP HDI infrastructure. The information covers the following areas from the perspective of security:

- Technical System Landscape of SAP HDI
  An overview of the architecture of SAP HDI and which protocols for communication between the various high-level components
- SAP HDI Users
  An overview of the predefined users which SAP HDI relies on and a description of each user's scope
- SAP HDI Database Roles
  An overview of the predefined database roles that are required to operate the SAP HANA Deployment Infrastructure (HDI) along with a description of each role
- Data Storage Security in SAP HDI
  An overview of the mechanisms used to secure and protect critical data managed by the SAP HDI
- Network and Communication Security with SAP HDI
  An overview of the security mechanisms applied to networking and communication in the context of SAP HDI
- Security-relevant Logging and Tracing in SAP HDI
  An overview of the auditing process, which enables you to trace the different kinds of operations performed by HDI users in the context of SAP HDI
- Data Protection and Privacy in SAP HDI
  An overview of SAP HDI's adherence to (and compliance with) data protection and privacy regulations

- SAP HDI Security in the Context of XS Advanced
  An overview of the security considerations to bear in mind when enabling SAP HDI for use in the context of SAP HANA extended application services, advanced model

> i Note
>
> SAP HDI is a service layer on top of the SAP HANA platform, so information about other components in other sections of the SAP HANA Security Guide might also apply.

Related Information

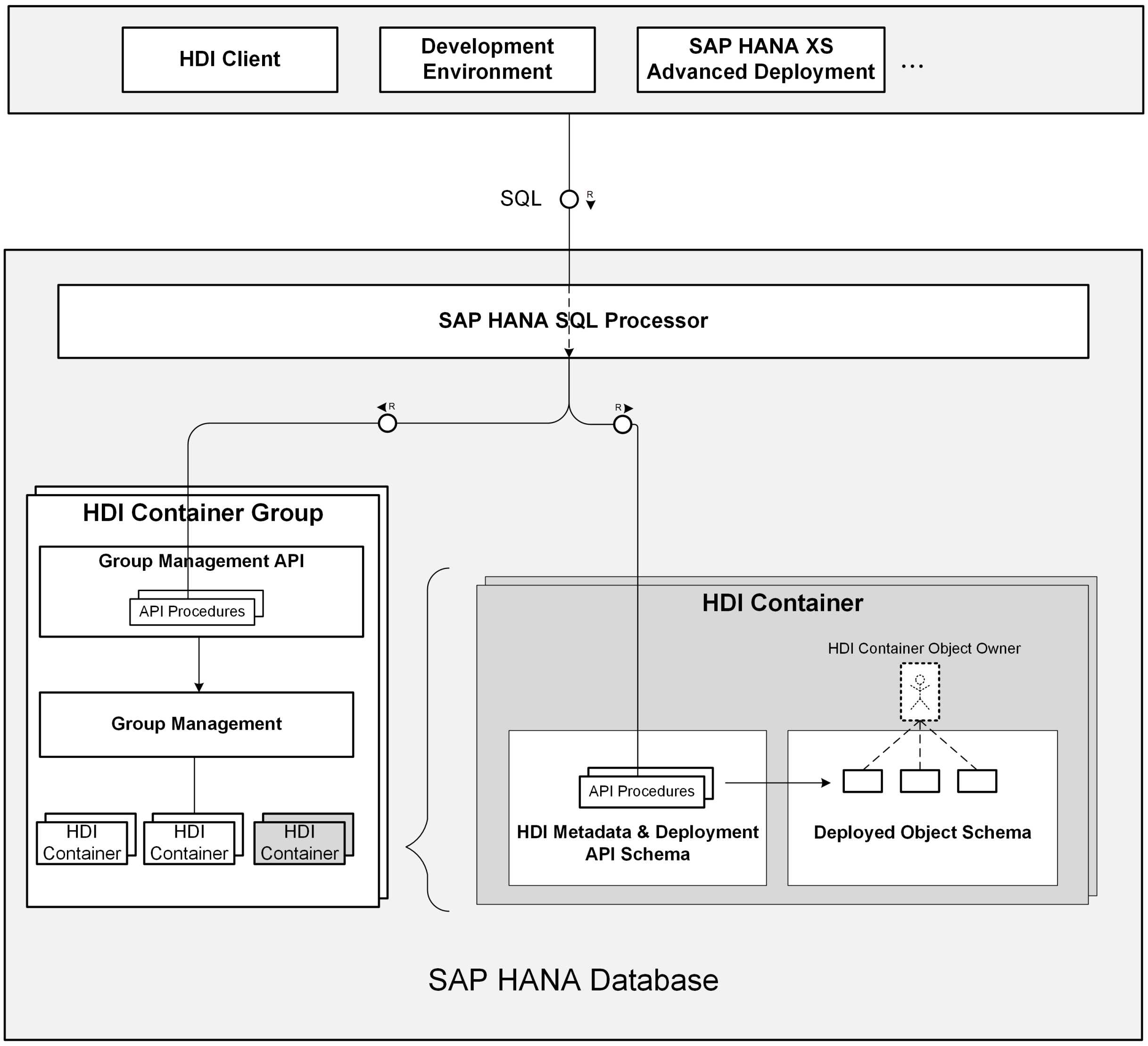
Technical System Landscape of SAP HDI [page 20]
SAP HDI Users [page 22]
Data Storage Security in SAP HDI [page 25]
Network and Communication Security with SAP HDI [page 25]
Security-relevant Logging and Tracing in SAP HDI [page 26]
Data Protection and Privacy in SAP HDI [page 30]
SAP HDI Security in the Context of XS Advanced [page 31]

4.3.1 Technical System Landscape of SAP HDI

An overview of the architecture of SAP HDI.

HDI clients such as the client libraries for Java and JavaScript use the HDI SQL API to manage containers and their content. So, too, does the HDI deployer tool as well as the SAP HANA extended application services, advanced model (XS advanced), which also uses HDI containers as a way of implementing database service instances.

The HDI API is based on SQL and database procedures. For example, the application programming interface (API) for system-wide container management includes procedures for creating and dropping containers. Each container has its own container API consisting of container-specific procedures for uploading (`WRITE`), listing (`READ`), and retrieving (`LIST`) design-time objects, for deploying uploaded design time objects (`MAKE`), and for granting/revoking access to/from the container's schemas and API (grant, revoke). Starting with HANA 2 SPS00, containers can be grouped via container groups where a corresponding group-level management API restricts the management operations (create/drop, configure, etc.) to the set of containers which are contained in the specific group.



Users and Clients

Users and applications interact with SAP HDI by means of one of the available clients, which use the SAP HDI's SQL application-programming interface to manage HDI containers and the corresponding content.

- HDI client
  HDI provides client libraries for Node.js (`@sap/hdi`) and Java (`sap-java-hdi`) which include a selection of methods that can be used to connect to the HDI and access HDI functionality.
- The development environment
  The HDI's API is based on a library of SQL procedures, which can be used (among other things) to create and drop containers; upload design-time objects to the virtual file system of the HDI container; deploy uploaded objects (make) to the container run time; and grant (or revoke) the permissions required to access HDI containers.
- Deployment to the SAP HANA XS advanced platform

The Node.js packages `@sap/hdi-deploy` and `@sap/hdi-dynamic-deploy` are based on the HDI's SQL API and enable the deployment of database content to HDI containers.

Related Information

SAP HDI Security [page 19]

4.3.2 SAP HDI Users

A list of the predefined users which SAP HDI relies on and a description of their respective role.

HDI uses several container-specific technical users to separate the different container-related tasks such as creating the schemas, triggering deployment, and creating database objects for XS advanced applications.

Most of the server agents require a data store in the SAP HANA database and therefore need secure access to schemas. For this reason, a dedicated technical SAP HANA user is generated for each such schema, and the credentials of the technical SAP HANA user are passed to the server agent. As the management of technical users is performed at the infrastructure level, end users typically do not interact with these users.

The XS advanced application's database objects are created during application deployment with a special container-specific technical system user, the container object owner (the container's "#OO" user). Unless explicitly granted such a privilege, this user has no access to the HDI container's metadata schema and especially not to the HDI API procedures. In this way, it is possible to prevent an application procedure with definer-mode from security acquiring unwanted access to metadata and the deployment API. Run-time users need the corresponding access privileges only on the container schema. Note that XS Advanced applications connect to HDI containers with container-specific, technical, run-time users.

The technical users listed in the following table are used to connect to the SAP HANA database with a specific set of conditions.

User ID	Service	Type	Description
`HDI_ADMIN_USER`	SAP HANA Broker	Technical database user	Owns the SAP HANA schema of the SAP HANA Service Broker; only used in the context of XS advanced
`HDI_BROKER_CONTROLLER`	SAP HANA Broker	Technical database user	Has the authorization required to access the service-broker API of the SAP HANA Service Broker; only used in the context of XS advanced

User ID	Service	Type	Description
`_SYS_DI`	HDI	Technical database user	Owns all HDI SQL-based APIs, for example all API procedures in the `_SYS_DI` schema and API procedures in containers
`_SYS_DI_*_CATALOG`	HDI	Technical database user	Technical users used by the HDI to access database system catalog tables and views
`_SYS_DI_SU`	HDI	Technical database user	Technical superuser of the HDI created at installation time
`_SYS_DI_TO`	HDI	Technical database user	Owns transaction and connections of all internal HDI transactions

Technical Users for HDI Schema-Based Containers

The deployment of database objects with SAP HANA Deployment Infrastructure (HDI) is based on a container model where each container corresponds roughly to a database schema. Each schema, and the database objects deployed into the schema, are owned by a dedicated technical database user.

For every container deployed, a new technical database user and schema with the same name as the container are created. Additional schemas and technical users required for metadata and deployment APIs are also created.

For example, for a container named `S`, HDI creates the following users:

- `S`:
  The user who is the owner of the container schema `S`
- User `S#DI`:
  In SAP HANA 2.0, the user who is the owner of the schema `S#DI` containing metadata and deployment APIs
- User `S#OO`:
  The user who is the owner of database objects in schema `S`
- Users `_DI#S#METADATA_COM_SAP_HANA_DI_<metadata>`:
  In SAP HANA 1.0 only, the users who are the owners of schemas containing build plug-in metadata

These technical database users are used internally by HDI only. As a general rule, these users are created as restricted database users who do not have any privileges by default (not even the role `PUBLIC`), and cannot be used to log on to the database. The only exception to this rule concerns user `S#OO` who, from SAP HANA 2.0 SPS 02 revision 20, is granted the role `PUBLIC` by default.

For more information, see *Maintaining HDI Containers* in the *Related Information*.

Related Information

Predefined Database Roles for SAP HDI [page 24]

Maintaining SAP HDI Containers [page 90]
SAP HDI Security in the Context of XS Advanced [page 31]

4.3.3 Predefined Database Roles for SAP HDI

Several predefined database roles are necessary for operating the SAP HANA Deployment Infrastructure (HDI).

> i Note
>
> The following roles are SQL-based roles that are available in the catalog of the SAP HANA database.

Role	Description
`_SYS_DI_OO_DEFAULTS`	This role contains the set of default privileges that are granted to all HDI container object-owner users (`<container>`#OO users). SAP HANA Deployment Infrastructure (HDI) uses this role internally to grant default privileges instead of using the `PUBLIC` role. It contains only privileges to `SYS` views where additional security checks apply. The role contains `SELECT` privileges on the views: `SYS.DUMMY`, `SYS.PROCEDURES`, `SYS.PROCEDURE_PARAMETERS`, `SYS.TABLES`, `SYS.TABLE_COLUMNS`. This role is not intended to be granted to database users. i Note Do **not** extend this role in a production system.
`_SYS_DI#<container_group>._SYS_DI_OO_DEFAULTS`	This role contains the set of default privileges that are granted to all HDI container object-owner users (`<container>`#OO users) who are part of the container group `<container_group>`. The role does not contain any privileges by default. Privileges granted to a container group-specific role will automatically be available to all HDI container object-owner users of the specified container group. i Note This role is not intended to be granted to database users.

Related Information

SAP HDI Security [page 19]
SAP HDI Users [page 22]

4.3.4 Data Storage Security in SAP HDI

Security mechanisms are applied to protect critical data managed by the SAP HDI.

Since SAP HDI is a service layer on top of the SAP HANA database, it adheres to all the rules and regulations that apply to the SAP HANA database itself and described in detail in the *Data Storage Security in SAP HANA* section of the *SAP HANA Security Guide*.

Database objects (tables, views, procedures, and so on) have an owner: the user who created the object. When the owner of a database object is deleted, all objects owned by the deleted user are removed from the database, too. In addition, if application objects are created by end-users, the objects are deleted when the end-user is deleted, for example when the employee leaves the organization. HDI ensures that during deployment all database objects are created by a container-specific **technical user**, which is never deleted as long as the container exists.

Related Information

SAP HDI Security [page 19]
SAP HANA Security Guide

4.3.5 Network and Communication Security with SAP HDI

An overview of the security mechanisms applied to networking and communication in the context of SAP HDI.

Security mechanisms are applied to protect the communication paths used by the SAP HANA Deployment Infrastructure (HDI). SAP provides network topology recommendations that enable administrators to restrict access at the network level.

The JDBC connection to the SAP HANA database is not encrypted by default. To activate JDBC TLS/SSL, it is necessary to configure custom SSL certificates as described in *SAP HDI Certificate Management* below.

SAP HDI Certificate Management

The HDI administrator is responsible for the certificate management in the SAP HANA JDBC area. The SAP HDI does not encrypt the JDBC connections to SAP HANA out of the box. Therefore, custom certificates must be configured, which involves the following high-level steps:

- Deploy the custom certificate in the SAP HANA database in order to provide the certificate to all index server instances
- Publish of the certificate to the SAP HDI server components
- Enable the JDBC SSL in the platform settings and restart the system

Related Information

SAP HDI Security [page 19]

SAP HDI Security in the Context of XS Advanced [page 31]

4.3.6 Security-relevant Logging and Tracing in SAP HDI

The auditing process enables you to trace who has performed which kinds of operations in the context of SAP HDI.

System audit logs can help detect undesired modifications that have security implications, for example, changes to user authorization, creation and deletion of database objects, or changes to the system configuration. Audit logs can also help uncover attempts to breach system security or reveal security holes, for example, where a specific user has privileges that are not necessary. Auditing can also help protect the system owner against the threat of security violations and data misuse or may serve to meet security standards of a company. For SAP HDI auditing, audit-log events can be written to SAP HANA's central auditing service.

Activating Auditing in SAP HANA

To use the auditing feature in productive SAP HANA systems, it must first be activated, for example, by adding an entry to a system configuration file or by executing the following SQL command with the system privilege "`AUDIT ADMIN`":

```
alter system alter configuration ('global.ini', 'SYSTEM') set ('auditing
configuration', 'global_auditing_state') = 'true' with reconfigure;
```

You define an audit policy by declaring the actions or operations that you want to trigger an entry in the audit report as well as the conditions under which the actions must be performed. For example, the following SQL command shows how to create an audit policy that checks on all "`create user`" or "`drop user`" operations, writes an entry to the audit log with the severity level "`critical`":

```
create audit policy TEST_AUDIT_POLICY
auditing all
create user, drop user
level critical;
```

In an SAP HANA SQL audit policy, `auditing all` refers to the status of the audited action or operation, for example, "successful", "unsuccessful", or "all". The `level` specifies the severity status assigned to the entries logged by the audit policy for the corresponding actions, in this example, `create user` and `drop user`. You create a new policy for each groups of actions to which you want to assign a different severity level, for example: `info`, `warning`, `critical`, or `alert`.

After creation, an audit policy must be activated, for example, with the following SQL command:

```
alter audit policy TEST_AUDIT_POLICY enable;
```

For more information audit policies, see the `CREATE AUDIT POLICY` command in the *SAP HANA SQL and System Views Reference for SAP HANA Platform*.

Auditing in SAP HDI

The standard capabilities provided by the SAP HANA audit policies are sufficient for auditing actions in the context of SAP HDI, too. The functionality of SAP HDI is provided by means of an SQL interface and implemented either as built-in procedures or stored procedures. Using the existing auditing features, it is possible to audit all calls to SAP HDI.

Auditing SAP HDI Procedures

In SAP HDI, general procedures enable the execution of high-level operations, for example, creating and dropping containers, granting and revoking privileges and roles, or configuring SAP HDI itself. The following code example shows how to use SQL to create and enable an audit policy that audits these general SAP HDI procedures:The standard capabilities provided by the SAP HANA auditing mechanism are sufficient for auditing in SAP HDI, too. The functionality of SAP HDI is provided by means of an SQL interface and implemented either as built-in procedures or stored procedures. Using the existing auditing features, it is possible to audit all calls to SAP HDI.

Sample Code

Creating a General Audit Policy

```
create audit policy TEST_AUDIT_POLICY_HDI
  auditing all
  execute on
    _SYS_DI.CANCEL,
    _SYS_DI.CONFIGURE_CONTAINER,
    _SYS_DI.CONFIGURE_CONTAINER_PARAMETERS,
    _SYS_DI.CONFIGURE_DI,
    _SYS_DI.CONFIGURE_DI_PARAMETERS,
    _SYS_DI.CONFIGURE_LIBRARIES,
    _SYS_DI.CREATE_CONTAINER,
    _SYS_DI.CREATE_CONTAINER_GROUP,
    _SYS_DI.DISABLE_CONTAINER_USER,
    _SYS_DI.DROP_CONTAINER,
    _SYS_DI.DROP_CONTAINER_GROUP,
    _SYS_DI.ENABLE_CONTAINER_USER,
    _SYS_DI.EXPORT_CONTAINER_FOR_COPY,
    _SYS_DI.EXPORT_CONTAINER_FOR_SUPPORT,
    _SYS_DI.GRANT_CONTAINER_API_PRIVILEGES,
    _SYS_DI.GRANT_CONTAINER_API_PRIVILEGES_WITH_GRANT_OPTION,
    _SYS_DI.GRANT_CONTAINER_GROUP_API_PRIVILEGES,
    _SYS_DI.GRANT_CONTAINER_GROUP_API_PRIVILEGES_WITH_GRANT_OPTION,
    _SYS_DI.GRANT_CONTAINER_SCHEMA_PRIVILEGES,
    _SYS_DI.GRANT_CONTAINER_SCHEMA_ROLES,
    _SYS_DI.GRANT_CONTAINER_SUPPORT_PRIVILEGE,
    _SYS_DI.IMPORT_CONTAINER_FOR_COPY,
    _SYS_DI.IMPORT_CONTAINER_FOR_SUPPORT,
    _SYS_DI.LIST_CONFIGURED_LIBRARIES,
    _SYS_DI.LIST_LIBRARIES,
    _SYS_DI.MOVE_CONTAINER_TO_GROUP,
    _SYS_DI.REVOKE_CONTAINER_API_PRIVILEGES,
    _SYS_DI.REVOKE_CONTAINER_GROUP_API_PRIVILEGES,
    _SYS_DI.REVOKE_CONTAINER_SCHEMA_PRIVILEGES,
    _SYS_DI.REVOKE_CONTAINER_SCHEMA_ROLES,
    _SYS_DI.REVOKE_CONTAINER_SUPPORT_PRIVILEGE
  level info;
alter audit policy TEST_AUDIT_POLICY_HDI enable;
```

Auditing Container-Specific SAP HDI Procedures

Developers using SAP HDI typically work within a single container, calling container-specific SAP HDI procedures, for example: writing, reading, and deleting files and folders, or deploying and undeploying these artifacts. The following SQL code shows how to create and enable an audit policy that audits container-specific SAP HDI procedures for a container, named “C”, for example, `C#DI.CANCEL`:

→ Tip

To apply the following example of an auditing scenario elsewhere, replace “C” with the name of your SAP HDI container.

Sample Code

Auditing SAP HDI Procedures in HDI Container “C”

```
create audit policy TEST_AUDIT_POLICY_CONTAINER
  auditing all
  execute on
    C#DI.CANCEL,
    C#DI.CONFIGURE_CONTAINER,
    C#DI.CONFIGURE_CONTAINER_PARAMETERS,
    C#DI.CONFIGURE_LIBRARIES,
    C#DI.DELETE,
    C#DI.EXPORT_CONTAINER_FOR_COPY,
    C#DI.GET_DEPENDENCIES,
    C#DI.GET_MAKE_GROUPS,
    C#DI.GRANT_CONTAINER_API_PRIVILEGES,
    C#DI.GRANT_CONTAINER_API_PRIVILEGES_WITH_GRANT_OPTION,
    C#DI.GRANT_CONTAINER_SCHEMA_PRIVILEGES,
    C#DI.GRANT_CONTAINER_SCHEMA_ROLES,
    C#DI.IMPORT_CONTAINER_FOR_COPY,
    C#DI.LIST,
    C#DI.LIST_CONFIGURED_LIBRARIES,
    C#DI.LIST_DEPLOYED,
    C#DI.LOCK,
    C#DI.MAKE,
    C#DI.MAKE_ASYNC,
    C#DI.READ,
    C#DI.READ_DEPLOYED,
    C#DI.REVOKE_CONTAINER_API_PRIVILEGES,
    C#DI.REVOKE_CONTAINER_SCHEMA_PRIVILEGES,
    C#DI.REVOKE_CONTAINER_SCHEMA_ROLES,
    C#DI.STATUS, <CONTAINER_NAME>#DI.WRITE
  level info;
alter audit policy TEST_AUDIT_POLICY_CONTAINER enable;
```

Auditing SAP HDI Procedures in All Containers

It is also possible to audit the SAP HDI procedures of **all** containers. Since all container-specific stored procedures call built-in procedures within the "`SYS`" schema, it is sufficient to audit these central built-in procedures if you want to audit the procedure calls of **all** containers. The following example shows how to use SQL to create and enable an audit policy that audits the SAP HDI procedures of **all** HDI containers:

Sample Code

Auditing SAP HDI Procedures in All HDI Containers

```
create audit policy TEST_AUDIT_POLICY_ALL_CONTAINERS
  auditing all
  execute on
    SYS.DI_CANCEL_DEV,
    SYS.DI_CONFIGURE_CONTAINER_DEV,
    SYS.DI_CONFIGURE_CONTAINER_PARAMETERS_DEV,
    SYS.DI_CONFIGURE_LIBRARIES_DEV,
    SYS.DI_DELETE_DEV,
    SYS.DI_EXPORT_CONTAINER_FOR_COPY_DEV,
    SYS.DI_GET_DEPENDENCIES_DEV, SYS.DI_GET_MAKE_GROUPS_DEV,
    SYS.DI_GRANT_CONTAINER_API_PRIVILEGES_DEV,
    SYS.DI_GRANT_CONTAINER_SCHEMA_PRIVILEGES_DEV,
    SYS.DI_GRANT_CONTAINER_SCHEMA_ROLES_DEV,
    SYS.DI_IMPORT_CONTAINER_FOR_COPY_DEV,
    SYS.DI_LIST_DEV,
    SYS.DI_LIST_CONFIGURED_LIBRARIES_DEV,
    SYS.DI_LIST_DEPLOYED_DEV,
    SYS.DI_LOCK_DEV,
    SYS.DI_MAKE_DEV,
    SYS.DI_MAKE_ASYNC_DEV,
    SYS.DI_READ_DEV,
    SYS.DI_READ_DEPLOYED_DEV,
    SYS.DI_REVOKE_CONTAINER_API_PRIVILEGES_DEV,
    SYS.DI_GRANT_CONTAINER_API_PRIVILEGES_WITH_GRANT_OPTION_DEV,
    SYS.DI_REVOKE_CONTAINER_SCHEMA_PRIVILEGES_DEV,
    SYS.DI_REVOKE_CONTAINER_SCHEMA_ROLES_DEV,
    SYS.DI_STATUS_DEV,
    SYS.DI_WRITE_DEV
  level info;
alter audit policy TEST_AUDIT_POLICY_ALL_CONTAINERS enable;
```

Auditing All SAP HDI Database Operations

The standard capabilities provided by the SAP HANA auditing mechanism are sufficient forIn rare cases, it can be helpful to audit all SAP HDI database operations, for example, creating and dropping tables and users, or inserting data into tables. As all those database operations are internally executed by a dedicated SAP HDI transaction owner named "`_SYS_DI_TO`", it is sufficient to limit auditing to this user. For example, the following SQL command shows how to create and enable an audit policy that audits all SAP HDI database operations for the SAP HDI transaction owner `_SYS_DI_TO`:

Sample Code

Auditing All Database Operations in SAP HDI

```
create audit policy TEST_AUDIT_POLICY_ALL_ACTIONS
  auditing all
  actions
  for _SYS_DI_TO
  level info;
alter audit policy TEST_AUDIT_POLICY_ALL_ACTIONS enable;
```

Related Information

SAP HDI Security [page 19]

4.3.7 Data Protection and Privacy in SAP HDI

An overview of compliance with data protection and privacy regulations in the context of SAP HDI.

The SAP HDI server (`diserver`) provides clients with a selection of tools and functions that conform to the legal requirements for the protection and privacy of data included in deployed artifacts, as well as the server infrastructure itself.

This section and any other sections in this Security Guide do not give any advice on whether these features and functions are the best method to support company, industry, regional, or country-specific requirements. Furthermore, this guide does not give any advice or recommendations regarding additional features that would be required in specific IT environments; decisions related to data protection must be made on a case-by-case basis, taking into consideration the given system landscape and the applicable legal requirements.

i Note

In the majority of cases, compliance with data privacy laws is not a product feature. SAP software supports data privacy by providing security features and specific functions relevant to data protection, such as functions for the simplified blocking and deletion of personal data. SAP does not provide legal advice in any form. The definitions and other terms used in this guide are not taken from any given legal source.

Related Information

SAP HDI Security [page 19]

4.3.8 SAP HDI Security in the Context of XS Advanced

An overview of the security considerations to bear in mind when enabling SAP HDI for use in the context of SAP HANA extended application services, advanced model

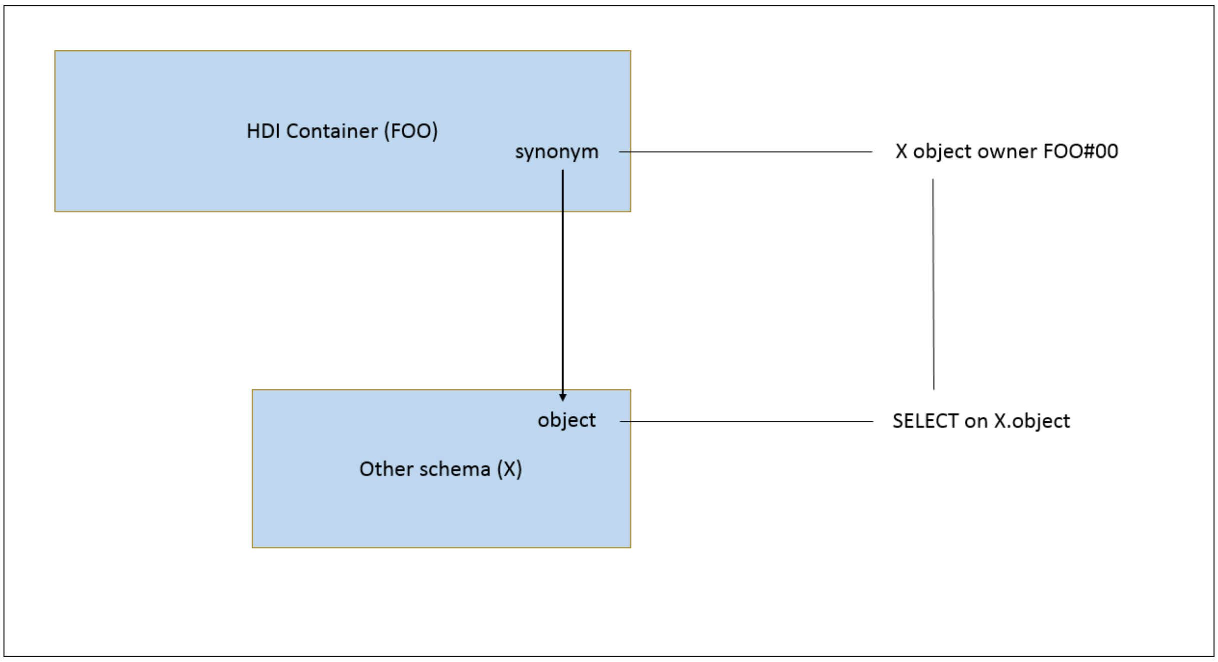
In the context of XS advanced, SAP HDI provides a service that enables you to deploy database development artifacts such as tables, views, and procedures to HDI containers. The deployment process populates the database run-time with the corresponding catalog objects. In addition to database artifacts, HDI also enables you to import and export table content such as business configuration data and translatable texts. This section includes security-related information about the following XS advanced components:

- The SAP HDI Deployer [page 31]
- The SAP HDI SQL API [page 33]
- SAP HDI Containers [page 34]
- SAP HDI Users in XS Advanced [page 36]
- SAP HDI Roles in XS Advanced [page 37]
- SAP HDI Data Storage in XS Advanced [page 39]

The SAP HDI Deployer

For XS advanced and Cloud Foundry environments, the SAP HANA Deployment Infrastructure (HDI) includes a so-called "Deployer", which is a Node application (`@sap/hdi-deploy`) that is included in a Multi-Target Application (MTA) for the purpose of deploying an XS advanced application's HDI database-related design-time artifacts to the application's corresponding HDI containers. The Node.js application `@sap/hdi-dynamic-deploy` enable the deployment of database artifacts to HDI containers that are created dynamically, too, for example, by the XS advanced Instance Manager.

By default, an HDI container is assigned very few database privileges. For example, the object owner (“`#OO`” user) is only assigned the `CREATE ANY` privilege on the container's run-time schema (schema “FOO” for an HDI container “FOO”). To access database objects inside other database schemata or other HDI containers, and to be able to deploy synonyms into the HDI container which point to objects outside the container, the object owner needs additional privileges. For example, for an object “`object`” in schema “X”, the `SELECT` privilege on “`X.object`” :



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>
    }
  }
}
```

In the code example above, the top-level key "`granting-service`" defines the grantors; the names of the bound services which **grant** the privileges required by the various users. The next level defines the users to whom the privileges are granted, these are the grantees. For example, "`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 Multi Target Application. The third level defines the set of privileges to grant using a structure that is similar to the format used in a `.hdbrole` role-definition file.

Note

For backwards compatibility, the suffix `.hdbsynonymgrantor` is also supported.

On startup, the HDI Deployer looks for files with the `.hdbgrants` suffix and processes them in the following way: For each grantor specified in the `.hdbgrants` file, the HDI Deployer looks up a bound service with the specified name (subject to service replacements), connects to the database with the service's credentials, and grants the specified privileges to the grantees.

Code Syntax

Example `cfg/SYS-access.hdbgrants` File with Some Privileges

```
{
  "SYS-access": {
    "object_owner": {
      "object_privileges":[
        {
          "schema": "SYS",
          "name": "VIEWS",
          "privileges": ["SELECT"]
        },
        {
          "schema": "SYS",
          "name": "TABLES",
          "privileges": ["SELECT"]
        }
      ]
    },
    "application_user": {
      "object_privileges":[
        {
          "schema": "SYS",
          "name": "VIEWS",
          "privileges": ["SELECT"]
        },
        {
          "schema": "SYS",
          "name": "TABLES",
          "privileges": ["SELECT"]
        }
      ]
    }
  }
}
```

The SAP HDI SQL API

In addition to the stand-alone deployer application, SAP HDI also provides an SQL application programming interface (API) that enables users and XS advanced applications to create database objects at run time by generating design-time artifacts within the application and deploying them into their container using HDI tools. The HDI user `_SYS_DI` is the owner of all HDI SQL-based APIs, for example, all API procedures in the `_SYS_DI` schema and API procedures in HDI containers.

Client Libraries for XS Advanced Applications

The client libraries for Node.js (`@sap/hdi`) and Java (`sap-java-hdi`) provide access to SAP HDI functionality by means of the SAP HDI's SQL APIs. Assuming the user `hdiUser` has the privileges required to access `_SYS_DI`, the client libraries `@sap/hdi` and `sap-java-hdi` can be used to connect to the SAP HANA database, configure HDI, create and drop HDI containers, configure container build plug-in libraries, and so on.

SAP HDI Containers

The SAP HANA Deployment Infrastructure (HDI) uses so-called containers to store design-time artifacts and the corresponding deployed run-time (catalog) objects. The HDI makes a strict separation between design-time and run-time objects by introducing the following, distinct container types:

- **Design-time** container (DTC)
  Provides an isolated environment for the storage of design-time files by means of a "work" and "deployed" file systems.

  > ! Restriction
  >
  > For maximum isolation and increased security, content stored in a design-time container is owned by a dedicated technical user.

- **Run-time** container (RTC)
  Stores the deployed objects built according to the specification stored in the corresponding design-time artifacts. The creation of a run-time container results in a new technical database user and a schema with the same name as the one chosen for the name of the run-time container itself. Access to content in the run-time container is controlled by database privileges, for example, by granting SELECT or EXECUTE privileges.

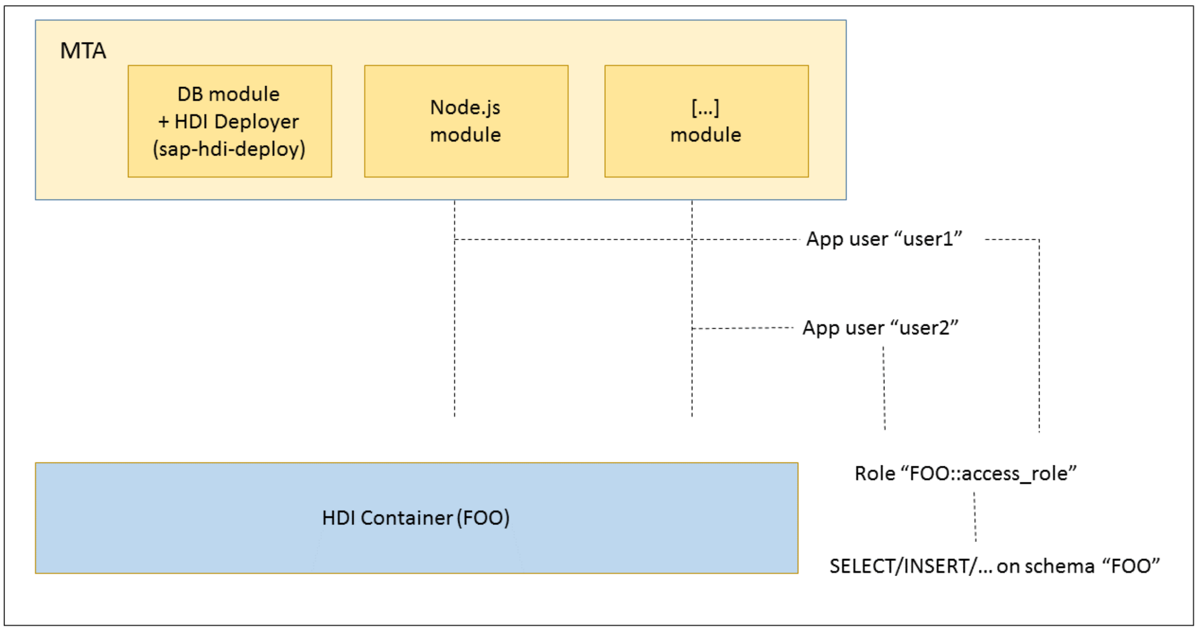
The Default Access Role for HDI Containers

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.



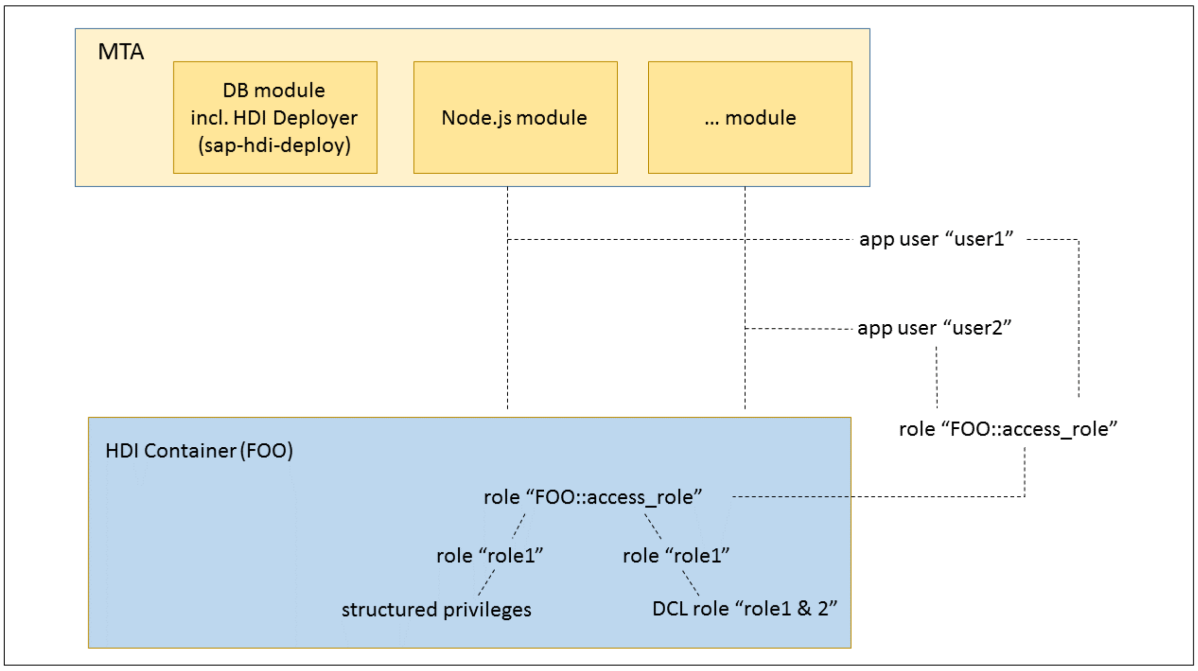
To assign roles from the HDI content to the application-binding-specific users (the users specified in `"user":`), the HDI Deployer implements an automatic assignment of the "`default_access_role`" role if it is present in the deployed content.

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.

SAP HDI Users in XS Advanced

HDI uses several container-specific technical users to separate the different container-related tasks such as creating the schemas, triggering deployment, and creating database objects for XS advanced applications. The XS advanced application's database objects are created during application deployment with a special container-specific technical system user, the container object owner (the container's "#OO" user). Unless explicitly granted such a privilege, this user has no access to the HDI container's metadata schema and especially not to the HDI API procedures. In this way, it is possible to prevent an application procedure with definer-mode security acquiring unwanted access to metadata and the deployment API. Run-time users need the corresponding access privileges only on the container schema. Note that XS Advanced applications connect to HDI containers with container-specific, technical, run-time users. For more information about HDI users, their roles, and the scope of their access privileges, see *SAP HDI Users* in *Related Information* below.

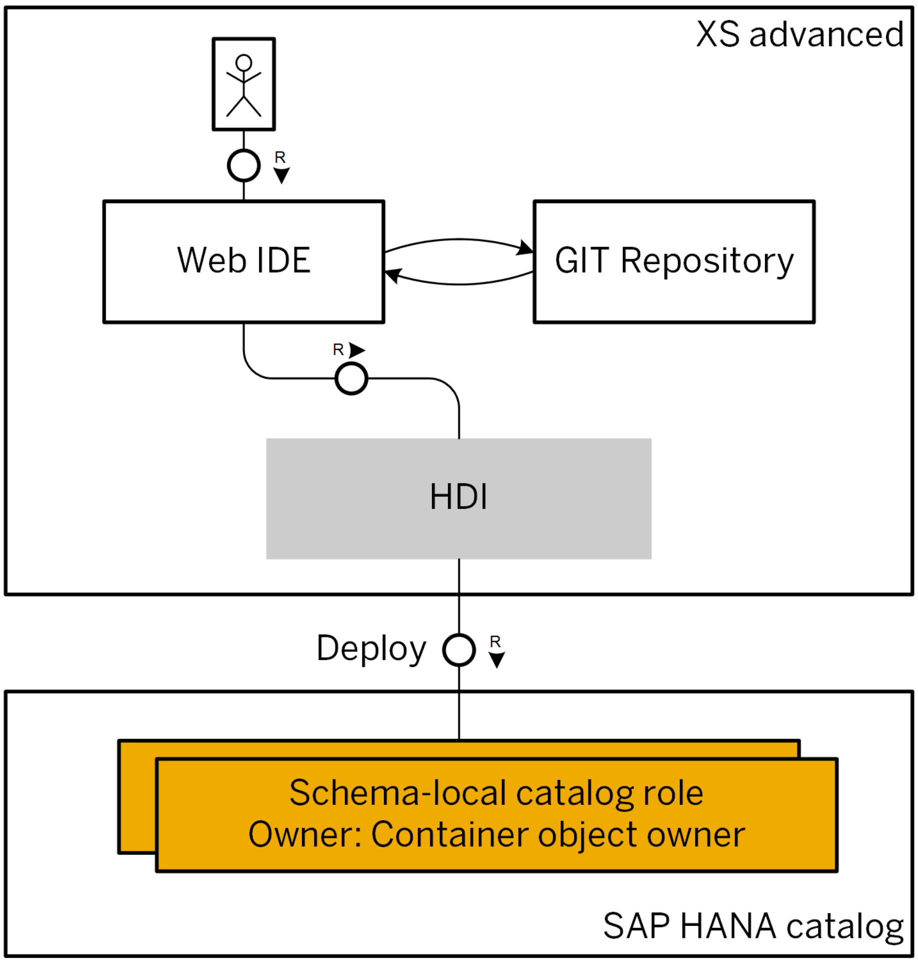
The technical users listed in the following table are used in the context of XS advanced to connect to the SAP HANA database with a specific set of conditions:

User ID	Service	Type	Description
`HDI_ADMIN_USER`	SAP HANA Broker	Technical database user	Owns the SAP HANA schema of the SAP HANA Service Broker

User ID	Service	Type	Description
`HDI_BROKER_CONTROLLER`	SAP HANA Broker	Technical database user	Has the authorization required to access the service-broker API of the SAP HANA Service Broker

SAP HDI Roles in XS Advanced

In an SAP HANA XS advanced environment, role developers use the SAP Web IDE for SAP HANA to create the design-time version of roles within a project stored in the GIT repository. When the developer deploys the project, a run-time version of the roles is created in an HDI container (schema) within the SAP HANA database.



Role Development in the SAP HANA Deployment Infrastructure (HDI)

Authorizations and HDI Role Development

The following users are involved in the creation and development of HDI roles:

- The role developer

- The owner of the HDI container ("container object owner")

The role developer must have the roles and role collections required to access the development tools, for example, SAP Web IDE for SAP HANA, and to create and manage objects in the relevant development organization and space. For more information about the authorizations required for developing HDI roles for XS advanced, for example in `.hdbrole` artifacts, see the *SAP HANA Developer Guide for XS Advanced Model* in *Related Information*.

The deployment of database objects with the HDI is based on a container model where each container corresponds roughly to a database schema. Each schema, and the database objects deployed into the schema, are owned by the HDI container object owner, a dedicated technical database user with the name `<container>#OO`. However, this user does not have any “external” privileges, for example, system privileges or object privileges, on objects in an HDI container in a different space to the role-development container or in an external schema. These privileges must be granted to the container object owner explicitly. If the container object owner does not have all privileges, role deployment will fail with a “missing authorization” error.

There are a number of ways of granting the required privileges to the container object owner. These range from simply granting the privileges to the container object owner directly to more sophisticated methods involving a user-provided service. The various alternatives are described in detail in the document *Best Practices and Recommendations for Developing Roles in SAP HANA* (see Related Information).

> i Note
>
> The container object owner must always be granted privileges on external objects with the additional parameters `WITH ADMIN OPTION` or `WITH GRANT OPTION`.

Granting and Revoking HDI Roles in XS Advanced

Roles deployed to an HDI container are granted and revoked by executing the procedures `GRANT_CONTAINER_SCHEMA_ROLES` and `REVOKE_CONTAINER_SCHEMA_ROLES` of either the container's or the container group's API. The container administrator and the container-group administrator are authorized to execute these procedures. For more information about these procedures and administrator roles, see *Maintaining SAP HDI Containers* in *Related Information*.

A role administrator (a user with the system privilege `ROLE ADMIN`) can also grant and revoke HDI roles, for example as follows:

```
GRANT <role_schema_name>.<role_name> TO <user_name>
```

Where `<role_schema_name>` is the name of the HDI container where the role was created.

> → Tip
>
> Roles can also be granted and revoked using the SAP HANA cockpit.

Dropping HDI Roles in XS Advanced

It is not possible to drop an HDI role by dropping the run-time version of the role, for example, using the SQL statement `DROP ROLE`. The role must be undeployed within the HDI container.

Auditing HDI Roles in XS Advanced

When role objects are deployed for the first time, a run-time version of the role is created in the corresponding schema in the database using SQL. The same is true if roles are changed or undeployed. You can therefore audit activity related to HDI roles with audit actions `CREATE ROLE`, `ALTER ROLE`, and `DROP ROLE`.

SAP HDI Data Storage in XS Advanced

In the context of SAP HANA extended application services, XS advanced model (XS advanced), SAP HANA is exposed as a managed service with the service name "hana". If an XS advanced 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. Services are instantiated based on a "service plan", one of which is specific to HDI: hdi-shared, which is described in more detail from the security perspective below.

Output Code

```
$ [xs | cf] marketplace -s hana

Getting services from marketplace...
Getting plans for service "hana"...

service plan   description                                      free/paid
-------------------------------------------------------------------------
hdi-shared     An HDI container on an SAP HANA database         free
sbss           User with permissions to use the SBSS            free
schema         A schema on an SAP HANA database                 free
securestore    User with permissions to use the secure store    free
```

The "hdi-shared" Service Plan

When you create a service instance with the service plan hdi-shared and bind it to an XS advanced application, the application receives the credentials it needs for access to the application's corresponding HDI container: a database schema that is equipped with additional metadata. 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 and the hdbgrants service. Within an SAP HANA database you can define an arbitrary number of HDI containers, and 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.

Database objects (tables, views, procedures, and so on) have an owner: the user who created the object. When the owner of a database object is deleted, all objects owned by the deleted user are removed from the database, too. In addition, if application objects are created by end-users, the objects are deleted when the end-user is deleted, for example when the employee leaves the organization. HDI ensures that during deployment all database objects are created by a container-specific **technical user**, which is never deleted as long as the container exists.

Each time the service instance is bound to an XS advanced application, the service broker creates a technical (application) user 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. However, 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" }
```

Related Information

The SAP HDI Deployer
Maintaining SAP HDI Containers [page 90]
SAP HDI Users [page 22]
SAP HANA Developer Guide for XS Advanced Model
Best Practices and Recommendations for Developing Roles in SAP HANA
Roles (.hdbrole) [page 244]

4.4 Enabling SAP HDI in the Database

Start the SAP HANA Deployment Infrastructure (HDI) services for the first time.

SAP HDI provides its services using a separate database process, `diserver`. If SAP HANA extended application services, advanced model (XS advanced), is installed in the system, then HDI is already enabled. Otherwise, a database administrator must enable HDI manually, for example, by starting the `diserver` process, before HDI can be used.

i Note

Depending on your scenario, further database processes may also need to be started.

Enabling HDI typically involves the following administrator tasks:

- Enable and disable HDI in the database *
- Create an HDI administrator *
- Revoke HDI administrator privileges

i Note

The SYSTEM user is required to enable HDI and create an initial HDI administrator, who can then configure HDI and create additional HDI administrators if required. If the SYSTEM user is deactivated (recommended), you will need to reactivate it temporarily to create the first HDI administrator, for example, with the SQL statement: `ALTER USER SYSTEM ACTIVATE USER NOW`. After you have completed these setup tasks, you can deactivate the SYSTEM user again and use the HDI administrator user to perform setup tasks.

Related Information

Enable SAP HDI for a Specific Tenant on a Multi-Tenant Database [page 41]
Create an SAP HDI Administrator [page 43]
Revoke the SAP HDI Administrator Privileges [page 44]

4.4.1 Enable SAP HDI for a Specific Tenant on a Multi-Tenant Database

If SAP HANA XS advanced model (XS advanced) is not installed in your system, you must enable SAP HANA Deployment Infrastructure (HDI) in the relevant tenant database.

Prerequisites

The SYSTEM user has been reactivated and you have its credentials.

> i Note
>
> SYSTEM user privileges are required to enable HDI. After enabling HDI, the SYSTEM user can be deactivated.

Context

HDI provides its services using a separate database process called `diserver`. On systems where XS advanced is installed, HDI is already enabled; on other systems where XS advanced is not installed, the `diserver` process must usually be enabled by the database administrator before HDI can be used. Additionally, if required by the usage scenario, other database process may need not be started as well.

To enable HDI for a specific tenant on a multi-tenant database, perform the following steps:

Procedure

1. In an SQL console, connect to the `SystemDB` database as the SYSTEM user.
2. Enable HDI in the target tenant database.

   Insert the following SQL code into the SQL console:

```
DO
BEGIN
  DECLARE dbName NVARCHAR(25) = 'XY1'; --<-- replace XY1 with the name of
your tenant DB
```

```
  -- Start diserver
  BEGIN
   DECLARE diserverCount INT = 0;
   SELECT COUNT(*) INTO diserverCount FROM SYS_DATABASES.M_SERVICES WHERE 
SERVICE_NAME = 'diserver' AND DATABASE_NAME = :dbName AND ACTIVE_STATUS = 
'YES';
   IF diserverCount = 0 THEN
     EXEC 'ALTER DATABASE ' || :dbName || ' ADD ''diserver''';
   END IF;
  END;
  -- [OPTIONAL] For AFLLang Procedure artifacts
  BEGIN
   DECLARE scriptserverCount INT = 0;
   SELECT COUNT(*) INTO scriptserverCount FROM SYS_DATABASES.M_SERVICES WHERE 
SERVICE_NAME = 'scriptserver' AND DATABASE_NAME = :dbName AND ACTIVE_STATUS = 
'YES';
   IF scriptserverCount = 0 THEN
     EXEC 'ALTER DATABASE ' || :dbName || ' ADD ''scriptserver''';
   END IF;
  END;
  -- [OPTIONAL] For Flow Graphs or Replication Task artifacts
  BEGIN
   DECLARE dpserverCount INT = 0;
   SELECT COUNT(*) INTO dpserverCount FROM SYS_DATABASES.M_SERVICES WHERE 
SERVICE_NAME = 'dpserver' AND DATABASE_NAME = :dbName AND ACTIVE_STATUS = 
'YES';
   IF dpserverCount = 0 THEN
     EXEC 'ALTER DATABASE ' || :dbName || ' ADD ''dpserver''';
   END IF;
  END;

  -- [OPTIONAL] For JSON DocStore and hdbcollection artifacts
  BEGIN
   DECLARE docstoreCount INT = 0;
   SELECT COUNT(*) INTO docstoreCount FROM SYS_DATABASES.M_SERVICES WHERE 
SERVICE_NAME = 'docstore' AND DATABASE_NAME = :dbName AND ACTIVE_STATUS = 
'YES';
   IF docstoreCount = 0 THEN
     EXEC 'ALTER DATABASE ' || :dbName || ' ADD ''docstore''';
   END IF;
  END;
END;
```

→ Tip

In the `DECLARE` statement, replace the name of the database `'XY1'` with the name of the tenant database for which HDI should be enabled.

a. Configure support for (Application Function Library language) procedures, if required.

   If support for "AFLLang" (Application Function Library language) procedures is required, use the optional section to start the `scriptserver` process, otherwise remove the section.

b. Configure support for flow graphs or replication tasks, if required.

   If support for flow graphs (`hdbflowgraph`) or replication tasks (`hdbreptask`) is required, use the optional paragraph to start the `dpserver` process, otherwise remove the section.

c. Configure support for the JSON Document Store (DocStore), if required.

   If support for the JSON Document Store (DocStore) and collection artifacts (`.hdbcollection`) is required, use the optional section to start the `docstore` process, otherwise remove the section.

3. Execute the SQL code.

Confirm that the SQL code completes successfully and displays the HDI return code 0.

4. Confirm that the desired processes are running.

> **i Note**
>
> Check which processes are running in the database administration view.

Related Information

SAP HDI Administrator Roles [page 18]

4.4.2 Create an SAP HDI Administrator

The HDI administrator is responsible for configuring general SAP HDI parameters, creating and dropping HDI container groups, moving HDI containers between groups, and managing the privileges of HDI container-group administrators.

Prerequisites

- The SYSTEM user has been reactivated and you have its credentials.
- SAP HDI is enabled.

Context

This method uses the predefined `_SYS_DI.T_DEFAULT_DI_ADMIN_PRIVILEGES` table, which contains the largest possible set of privileges that can be granted for a user of this type. It is also possible to reduce the set of privileges by explicitly specifying the desired set of privileges and not using this default table.

> **i Note**
>
> A variant of this procedure, `_SYS_DI.GRANT_CONTAINER_GROUP_API_PRIVILEGES_WITH_GRANT_OPTION`, also exists; it grants the given privileges `WITH GRANT OPTION` to the target user. However, this variant procedure is only needed in special scenarios, for example, when building a custom SQL API by wrapping the HDI SQL API in SQLScript procedures.

Procedure

1. In an SQL console, connect to the database with the SYSTEM user.
2. Optional: Create a new user by executing the following statement:

```
CREATE USER <HDI_admin_username> PASSWORD <password> NO
FORCE_FIRST_PASSWORD_CHANGE;
```

3. Grant the new HDI administrator user the required privileges by executing the following statement:

Sample Code

```
CREATE LOCAL TEMPORARY TABLE #PRIVILEGES LIKE _SYS_DI.TT_API_PRIVILEGES;
INSERT INTO #PRIVILEGES (PRINCIPAL_NAME, PRIVILEGE_NAME, OBJECT_NAME)
SELECT '<HDI_admin_username>', PRIVILEGE_NAME, OBJECT_NAME FROM
_SYS_DI.T_DEFAULT_DI_ADMIN_PRIVILEGES;
CALL _SYS_DI.GRANT_CONTAINER_GROUP_API_PRIVILEGES('_SYS_DI', #PRIVILEGES,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PRIVILEGES;
```

4. Execute the SQL code.

   Confirm that the SQL code completes successfully and displays the HDI return code 0.
5. Confirm that the new HDI administrator user can call HDI API procedures in the `_SYS_DI` schema.

Next Steps

Deactivate the SYSTEM user: `ALTER USER SYSTEM DEACTIVATE USER NOW`.

Related Information

Enable SAP HDI for a Specific Tenant on a Multi-Tenant Database [page 41]

4.4.3 Revoke the SAP HDI Administrator Privileges

Revoke the SAP HDI administrator privileges from a specified user.

Prerequisites

- The SYSTEM user has been reactivated and you have its credentials.
- SAP HDI is enabled.

Context

The database administrator SYSTEM can revoke the SAP HDI administrator privileges from a user.

> i Note
>
> This method uses the predefined `_SYS_DI.T_DEFAULT_DI_ADMIN_PRIVILEGES` table, which contains the largest possible set of privileges that can be revoked from a user of this type. It is also possible to reduce the set of privileges by explicitly specifying the desired set of privileges and not using this default table.

Procedure

1. In an SQL console, connect to the database with the SYSTEM user.
2. Open the SQL editor for this database.
3. Revoke the HDI administrator privileges from the selected user.

   Insert the following SQL code into the SQL console:

   > i Note
   >
   > In the following code example, replace `NEW_HDI_ADMIN` with the name of the user from whom the HDI administrator privileges should be revoked.

   ```
   CREATE LOCAL TEMPORARY TABLE #PRIVILEGES LIKE _SYS_DI.TT_API_PRIVILEGES;
   INSERT INTO #PRIVILEGES (PRINCIPAL_NAME, PRIVILEGE_NAME, OBJECT_NAME) SELECT
   'NEW_HDI_ADMIN', PRIVILEGE_NAME, OBJECT_NAME FROM
   _SYS_DI.T_DEFAULT_DI_ADMIN_PRIVILEGES WHERE NOT (PRIVILEGE_NAME = 'SELECT'
   AND OBJECT_NAME LIKE 'T%');
   CALL _SYS_DI.REVOKE_CONTAINER_GROUP_API_PRIVILEGES('_SYS_DI', #PRIVILEGES,
   _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
   DROP TABLE #PRIVILEGES;
   ```
4. Execute the SQL code.

   Confirm that the SQL code completes successfully and displays the HDI return code 0.
5. (Optional) Confirm that the `NEW_HDI_ADMIN` user is no longer able to call HDI API procedures in the `_SYS_DI` schema.

Next Steps

Deactivate the SYSTEM user, for example, with the following command in the SQL console:

```
ALTER USER SYSTEM DEACTIVATE USER NOW
```

Related Information

Enabling SAP HDI in the Database [page 40]

4.5 Maintaining the SAP HDI

Maintenance of the SAP HANA Deployment infrastructure (HDI) is the responsibility of the HDI administrator, who must set up and configure general HDI parameters.

Managing the SAP HDI typically involves the following administrator tasks:

- Configure the HDI
- List plug-in libraries available to a container
- Create and drop containers and container groups
- Grant and revoke container-group administration privileges
- Maintain containers and container groups
- Move containers between container groups

Exporting and Importing Containers

In special cases, a container and its dependencies can be exported from the database and imported into a different database for support purposes.

⚠ Caution

The API procedures `_SYS_DI.EXPORT_CONTAINER_FOR_SUPPORT` and `_SYS_DI.IMPORT_CONTAINER_FOR_SUPPORT` are available to the HDI administrator but intended for use by SAP support, exclusively. Bear in mind that the exported data might include private or confidential data from the container, and the container import could also compromise the integrity of the database.

Related Information

Configure SAP HDI Parameters [page 49]
Enabling SAP HDI in the Database [page 40]
Maintaining SAP HDI Container Groups [page 77]
Maintaining SAP HDI Containers [page 90]

4.5.1 SAP HDI Container-Group Administration

The administration of SAP HANA Deployment Infrastructure (HDI) container-groups involves the management of a set of HDI containers in an assigned HDI container group.

Most container-group-related administration tasks can be performed by both the HDI container-group administrator and the HDI administrator; the tasks include creating and dropping containers in the container group or, if necessary, granting container-administration privileges to other users of a container. However, only the HDI administrator can create and drop a container **group**. After creating a container group, the HDI administrator creates a new container-group administrator by assigning the necessary container-group administrator privileges to one or more selected users. The new container-group administrator is responsible for maintaining the containers in the new container group.

Each container group can have its own set of administrators. Administrative privileges for a container group can only be granted by an HDI administrator or a container group administrator who has the necessary privileges. Container-group administrators can only grant privileges for the container groups for which they are directly responsible.

For the HDI administrator, the APIs of a container group are located in the schema `_SYS_DI`; for the HDI container-group administrator, the APIs of a container group called “G” are located in the schema `_SYS_DI#G`. For more information about the functionality available to the HDI container-group administrator, see the section *Maintaining HDI Container Groups* listed in *Related Information*.

⚠ Caution

The administration of a container group should normally be performed by the container group’s assigned administrator. The HDI administrator should only be used for container-group administration purposes in exceptional circumstances.

Exporting and Importing HDI Containers

In special cases, a container and its dependencies can be exported from the database and imported into a different database for support purposes.

⚠ Caution

The API procedures `_SYS_DI#G.EXPORT_CONTAINER_FOR_SUPPORT` and `_SYS_DI#G.IMPORT_CONTAINER_FOR_SUPPORT` are available to the HDI container-group administrator but intended for use only by SAP support. The exported data might include private or confidential data from the container, and the container import could also compromise the integrity of the database.

The same API procedures are also available to the HDI administrator, but in the schema `_SYS_DI`, for example, `_SYS_DI.EXPORT_CONTAINER_FOR_SUPPORT`.

Related Information

Maintaining SAP HDI Container Groups [page 77]

4.5.2 SAP HDI Container Administration

Most of the tasks related to the administration of SAP HANA Deployment Infrastructure (HDI) containers can be performed by both the HDI administrator and the HDI container administrator.

HDI-container-related administrator tasks are performed by calling the respective HDI container-administration SQL procedures, not of a target container, but of the `_SYS_DI` schema of the HDI administrator with an additional first parameter taking the name of the target container. For details about the tasks required to maintain HDI containers and the functionality available to help perform these tasks, refer to the section about *Maintaining HDI Containers* listed in *Related Information*.

To grant a user "U" the privileges to access the run-time objects in the container "C", the HDI administrator can call the SQL procedure `GRANT_CONTAINER_SCHEMA_PRIVILEGES` in the `_SYS_DI` schema, passing the container's name (in this example, "C") as the additional first parameter, as illustrated in the following example:

⚠ Caution

The administration of a container should normally be performed by the container's assigned administrator. The HDI administrator should only be used for this purpose in exceptional circumstances.

Sample Code

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PRIVILEGES LIKE
_SYS_DI.TT_SCHEMA_PRIVILEGES;
INSERT INTO #PRIVILEGES ( PRIVILEGE_NAME, PRINCIPAL_SCHEMA_NAME,
PRINCIPAL_NAME ) VALUES ( 'SELECT', '', 'U' );
CALL _SYS_DI.GRANT_CONTAINER_SCHEMA_PRIVILEGES( 'C', #PRIVILEGES,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PRIVILEGES;
```

Related Information

Maintaining SAP HDI Containers [page 90]

4.5.3 SAP HDI Container Schemas

An SAP HANA Deployment Infrastructure (HDI) container makes use of multiple schemas; each of the different schemas serves different aims and tasks.

Maintaining HDI containers involves the configuration and use of the schemas listed and described in the following table:

i Note

In the following table, the schema names are based on the assumption that the base HDI container is named "C".

HDI Container Schema Names and Usage

HDI Container Schema Name	Description
C	Contains generated database objects that belong to a special object-owner user called `C#OO`. The database objects are generated from design-time objects in container C.
C#DI	Contains the API procedures and HDI-internal data required for the container management
C#OO	The schema for the user to whom the artifacts in the base container "C" belong. The user schema C#00 is empty.

Related Information

Maintaining SAP HDI Containers [page 90]
Configure SAP HDI Parameters [page 49]
Maintaining the SAP HDI [page 46]

4.5.4 Configure SAP HDI Parameters

The SAP HANA Deployment Infrastructure (HDI) administrator can configure some general aspects of the HDI with parameters, for example, how long an HDI operation waits for a locking conflict to clear or the default behavior of HDI containers.

Procedure

1. In an SQL console, connect to the database as the HDI administrator.
2. Configure the required configuration parameters by executing the following SQL statement:

```
CREATE LOCAL TEMPORARY COLUMN TABLE #CONFIG_PARAMETERS LIKE
_SYS_DI.TT_PARAMETERS;
INSERT INTO #CONFIG_PARAMETERS (KEY, VALUE) VALUES
('make.default_max_parallel_jobs', '<value>');
-- insert more parameters as required
CALL _SYS_DI.CONFIGURE_DI_PARAMETERS(#CONFIG_PARAMETERS,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #CONFIG_PARAMETERS;
```

   For a description of all available parameters, see *SAP HANA DI Configuration Parameters*.
3. Execute the SQL code.

   Confirm that the SQL code completes successfully and displays the HDI return code 0.
4. Verify that the configuration parameters have been set correctly in the `diserver.ini` configuration file.

Related Information

SAP HDI Configuration Parameters [page 66]

4.5.4.1 SAP HDI Parameters

An overview of the parameters available for the SAP HANA Deployment Infrastructure (HDI) and the corresponding build plug-ins.

In SAP HANA Deployment Infrastructure (HDI), parameters are a means of controlling the execution flow of SAP HDI procedure calls. SAP HDI includes the following parameter types:

- SAP HDI parameters
  SAP HDI parameters are used to control the execution flow of SAP HDI procedures and SAP HDI container-specific procedures. For example, they specify the time a container operation waits for a locking conflict to clear or they indicate if warnings during an SAP HDI call should be treated as errors.
- Build-plug-in parameters
  Build plug-in parameters control the execution flow of the deployment process of a build plug-in for all database objects of the corresponding type. For example, a build-plug-in parameter can be used to specify the batch size for batched database access or for batch-processing within a build plug-in.

SAP HDI Procedures

The following table lists the available parameters for SAP HDI procedures.

SAP HDI Call	Available Parameters
`_SYS_DI.CANCEL`	container_lock_wait_timeout trace_context trace_level.`<trace topic>` treat_warnings_as_errors message_severity
`_SYS_DI.CONFIGURE_CONTAINER_PARAMETERS`	container_lock_wait_timeout trace_context trace_level.`<trace topic>` message_severity
`_SYS_DI.CONFIGURE_DI_PARAMETERS`	trace_context trace_level.`<trace topic>` message_severity

SAP HDI Call	Available Parameters
`_SYS_DI.CONFIGURE_LIBRARIES`	container_lock_wait_timeout trace_context trace_level.`<trace topic>` undeploy message_severity
`_SYS_DI.CREATE_CONTAINER`	trace_context trace_level.`<trace topic>` message_severity
`_SYS_DI.CREATE_CONTAINER_GROUP`	trace_context trace_level.`<trace topic>` message_severity
`_SYS_DI.DROP_CONTAINER`	container_lock_wait_timeout ignore_deployed ignore_errors ignore_work enable_drop_enforcer trace_context trace_level.`<trace topic>` message_severity
`_SYS_DI.DROP_CONTAINER_GROUP`	move_containers_to_default_group group trace_context trace_level.`<trace topic>` message_severity
`_SYS_DI.EXPORT_CONTAINER_FOR COPY`	container_lock_wait_timeout export_container_schema_data export_container_use_binary_export trace_context trace_level.`<trace topic>` message_severity
`_SYS_DI.GRANT_CONTAINER_API_PRIVILEGES`	container_lock_wait_timeout trace_context trace_level.`<trace topic>` message_severity

SAP HDI Call	Available Parameters
`_SYS_DI.GRANT_CONTAINER_API_PRIVILEGES_WITH_GRANT_OPTION`	container_lock_wait_timeout trace_context trace_level.<trace topic> message_severity
`_SYS_DI.GRANT_CONTAINER_GROUP_API_PRIVILEGES`	trace_context trace_level.`<trace topic>` message_severity
`_SYS_DI.GRANT_CONTAINER_GROUP_API_PRIVILEGES_WITH_ GRANT_OPTION`	trace_context trace_level.`<trace topic>` message_severity
`_SYS_DI.GRANT_CONTAINER_SCHEMA_PRIVILEGES`	container_lock_wait_timeout trace_context trace_level.`<trace topic>` message_severity
`_SYS_DI.GRANT_CONTAINER_SCHEMA_ROLES`	container_lock_wait_timeout trace_context trace_level.`<trace topic>` message_severity
`_SYS_DI.GRANT_CONTAINER_SUPPORT_PRIVILEGE`	container_lock_wait_timeout trace_context trace_level.`<trace topic>` message_severity
`_SYS_DI.IMPORT_CONTAINER_FOR_COPY`	import_container_use_loopback_remote_source container_lock_wait_timeout trace_context trace_level.`<trace topic>` message_severity
`_SYS_DI.LIST_CONFIGURED_LIBRARIES`	container_lock_wait_timeout trace_context trace_level.`<trace topic>` message_severity
`_SYS_DI.LIST_LIBRARIES`	trace_context trace_level.`<trace topic>` message_severity

SAP HDI Call	Available Parameters
`_SYS_DI.MOVE_CONTAINER_TO_GROUP`	trace_context trace_level.`<trace topic>` message_severity
`_SYS_DI.REVOKE_CONTAINER_API_PRIVILEGES`	container_lock_wait_timeout trace_context trace_level.`<trace topic>` message_severity
`_SYS_DI.REVOKE_CONTAINER_GROUP_API_PRIVILEGES`	trace_context trace_level.`<trace topic>` message_severity
`_SYS_DI.REVOKE_CONTAINER_SCHEMA_PRIVILEGES`	container_lock_wait_timeout trace_context trace_level.`<trace topic>` message_severity
`_SYS_DI.REVOKE_CONTAINER_SCHEMA_ROLES`	container_lock_wait_timeout trace_context trace_level.`<trace topic>` message_severity
`_SYS_DI.REVOKE_CONTAINER_SUPPORT_PRIVILEGE`	container_lock_wait_timeout trace_context trace_level.`<trace topic>` message_severity
`_SYS_DI.CONFIGURE_CONTAINER`*	⚠ Caution Deprecated since SAP HANA 1.0 SPS 12.
`_SYS_DI.CONFIGURE_DI`*	⚠ Caution Deprecated since SAP HANA 1.0 SPS 12.

Example: Calling an SAP HDI Procedure with Parameters Set

Sample Code

```
-- prepare parameters table
create table MY_PARAMETERS like _SYS_DI.TT_PARAMETERS;
insert into MY_PARAMETERS (KEY, VALUE) values ('ignore_work', 'true');
insert into MY_PARAMETERS (KEY, VALUE) values ('ignore_deployed', 'true');
-- call procedure
call _SYS_DI.DROP_CONTAINER('MY_CONTAINER', MY_PARAMETERS, ?, ?, ?);
```

SAP HDI Container-Specific Procedures

The following table lists the available parameters for SAP HDI container-specific procedures.

SAP HDI Container-Specific Call	Available Parameters
`<container>#DI.CANCEL`	container_lock_wait_timeout trace_context trace_level.`<trace topic>` treat_warnings_as_errors message_severity
`<container>#DI.CONFIGURE_CONTAINER_PARAMETERS`	container_lock_wait_timeout trace_context trace_level.`<trace topic>` message_severity
`<container>#DI.CONFIGURE_LIBRARIES`	container_lock_wait_timeout trace_context trace_level.`<trace topic>` undeploy message_severity
`<container>#DI.DELETE`	container_lock_wait_timeout ignore_non_existing_paths recursive trace_context trace_level.`<trace topic>` message_severity
`<container>#DI.EXPORT_CONTAINER_FOR_COPY`	container_lock_wait_timeout export_container_schema_data export_container_use_binary_export trace_context trace_level.`<trace topic>` message_severity
`<container>#DI.GET_DEPENDENCIES`	container_lock_wait_timeout trace_context trace_level.`<trace topic>` variant message_severity

SAP HDI Container-Specific Call	Available Parameters
`<container>#DI.GET_MAKE_GROUPS`	container_lock_wait_timeout max_parallel_jobs optimized_redeploy simulate_make skip_unchanged_expansions trace_context trace_level.`<trace topic>` treat_warnings_as_errors undeploy_dependent_recursively message_severity
`<container>#DI.GRANT_CONTAINER_API_PRIVILEGES`	container_lock_wait_timeout trace_context trace_level.`<trace topic>` message_severity
`<container>#DI.GRANT_CONTAINER_API_PRIVILEGES_WITH_GRANT_OPTION`	container_lock_wait_timeout trace_context trace_level.`<trace topic>` message_severity
`<container>#DI.GRANT_CONTAINER_SCHEMA_PRIVILEGES`	container_lock_wait_timeout trace_context trace_level.`<trace topic>` message_severity
`<container>#DI.GRANT_CONTAINER_SCHEMA_ROLES`	container_lock_wait_timeout trace_context trace_level.`<trace topic>` message_severity
`<container>#DI.IMPORT_CONTAINER_FOR_COPY`	import_container_use_loopback_remote_source container_lock_wait_timeout trace_context trace_level.`<trace topic>` message_severity
`<container>#DI.LIST`	container_lock_wait_timeout ignore_files ignore_folders recursive trace_context trace_level.`<trace topic>` message_severity
`<container>#DI.LIST_CONFIGURED_LIBRARIES`	container_lock_wait_timeout trace_context trace_level.`<trace topic>` message_severity

SAP HDI Container-Specific Call	Available Parameters
`<container>#DI.LIST_DEPLOYED`	container_lock_wait_timeout ignore_files ignore_folders recursive trace_context trace_level.`<trace topic>` message_severity
`<container>#DI.MAKE`	container_lock_wait_timeout max_parallel_jobs optimized_redeploy simulate_make skip_unchanged_expansions trace_context trace_level.`<trace topic>` treat_warnings_as_errors undeploy_dependent_recursively validate_virtual_tables enable_make_enforcer message_severity
`<container>#DI.MAKE_ASYNC`	container_lock_wait_timeout max_parallel_jobs optimized_redeploy simulate_make skip_unchanged_expansions trace_context trace_level.`<trace topic>` treat_warnings_as_errors undeploy_dependent_recursively validate_virtual_tables enable_make_enforcer message_severity
`<container>#DI.READ`	container_lock_wait_timeout ignore_files ignore_folders recursive trace_context trace_level.`<trace topic>` message_severity

SAP HDI Container-Specific Call	Available Parameters
`<container>#DI.READ_DEPLOYED`	container_lock_wait_timeout ignore_files ignore_folders recursive trace_context trace_level.`<trace topic>` message_severity
`<container>#DI.REVOKE_CONTAINER_API_PRIVILEGES`	container_lock_wait_timeout trace_context trace_level.`<trace topic>` message_severity
`<container>#DI.REVOKE_CONTAINER_SCHEMA_PRIVILEGES`	container_lock_wait_timeout trace_context trace_level.`<trace topic>` message_severity
`<container>#DI.REVOKE_GRANT_CONTAINER_SCHEMA_ROLES`	container_lock_wait_timeout trace_context trace_level.`<trace topic>` message_severity
`<container>#DI.STATUS`	container_lock_wait_timeout trace_context trace_level.`<trace topic>` message_severity
`<container>#DI.WRITE`	container_lock_wait_timeout trace_context trace_level.`<trace topic>` message_severity
`<container>#DI.CONFIGURE_CONTAINER`	⚠ Caution Deprecated since SAP HANA 1.0 SPS 12.

Example: Calling a Container-Specific Procedure with Parameters Set

Sample Code

```
-- prepare path content table
create table MY_PATH_CONTENT like _SYS_DI.TT_FILESFOLDERS_CONTENT;
insert into MY_PATH_CONTENT (PATH, CONTENT) values ('mypath/', '');
insert into MY_PATH_CONTENT (PATH, CONTENT) values ('mypath/
myfile1.hdbtable', 'ROW TABLE MY_TABLE (X INTEGER)');
insert into MY_PATH_CONTENT (PATH, CONTENT) values ('mypath/.hdiconfig', '{
"file_suffixes" : { "hdbtable" : { "plugin_name" : "com.sap.hana.di.table",
"plugin_version" : "12.0.0" } } }');

-- prepare parameters table
create table MY_PARAMETERS like _SYS_DI.TT_PARAMETERS;
```

```
insert into MY_PARAMETERS (KEY, VALUE) values ('container_lock_wait_timeout',
'10');
-- call procedure
call MY_CONTAINER#DI.WRITE(MY_PATH_CONTENT, MY_PARAMETERS, ?, ?, ?);
```

Available SAP HDI Parameters

The following table describes the parameters available in SAP HDI and their possible values.

Parameter	Possible values	Description
`container_lock_wait_timeout`	0 ... 2,147,483,647	Specifies the time (in milliseconds) a container operation waits for a locking conflict to clear. The default value is the value of the corresponding SAP HDI configuration parameter `connection.container_default_transaction_lock_wait_timeout`. For more information, see *SAP HDI Configuration Parameters*.
`accept_risk_of_database_corruption_by_container_import`	true, false	Indicates that the user knows that a container import could potentially corrupt the database. The default value is “false”.
`enable_drop_enforcer`	true, false	If “true”, terminate all external database connections blocking a make call and, in addition, all external database connections holding data in a global temporary table in the container schema. The default value is true.
`enable_make_enforcer`	true, false	If “true”, terminate all external database connections blocking a make call. The default value is “true”.
`export_container_schema_data`	true, false	Indicates that a container export should also export the data of the container schema. If set to “false”, only the schema structure will be exported. The default value is “true”.

Parameter	Possible values	Description
`export_container_use_binary_export`	true, false	Indicates that the container export should export data in a binary format. If set to “false”, the data is exported as CSV. The default value is “true”
`ignore_deployed`	true, false	Indicates if existing files in the deployed file system are to be ignored when dropping a container. The default value is “false”.
`ignore_errors`	true, false	Indicates if errors during an SAP HDI call should be ignored, that is, execute and commit as many internal operations as possible. Failing operations are reported to the user. The default value is “false”.
`ignore_files`	true, false	Indicates if files are to be ignored in the output when reading files. The default value is “false”.
`ignore_folders`	true, false	Indicates if folders are to be ignored in the output when reading files. The default value is “false”.
`ignore_non_existing_paths`	true, false	Indicates if paths that do not exist should be ignored, for example, when deleting folders. The default value is “false”.
`ignore_work`	true, false	Indicates if existing files in the work file system are to be ignored when dropping a container. The default value is “false”.

Parameter	Possible values	Description
`import_container_use_loopback_remote_source`	true, false	Virtual tables can be imported, if the remote source and the tables behind it are not there. When importing tables, a loopback remote source is created that points to the local SAP HANA server. For every virtual table imported, a corresponding dummy local table is created with the same metadata. Virtual tables are then created by using this loopback remote source and the dummy local tables. The default value is “false”.
`max_parallel_jobs`	0 … 2,147,483,647	Specifies the maximum number of parallel jobs for graph execution and artifact deployment. The default value is 8.
`move_containers_to_default_ group`	true, false	Indicates if all containers in a group should be moved to the default group _SYS_DI first before dropping the container group. The default value is “false”.
`optimized_redeploy`	true, false	Indicates if the optimized redeployment strategy should be used instead of the undeploy-deploy mechanism when redeploying artifacts. The default value is “true”.
`recursive`	true, false	Indicates if folders are to be read or deleted recursively. The default value is “false”.
`simulate_make`	true, false	Indicates if the make should run only in simulation mode. The default value is “false”.

Parameter	Possible values	Description
`trace_context`	request, container	Indicates if, during an SAP HDI request, all traces for trace topics configured using the `trace_level_<trace topic>` parameter are written to a separate trace file in addition to the DI server trace file. If set to “request”, a new trace file is created for the request. For container operations, if set to “container”, a trace file for the corresponding container is created or appended to. There is no default value.
`trace_level.<trace topic>`	Fatal, Error, Warning, Info, Interface, Debug, InterfaceFull, DebugFull	Specifies the trace level of a specific trace topic. `<trace topic>` may be an arbitrary SAP HANA trace topic. There is no default value.
`treat_warnings_as_errors`	true, false	Indicates if warnings during an SAP HDI call should be treated as errors. The default value is “false”.
`undeploy`	true, false	Indicates if, in case of a call to configure libraries, files corresponding to a library to be removed should also be undeployed. The default value is “false”.
`undeploy_dependent_recursively`	true, false	Indicates that all dependent artifacts should be considered for an undeployment. The default value is “false”.
`validate_virtual_tables`	true, false	Indicates that all deployed virtual tables should be checked if a redeployment is necessary. The default value is “false”.

Parameter	Possible values	Description
`variant`	“providers”, “provides_and_requires”, “impacted”, “depends”	Specifies the variant of the SAP HDI (HDI) container-specific procedure `<container>#DI.GET_DEPENDENCIES` to be used; the following values are permitted: • “`providers`” Returns the providing file for a given database object • “`provides_and_requires`” Returns the provided and required database objects for a given artifact • “`impacted`” Returns the files that depend on (are affected by) the specified files. • “`depends`” Returns the files that the specified files depend on. The default value is “providers”.
`message_severity`	INFO, WARNING, ERROR	Specifies the minimum severity of the messages to be returned by a SAP HDI procedure call. The default value is INFO (all messages are returned).

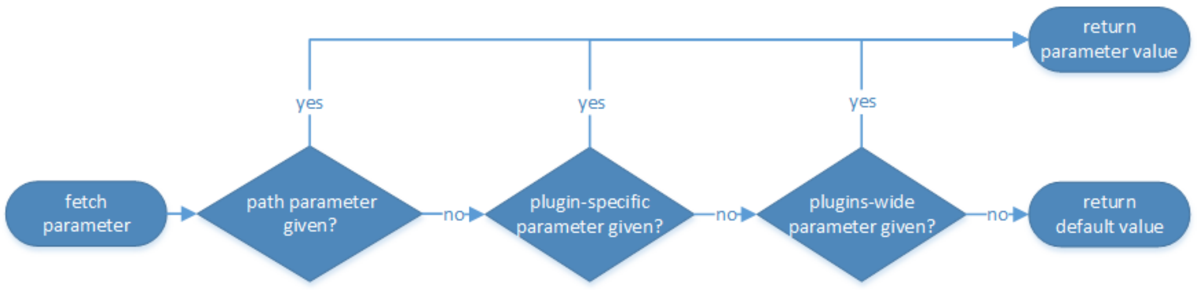
Parameters for Build Plugins

SAP HDI supports three types of parameters for controlling the execution flow of build plug-ins. On a global level, a “plug-ins-wide parameter” applies to all build plug-ins supporting the parameter. On a more fine-grained level, a “plug-in-specific parameter” only applies to the specified build plug-in. Eventually, a “path parameter” serves to control the handling of a single file.

The following table describes the structure of each parameter type.

Parameter Type	File	Parameter Structure
plug-ins-wide parameter	-	com.sap.hana.di/<key>
plug-in-specific parameter	-	com.sap.hana.di.<plug-in>/<key>
path parameter	<file>	<key>

From the point of view of a build plug-in, the three types of parameters form a hierarchy whereby the most specific parameter type is considered first. The following diagram visualizes the process of fetching a parameter from the point of view of a build plug-in.



The plug-in-specific parameters support additional layering by allowing additional layers within the parameter structure. For example, if a requested parameter `com.sap.hana.di.<layer1>.<plug-in1>/<key>` is not found, the build plug-in automatically searches for a parameter `com.sap.hana.di.<layer1>/<key>`. The following table shows an example of layering of parameters.

Parameter Type	File	Parameter Structure
plug-ins-wide parameter	-	com.sap.hana.di/`<key>`
layered parameter	-	com.sap.hana.di.`<layer1>`/`<key>`
layered plug-in parameter	-	com.sap.hana.di.`<layer1>`.`<plug-in1>`/`<key>`
path parameter	<file>	`<key>`

The following section lists the available build plug-in parameters and path parameters in SAP HDI. The section *Build Plugin Parameters* describes the available parameters for the configuration of uild plug-ins.

Build Plug-ins

The following table lists the available parameters for the build plug-ins.

Build Plug-in	Available Build Plug-in Parameters	Available Path Parameters

Applies to all build plug-ins	`optimized_redeploy`
	`skip_unchanged_expansions`
	`undeploy_dependent_recursively`
		-

com.sap.hana.di.cds	`force_undeploy`	-

Build Plug-in	Available Build Plug-in Parameters	Available Path Parameters
com.sap.hana.di.role	`force_undeploy`	-
com.sap.hana.di.sequence	`force_undeploy`	-
com.sap.hana.di.table	`force_undeploy`	-
com.sap.hana.di.tabledata	`batch_size`	-
com.sap.hana.di.table.migration	`development_mode`	-

Example for calling the make procedure with a plug-ins-wide parameter set:

Sample Code

```
-- prepare deploy paths table
create table MY_DEPLOY_PATHS like _SYS_DI.TT_FILESFOLDERS;
insert into MY_DEPLOY_PATHS (PATH) values ('mypath/myfile1.hdbtable');
insert into MY_DEPLOY_PATHS (PATH) values ('mypath/.hdiconfig');
-- prepare parameters table with a plug-ins-wide parameter
create table MY_PARAMETERS like _SYS_DI.TT_PARAMETERS;
insert into MY_PARAMETERS (KEY, VALUE) values ('com.sap.hana.di /
force_undeploy', 'true');
-- call procedure
call MY_CONTAINER#DI.MAKE(MY_DEPLOY_PATHS, _SYS_DI.T_NO_FILESFOLDERS,
_SYS_DI.T_NO_FILESFOLDERS_PARAMETERS, MY_PARAMETERS, ?, ?, ?);
```

Example for calling the make procedure with a plug-in-specific parameter set:

Sample Code

```
-- prepare deploy paths table
create table MY_DEPLOY_PATHS like _SYS_DI.TT_FILESFOLDERS;
insert into MY_DEPLOY_PATHS (PATH) values ('mypath/myfile1.hdbtable');
insert into MY_DEPLOY_PATHS (PATH) values ('mypath/.hdiconfig');
-- prepare parameters table with a plug-in-specific parameter
create table MY_PARAMETERS like _SYS_DI.TT_PARAMETERS;
insert into MY_PARAMETERS (KEY, VALUE) values ('com.sap.hana.di.table/
force_undeploy', 'true');
-- call procedure
call MY_CONTAINER#DI.MAKE(MY_DEPLOY_PATHS, _SYS_DI.T_NO_FILESFOLDERS,
_SYS_DI.T_NO_FILESFOLDERS_PARAMETERS, MY_PARAMETERS, ?, ?, ?);
```

Example for calling the make procedure with a path parameter set:

Sample Code

```
-- prepare deploy paths table
create table MY_DEPLOY_PATHS like _SYS_DI.TT_FILESFOLDERS;
insert into MY_DEPLOY_PATHS (PATH) values ('mypath/myfile1.hdbtable');
insert into MY_DEPLOY_PATHS (PATH) values ('mypath/.hdiconfig');
-- prepare path parameters table
create table MY_PATH_PARAMETERS like _SYS_DI.TT_FILESFOLDERS_PARAMETERS;
insert into MY_PATH_PARAMETERS (PATH, KEY, VALUE) values ('mypath/
myfile1.hdbtable', 'force_undeploy', 'true');
-- call procedure
```

```
call MY_CONTAINER#DI.MAKE(MY_DEPLOY_PATHS, _SYS_DI.T_NO_FILESFOLDERS,
MY_PATH_PARAMETERS, _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
```

Build Plug-in Parameters

The following table describes the build plug-in parameters available in SAP HDI and their possible values.

Build Plug-in Parameter	Possible Values	Description
`batch_size`	0 … 2,147,483,647	Specifies the batch size, for example, for batch database access or for batch processing within a build plug-in. The default value is "1".
`development_mode`	true, false	Indicates to the migration-table build plug-in (`hdbmigrationtable`) that it should use development mode, if available. For development purposes, a development version with separate definition can be specified, which is the base for the current migration development. In development mode, all data stored in the table is lost. The default value is "false".
`force_undeploy`	true, false	Indicates if the undeployment of files should be forced within a build plug-in that would alter an existing database object instead of simply re-creating it. The default value is "false".
`optimized_redeploy`	true, false	Indicates if the optimized redeployment strategy should be used instead of the undeploy-deploy mechanism when redeploying artifacts. The default value is "true".
`skip_unchanged_expansions`	true, false	Instructs the make expander to not add expansions with same name and content as a deployed expansion to the deploy and undeploy sets. The default value is "false".

Build Plug-in Parameter	Possible Values	Description
`undeploy_dependent_recursively`	true, false	Indicates that all dependent artifacts should be considered for an undeployment. The default value is "false".

Related Information

SAP HDI Configuration Parameters [page 66]

4.5.4.2 SAP HDI Configuration Parameters

Configuration parameters are used to configure the behavior of SAP HANA Deployment Infrastructure (HDI). There are two types of configuration parameters: HDI configuration parameters and HDI **container-specific** configuration parameters.

SAP HDI configuration parameters configure the general behavior of SAP HDI. For example, they specify the time an SAP HDI operation waits for a locking conflict to clear or they specify the default behavior of containers.

Container-specific configuration parameters are used to control the behavior of a single container. For example, they specify the time a container operation waits for a locking conflict to clear or the maximum number of parallel jobs to be spawned during a make.

SAP HDI Configuration Parameters

Parameter	Possible Values	Description
`api.severity_for_invalid_parameter`	ERROR, WARNING, INFO	Specifies the severity of the corresponding log message when an invalid parameter has been passed with the SAP HANA DI operation. The default value is ERROR.
`blobs.container_default_days_to_keep`	-2,147,483,648 ... 2,147,483,647	Specifies the default number of days to keep data entries in the container-specific blob store. A value of 0 means deleting all entries. A negative value means keeping all entries. The default value is 10.

Parameter	Possible Values	Description
`blobs.transaction_lock_wait_timeout`	0 … 2,147,483,647	Specifies the time (in milliseconds) a blob store operation waits for a locking conflict to clear. The default value is 100,000.
`connection.container_default_transaction_lock_wait_timeout`	0 … 2,147,483,647	Specifies the default time (in milliseconds) a container operation waits for a locking conflict to clear. The default value is the value of the corresponding SAP HANA DI configuration parameter `connection.global_transaction_lock_wait_timeout`.
`connection.global_transaction_lock_wait_timeout`	0 … 2,147,483,647	Specifies the time (in milliseconds) an SAP HANA DI operation waits for a locking conflict to clear. The default value is 100,000.
`connection.max_polls_for_master_indexserver`	0 … 2,147,483,647	Specifies the maximum number of polls for the master indexserver before aborting the SAP HANA DI operation. The default value is 100.
`connection.poll_interval_for_master_indexserver`	0 … 2,147,483,647	Specifies the interval (in seconds) between polls for the master indexserver. The default value is 5.
`connection.max_retries_for_initialization`	0 … 2,147,483,647	Specifies the maximum number of retries to initialize the DI-server during startup. The default value is 10.
`make.default_max_parallel_jobs`	0 … 2,147,483,647	Specifies the default maximum number of parallel jobs to be spawned during a make. The default value is 0.
`make.validate_virtual_tables`	true, false	Indicates that during a make all deployed virtual tables should be checked if a redeployment is necessary. The default value is “false”.

Parameter	Possible Values	Description
`messages.container_default_days_to_keep`	-2,147,483,648 ... 2,147,483,647	Specifies the default number of days to keep container-specific log messages. A value of 0 means all entries are deleted. A negative value means all entries are kept. The default value is the value of the corresponding SAP HANA DI configuration parameter `messages.global_days_to_keep`.
`messages.container_default_requests_to_keep`	-2,147,483,648 ... 2,147,483,647	Specifies the default number of requests to keep container-specific log messages. A value of 0 means all entries are deleted. A negative value means all entries are kept. The default value is the value of the corresponding SAP HANA DI configuration parameter `messages.global_requests_to_keep`.
`messages.global_days_to_keep`	-2,147,483,648 ... 2,147,483,647	Specifies the number of days to keep global SAP HANA DI log messages. A value of 0 means all entries are deleted. A negative value means all entries are kept. The default value is 10.
`messages.global_requests_to_keep`	-2,147,483,648 ... 2,147,483,647	Specifies the number of requests to keep global SAP HANA DI log messages. A value of 0 means all entries are deleted. A negative value means all entries are kept. The default value is 100.
`messages.treat_errors_as_warnings`	A list of comma-separated error codes	Specifies error codes whose corresponding messages should be treated as warnings instead of errors. The default value is an empty list.
`trace.max_content_bytes_traced`	0 ... 2,147,483,647	Specifies the maximum length (in bytes) of a content to be traced. The default value is 100.

Example: Configuring SAP HANA DI with an SAP HANA DI Configuration Parameter

Sample Code

```
-- prepare configuration parameters table
create table MY_CONFIG_PARAMETERS like _SYS_DI.TT_PARAMETERS;
insert into MY_CONFIG_PARAMETERS(KEY, VALUE) values
('make.default_max_parallel_jobs', '10');
-- prepare parameters table
create table MY_PARAMETERS like _SYS_DI.TT_PARAMETERS;
-- call procedure
call _SYS_DI.CONFIGURE_DI_PARAMETERS(MY_CONFIG_PARAMETERS,
MY_PARAMETERS, ?, ?, ?);
```

Container-Specific Configuration Parameters

The following table describes the container-specific configuration parameters and their possible values.

SAP HANA DI Container-specific Configuration Parameters

Parameter	Possible Values	Description
`blobs.days_to_keep`	-2,147,483,648 ... 2,147,483,647	Specifies the number of days to keep data entries in the blob store. A value of 0 means all entries are deleted. A negative value means all entries are kept. The default value is the value of the corresponding SAP HANA DI configuration parameter `blobs.container_default_days_to_keep`.
`connection.transaction_lock_wait_timeout`	0 ... 2,147,483,647	Specifies the time (in milliseconds) a container operation waits for a locking conflict to clear. The default value is the value of the corresponding SAP HANA DI configuration parameter `connection.container_default_transaction_lock_wait_timeout`.

Parameter	Possible Values	Description
`global.usage`	production, test, development, custom	Indicates the intended usage of the container for which the make operation is run. Depending on the usage, different restrictions might apply. If this configuration parameter is not set, then the value is taken from the `system information` section in `global.ini` configuration file.
`make.force_logical_schema_targets`	"true" or "false"	Indicates if the usage of logical schemas in design-time artifacts should be forced. The default value is "false".
`make.max_parallel_jobs`	0 ... 2,147,483,647	Specifies the maximum number of parallel jobs to be spawned during a make. The default value is the value of the corresponding SAP HANA DI configuration parameter `make.default_max_parallel_jobs`.
`make.prohibit_config_file`	"true" or "false"	Indicates if the deployment of configuration files (`.hdiconfig`) should be disabled. The default value is "false".
`make.prohibit_definer_mode`	"true" or "false"	Indicates if the creation of "definer-mode" procedures is disabled. The default value is "false".
`make.prohibit_namespace_file`	"true" or "false"	Indicates if the deployment of namespace files (`.hdinamespace`) should be disabled. The default value is "false".
`make.prohibit_table_creation`	"true" or "false"	Indicates if the creation of tables is disabled. The default value is "false".

Parameter	Possible Values	Description
messages.days_to_keep	-2,147,483,648 ... 2,147,483,647	Specifies the number of days to keep log messages. A value of 0 means all entries are deleted. A negative value means all entries are kept. The default value is the value of the corresponding SAP HANA DI configuration parameter messages.container_default_days_to_keep.
messages.requests_to_keep	-2,147,483,648 ... 2,147,483,647	Specifies the number of requests to keep log messages. A value of 0 means all entries are deleted. A negative value means all entries are kept. The default value is the value of the corresponding SAP HANA DI configuration parameter messages.container_default_requests_to_keep.
build.plugin.name/disabled	"true" or "false"	Indicated if the build plugin specified by the parameter is disabled. By default, there is no such parameter set; no build plugin is disabled by default..

Example: Configuring a Container with a Container-specific Configuration Parameter

Sample Code

```
-- prepare configuration parameters table
create table MY_CONFIG_PARAMETERS like _SYS_DI.TT_PARAMETERS;
insert into MY_CONFIG_PARAMETERS(KEY, VALUE) values
('make.max_parallel_jobs', '10');
-- prepare parameters table
create table MY_PARAMETERS like _SYS_DI.TT_PARAMETERS;
-- call procedure
call MY_CONTAINER#DI.CONFIGURE_CONTAINER_PARAMETERS(MY_CONFIG_PARAMETERS,
MY_PARAMETERS, ?, ?, ?);
```

4.5.5 List The Plug-in Libraries That Can Be Configured for an SAP HDI Container

You can find out which SAP HANA Deployment Infrastructure (HDI) plug-in libraries and versions are available in the database and can be configured for use in an HDI container.

Procedure

1. In an SQL console, connect to the database as the HDI administrator.
2. Display a list of plug-in libraries available in the HDI container.

   Run the following SQL statement

   ```
   CALL _SYS_DI.LIST_LIBRARIES(_SYS_DI.T_NO_PARAMETERS, ?, ?, ?, ?);
   ```

   Confirm that the SQL code completes successfully and displays the HDI return code 0.
3. Examine the result set returned by SQL.

   The following example shows an excerpt of a typical result set for this request:

   Output Code

   Excerpt of result set

   ```
   LIBRARY_NAME;LIBRARY_VERSION;PLUGIN_ID;PLUGIN_VERSION
   com.sap.hana.di.afllangprocedure;0.0;com.sap.hana.di.afllangprocedure;
   2.0.10.0
   com.sap.hana.di.analyticprivilege;0.0;com.sap.hana.di.analyticprivilege;
   2.0.10.0
   com.sap.hana.di.calculationview;0.0;com.sap.hana.di.calculationview;
   2.0.10.0
   com.sap.hana.di.cds;0.3;com.sap.hana.di.cds;2.0.10.0
   com.sap.hana.di.constraint;0.0;com.sap.hana.di.constraint;2.0.10.0
   com.sap.hana.di.copyonly;0.0;com.sap.hana.di.copyonly;2.0.10.0
   com.sap.hana.di.dropcreatetable;0.0;com.sap.hana.di.dropcreatetable;
   2.0.10.0
   com.sap.hana.di.flowgraph;0.0;com.sap.hana.di.flowgraph;2.0.10.0
   ```

4.5.6 Create an SAP HDI Container Group

In SAP HANA Deployment Infrastructure (HDI), an HDI container group is used for administrating a set of HDI containers. Each HDI container group can be managed by different (HDI administrator) users.

Procedure

1. In an SQL console, connect to the database as the HDI administrator.
2. Insert the following SQL statement:

   ```
   CALL _SYS_DI.CREATE_CONTAINER_GROUP('<container_group_name>',
   _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
   ```

   > **i Note**
   >
   > Replace the name of the container group `<container_group_name>` in the SQL `CALL` statement with your container-group name

3. Execute the SQL code.

   Confirm that the SQL code completes successfully and displays the HDI return code 0.
4. Confirm that the new container group has been created.

   > **→ Tip**
   >
   > The system view `_SYS_DI.M_ALL_CONTAINER_GROUPS` contains a list of **all** created HDI container groups.

4.5.7 Drop an SAP HDI Container Group

The SAP HANA Deployment Infrastructure (HDI) administrator can drop an HDI container group.

Prerequisites

The HDI container group must be empty (no HDI containers are assigned).

Procedure

1. In an SQL console, connect to the database as the HDI administrator.
2. Drop the specified container group.

Insert the following SQL statement into the SQL console:

```
CALL _SYS_DI.DROP_CONTAINER_GROUP('<container_group_name>',
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
```

> i Note
>
> Replace the name of the container group `<container_group_name>` in the SQL `CALL` statement with your container-group name

3. Execute the SQL code.

   Confirm that the SQL code completes successfully and displays the HDI return code 0.
4. Confirm that the schema _SYS_DI#`<container_group_name>` no longer exists.

4.5.8 Grant SAP HDI Container-Group Administrator Privileges to Another User

An SAP HANA Deployment Infrastructure (HDI) administrator or a container-group administrator must explicitly grant administrative privileges for a container group.

Context

Every HDI container group can have its own set of administrators. An HDI administrator can grant another user administrator privileges for **any** HDI container group, whereas an HDI container-group administrator can only grant another user the container-group administrator privileges for his (or her) own HDI container group. This method uses the predefined `_SYS_DI.T_DEFAULT_CONTAINER_GROUP_ADMIN_PRIVILEGES` table, which contains the largest possible set of privileges that can be granted for a user of this type. You can reduce the set of privileges granted by explicitly specifying the desired set of privileges and not using this default table.

> i Note
>
> A variant of this procedure, `_SYS_DI.GRANT_CONTAINER_GROUP_API_PRIVILEGES_WITH_GRANT_OPTION`, also exists; it grants the specified privileges `WITH GRANT OPTION` to the target user. This variant is only needed in special scenarios, for example, when building a custom SQL API by wrapping the HDI SQL API in SQLScript procedures.

Procedure

1. In an SQL console, connect to the database as the HDI administrator.
2. Grant container-group administrator privileges to a specified user.

Insert the following SQL statement into the SQL console:

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PRIVILEGES LIKE
_SYS_DI.TT_API_PRIVILEGES;
INSERT INTO #PRIVILEGES (PRINCIPAL_NAME, PRIVILEGE_NAME, OBJECT_NAME) SELECT
'<new_container_group_admin_username>', PRIVILEGE_NAME, OBJECT_NAME FROM
_SYS_DI.T_DEFAULT_CONTAINER_GROUP_ADMIN_PRIVILEGES;
CALL _SYS_DI.GRANT_CONTAINER_GROUP_API_PRIVILEGES('<container_group_name>',
#PRIVILEGES, _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PRIVILEGES;
```

> → Tip
>
> Replace the name of the user `<container_group_admin_username>` in `INSERT` command in line 2 with the name of the user to whom the API privileges should be granted, and replace the name of the container group `<container_group_name>` in the `CALL` command in line 3 with the name of the desired container group name.

3. Execute the SQL code.

   Confirm that the SQL code completes successfully and displays the HDI return code 0.
4. Confirm that the new container group administrator is able to call HDI API procedures in the API schema _SYS_DI#`<container_group_name>`.

4.5.9 Revoke SAP HDI Container-Group Administrator Privileges from a Container Group Administrator

An SAP HANA Deployment Infrastructure (HDI) administrator can revoke administration privileges on any HDI container group from a user.

Procedure

1. In an SQL console, connect to the database as the HDI administrator.
2. Revoke the specified container-group administrator privileges.

   Insert the following SQL statement into the SQL console:

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PRIVILEGES LIKE
_SYS_DI.TT_API_PRIVILEGES;
INSERT INTO #PRIVILEGES (PRINCIPAL_NAME, PRIVILEGE_NAME, OBJECT_NAME) SELECT
'NEW_CONTAINER_GROUP_ADMIN', PRIVILEGE_NAME, OBJECT_NAME FROM
_SYS_DI.T_DEFAULT_CONTAINER_GROUP_ADMIN_PRIVILEGES WHERE NOT (PRIVILEGE_NAME
= 'SELECT' AND OBJECT_NAME LIKE '_SYS_DI.T%');
CALL _SYS_DI.REVOKE_CONTAINER_GROUP_API_PRIVILEGES('G', #PRIVILEGES,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PRIVILEGES;
```

> → Tip
>
> Replace the name of the user `<NEW_CONTAINER_GROUP_ADMIN>` in `INSERT` command in line 2 with the name of the user from whom the API privileges should be revoked, and replace the name of the

container group <G> in the CALL command in line 3 with the name of the desired container group name.

3. Execute the SQL code.

   Confirm that the SQL code completes successfully and displays the HDI return code 0.
4. Confirm that the container group administrator is no longer able to call HDI API procedures in the API schema _SYS_DI#<container_group_name>.

Related Information

Maintaining SAP HDI Container Groups [page 77]

Grant Container-Group Administrator Privileges to a User [page 78]

4.5.10 Move an SAP HDI Container to Another Container Group

The SAP HANA Deployment Infrastructure (HDI) administrator can move an HDI container to another container group.

Procedure

1. In an SQL console, connect to the database as the HDI administrator.
2. Move the specified container to the new target container group.

   Insert the following SQL statement into the SQL console:

   ```
   CALL _SYS_DI.MOVE_CONTAINER_TO_GROUP('<container_to_be_moved>',
   '<target_container_group>', _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
   ```
3. Execute the SQL code.

   Confirm that the SQL code completes successfully and displays the HDI return code 0.
4. Using the M_ALL_CONTAINERS monitoring view, check that <container_group_to_be_moved> has the container group <target_container_group> assigned in the CONTAINER_GROUP_NAME column.

4.6 Maintaining SAP HDI Container Groups

The administrator of an SAP HDI container group is responsible for managing the SAP HDI containers that are organized into one or more HDI container groups.

In SAP HANA Deployment Infrastructure (HDI), managing an HDI container group typically involves the following administrator tasks:

- Grant and revoke container-group administrator privileges
- Create and drop containers
- Grant and revoke container administrator privileges
- Grant and revoke container access

→ Tip

The APIs of a container group named “G” are in the schema `_SYS_DI#G`.

HDI Container Administration

The HDI container group administrator can perform the same administrative tasks as an HDI container administrator. For details about the functionality available to the HDI container administrator, see the section about HDI container administration in *Related Information* below. To perform container-related administration tasks, the container-group administrator calls the appropriate HDI container administration SQL procedures, not of the target container, but of the container **group schema** (for example, `_SYS_DI#G`) of the HDI container group administrator. The name of the target container is specified by means of an additional first parameter, as illustrated in the sample below.

Sample Code

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PRIVILEGES LIKE
_SYS_DI.TT_SCHEMA_PRIVILEGES;
INSERT INTO #PRIVILEGES ( PRIVILEGE_NAME, PRINCIPAL_SCHEMA_NAME,
PRINCIPAL_NAME ) VALUES ( 'SELECT', '', 'U' );
CALL _SYS_DI#G.GRANT_CONTAINER_SCHEMA_PRIVILEGES( 'C', #PRIVILEGES,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PRIVILEGES;
```

The example above shows how to grant a user “U” the privileges to access the run-time objects in the container “C”. The HDI container group administrator calls the SQL procedure `GRANT_CONTAINER_SCHEMA_PRIVILEGES` in the `_SYS_DI` schema, passing the container’s name “C” as the additional first parameter:

i Note

Container-administration tasks should normally be performed by the container’s assigned administrator. The HDI container **group** administrator should only be used to perform every-day container administration tasks in exceptional circumstances. The only exception to this rule is the creation and dropping of containers, which can only be performed by the container **group** administrator.

Exporting and Importing Containers

In special cases, a container and its dependencies can be exported from the database and imported into a different database, for, example, when a container needs to be copied from one container to another. For more information about how to export and import containers for copy purposes, see *Related Information* below.

It is also possible to export and import a container for support purposes. However, it is not recommended and is not without risk, as described in the following caution note.

> ⚠ Caution
>
> The API procedures `_SYS_DI#G.EXPORT_CONTAINER_FOR_SUPPORT` and `_SYS_DI#G.IMPORT_CONTAINER_FOR_SUPPORT` are available to the HDI container-group administrator but intended for use by SAP support, exclusively. The exported data might include private or confidential data from the container, and the container import could also compromise the integrity of the database.

Related Information

Maintaining SAP HDI Containers [page 90]
Maintaining the SAP HDI [page 46]
SAP HANA Deployment Infrastructure [page 7]
Export an SAP HDI Container for Copy Purposes [page 87]

4.6.1 Grant Container-Group Administrator Privileges to a User

Container-group administrator privileges can be granted to another user at any time.

Context

Each container group can have its own set of administrators. Administrative privileges for a container group must be explicitly granted by an HDI administrator, or the container group's administrator. Unlike the HDI administrator, the container-group administrator can only grant another user the container group administrator privileges for their own container groups.

This method uses the predefined `_SYS_DI.T_DEFAULT_CONTAINER_GROUP_ADMIN_PRIVILEGES` table, which contains the largest possible set of privileges that can be granted for a user of this type. However, it is possible to reduce the set of privileges granted by specifying the desired set of privileges explicitly and not using this default table.

> → Tip
>
> The procedure `_SYS_DI#G.GRANT_CONTAINER_GROUP_API_PRIVILEGES_WITH_GRANT_OPTION` can also be used; it grants the given privileges “`WITH GRANT OPTION`” to the target user. This procedure is

only needed in special scenarios, for example, when building a custom SQL API by wrapping the HDI SQL API in SQLScript procedures.

Procedure

1. In an SQL console, connect to the database with an HDI administrator user.
2. Open the SQL editor for this database.
3. Insert the following SQL code into the SQL editor:

   In the following code snippet, the name of the HDI container-group is "G", for example, `CALL _SYS_DI#**G**.GRANT_...`. You will have to change this name to reflect the name of the HDI container group in your scenario.

   Sample Code

   ```
   CREATE LOCAL TEMPORARY COLUMN TABLE #PRIVILEGES LIKE
   _SYS_DI.TT_API_PRIVILEGES;
   INSERT INTO #PRIVILEGES (PRINCIPAL_NAME, PRIVILEGE_NAME, OBJECT_NAME)
   SELECT 'OTHER_CONTAINER_GROUP_ADMIN', PRIVILEGE_NAME, OBJECT_NAME FROM
   _SYS_DI.T_DEFAULT_CONTAINER_GROUP_ADMIN_PRIVILEGES;
   CALL _SYS_DI#G.GRANT_CONTAINER_GROUP_API_PRIVILEGES(#PRIVILEGES,
   _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
   DROP TABLE #PRIVILEGES;
   ```

4. Adjust the name of the user `OTHER_CONTAINER_GROUP_ADMIN` in the `INSERT` command in line 2 to reflect the name of the user to whom the API privileges should be granted, and the name of the container group “G” in line 3 to correspond with the name of your container group.
5. Execute the SQL code.
6. (Optional) Confirm that the `OTHER_CONTAINER_GROUP_ADMIN` user is now able to call HDI API procedures in the container group “G”'s API schema `_SYS_DI#G`, where “G” is replaced with the name of your HDI container group.

Related Information

Maintaining SAP HDI Container Groups [page 77]
Revoke SAP HDI Container-Group Administrator Privileges from an Administrator User [page 80]

4.6.2 Revoke SAP HDI Container-Group Administrator Privileges from an Administrator User

In SAP HANA Deployment Infrastructure (HDI), HDI container-group administration privileges can be revoked from a user at any time.

Context

Each container group can have its own set of administrators. Administrative privileges for a container group must be explicitly granted and revoked by an HDI administrator, or a container-group's own administrator. Unlike the HDI administrator, the container-group administrator can only revoke container-group administrator privileges from another user for their own container groups.

→ Tip

This method uses the predefined table `_SYS_DI.T_DEFAULT_CONTAINER_GROUP_ADMIN_PRIVILEGES`, which contains the largest possible set of privileges that can be granted for a user of this type. However, it is possible to reduce the set of privileges granted by specifying the desired set of privileges individually and explicitly instead.

Procedure

1. In an SQL console, connect to the database with an HDI administrator user.
2. Open the SQL editor for this database.
3. Revoke the container-group administrator privileges.

   Insert the following SQL code into the SQL editor:

   Sample Code

   ```
   CREATE LOCAL TEMPORARY COLUMN TABLE #PRIVILEGES LIKE
   _SYS_DI.TT_API_PRIVILEGES;
   INSERT INTO #PRIVILEGES (PRINCIPAL_NAME, PRIVILEGE_NAME, OBJECT_NAME)
   SELECT 'OTHER_CONTAINER_GROUP_ADMIN', PRIVILEGE_NAME, OBJECT_NAME FROM
   _SYS_DI.T_DEFAULT_CONTAINER_GROUP_ADMIN_PRIVILEGES WHERE NOT
   (PRIVILEGE_NAME = 'SELECT' AND OBJECT_NAME LIKE '_SYS_DI.T%');
   CALL _SYS_DI#G.REVOKE_CONTAINER_GROUP_API_PRIVILEGES(#PRIVILEGES,
   _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
   DROP TABLE #PRIVILEGES;
   ```

   a. Adjust the name of the user `OTHER_CONTAINER_GROUP_ADMIN` in the `INSERT` statement in line 2 to reflect the name of the user from whom the API privileges should be revoked, and the name of the container group “G” in the `CALL` statement in line 3 to correspond with the name of your container group.
4. Execute the SQL code.

   Check that the code completes successfully with the HDI return code 0.

5. (Optional) Confirm that the `OTHER_CONTAINER_GROUP_ADMIN` user is no longer able to call HDI API procedures in the container group “G”'s API schema `_SYS_DI#G`, where “G” is replaced with the name of your HDI container group.

Related Information

Maintaining SAP HDI Container Groups [page 77]
Grant Container-Group Administrator Privileges to a User [page 78]

4.6.3 Create an SAP HDI Container

In SAP HANA Deployment Infrastructure (HDI), the HDI container-group administrator can create a new container.

Context

HDI container-group administrators can create HDI containers in any HDI container group for which they are responsible.

Procedure

1. In an SQL console, connect to the database with an HDI administrator user.
2. Open the SQL editor for this database.
3. Create the container.

   Insert the following SQL code into the SQL editor:

   Sample Code

   ```
   CALL _SYS_DI#G.CREATE_CONTAINER('C', _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
   ```
4. Adjust the name of the container group “G” and the name of the new container “C” to suit the names of your container group and container respectively.
5. Execute the SQL code.

   Check that the code completes successfully with the HDI return code 0.
6. (Optional) Confirm that the container exists by checking that a corresponding new entry in the container group schema's `_SYS_DI#G.M_CONTAINERS` view is now present.

Related Information

Maintaining SAP HDI Container Groups [page 77]
Drop an SAP HDI Container [page 82]

4.6.4 Drop an SAP HDI Container

In SAP HANA Deployment Infrastructure (HDI), the HDI container-group administrator can drop a container.

Context

HDI container-group administrators can drop HDI containers from any HDI container group for which they are responsible.

Procedure

1. In an SQL console, connect to the database with an HDI administrator user.
2. Open the SQL editor for this database.
3. Drop the container.

   Insert the following SQL code into the SQL editor:

   Sample Code

   ```
   CALL _SYS_DI#G.DROP_CONTAINER('C', _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
   ```
4. Adjust the name of the container group “G” and the name of the new container “C” to suit the names of your container group and container respectively.
5. Execute the SQL code.

   Check that the code completes successfully with the HDI return code 0.
6. (Optional) Confirm that the container no longer exists by checking that a corresponding entry in the container group schema's `_SYS_DI#G.M_CONTAINERS` view is not present anymore.

Related Information

Maintaining SAP HDI Container Groups [page 77]
Create an SAP HDI Container [page 81]

4.6.5 Grant SAP HDI Container Administrator Privileges to a User

In SAP HANA Deployment Infrastructure (HDI), administrator privileges for HDI containers can be granted to another user at any time.

Context

Each HDI container can have its own set of administrators. Administrative privileges for a container must be explicitly granted or revoked either by an HDI container-group administrator or an HDI-container administrator with the necessary privileges.

> **Note**
>
> This method uses the predefined table `_SYS_DI.T_DEFAULT_CONTAINER_ADMIN_PRIVILEGES`, which contains the largest possible set of privileges that can be granted for a user of this type. You can reduce the set of privileges granted to a user by explicitly specifying the desired set of privileges and not using this default table.

Procedure

1. In an SQL console, connect to the database as the administrator of the target HDI container group (for example, “G”).
2. Open the SQL editor for this database.
3. Grant container-administrator privileges.

   Insert the following SQL code into the editor:

   Sample Code

   ```
   CREATE LOCAL TEMPORARY COLUMN TABLE #PRIVILEGES LIKE
   _SYS_DI.TT_API_PRIVILEGES;
   INSERT INTO #PRIVILEGES (PRINCIPAL_NAME, PRIVILEGE_NAME, OBJECT_NAME)
   SELECT 'NEW_CONTAINER_ADMIN', PRIVILEGE_NAME, OBJECT_NAME FROM
   _SYS_DI.T_DEFAULT_CONTAINER_ADMIN_PRIVILEGES;
   CALL _SYS_DI#G.GRANT_CONTAINER_API_PRIVILEGES('C', #PRIVILEGES,
   _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
   DROP TABLE #PRIVILEGES;
   ```
4. Adjust the name of the user `NEW_CONTAINER_ADMIN` in the `INSERT` statement in line 2 to reflect the name of the user to whom the API privileges should be granted. You will also have to change the name of the container group “G” in the API schema `_SYS_DI#G` and the name of the container “C” to reflect the names in your landscape.
5. Execute the SQL code.

   Confirm that the SQL code completes successfully and displays the HDI return code 0.

6. (Optional) Confirm that the `NEW_CONTAINER_ADMIN` user is now able to call HDI container API procedures in the container C's API schema `C#DI`.

Related Information

Revoke SAP HDI Container Administrator Privileges from a User [page 84]
Maintaining SAP HDI Container Groups [page 77]

4.6.6 Revoke SAP HDI Container Administrator Privileges from a User

In SAP HANA Deployment Infrastructure (HDI), administrator privileges for HDI containers can be revoked from a user at any time.

Context

Each container can have its own set of administrators. Administrative privileges for a container must be explicitly granted or revoked by an HDI container group administrator, or an HDI container administrator with the necessary privileges.

> i Note
>
> This method uses the predefined table `_SYS_DI.T_DEFAULT_CONTAINER_ADMIN_PRIVILEGES`, which contains the largest possible set of privileges that can be granted for a user of this type. You can reduce the set of privileges granted to a user by explicitly specifying the desired set of privileges and not using this default table.

Procedure

1. In an SQL console, connect to the database as the administrator of the HDI container group “G”.
2. Open the SQL editor for this database.
3. Insert the following SQL code into the editor:

Sample Code

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PRIVILEGES LIKE
_SYS_DI.TT_API_PRIVILEGES;
INSERT INTO #PRIVILEGES (PRINCIPAL_NAME, PRIVILEGE_NAME, OBJECT_NAME)
SELECT 'NEW_CONTAINER_ADMIN', PRIVILEGE_NAME, OBJECT_NAME FROM
_SYS_DI.T_DEFAULT_CONTAINER_ADMIN_PRIVILEGES WHERE NOT (PRIVILEGE_NAME =
'SELECT' AND OBJECT_NAME LIKE '_SYS_DI.T%');
```

```
CALL _SYS_DI#G.REVOKE_CONTAINER_API_PRIVILEGES('C', #PRIVILEGES,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PRIVILEGES;
```

4. Adjust the name of the user `NEW_CONTAINER_ADMIN` in the `INSERT` statement in line 2 to reflect the name of the user from whom the API privileges should be revoked. In the `CALL` statement in line 3, you will also have to change the name of the container group "G" in the API schema `_SYS_DI#G` and the name of the container "C" to reflect the names in your landscape.
5. Execute the SQL code.

   Confirm that the SQL code completes successfully and displays the HDI return code 0.
6. (Optional) Confirm that the `NEW_CONTAINER_ADMIN` user is no longer able to call HDI container API procedures in the container C's API schema `C#DI`.

Related Information

Maintaining SAP HDI Container Groups [page 77]
Grant SAP HDI Container Administrator Privileges to a User [page 83]

4.6.7 Grant a Support User Access to an SAP HDI Container

Provide members of the support teams with temporary access to an SAP HDI container.

Context

In the event of container-related problems in SAP HANA Deployment Infrastructure (HDI), it might be necessary for a support user to access HDI-internal objects in a container API schema, for example, `C#DI` for the container "C". These privileges must be temporarily granted to an explicit support user and then revoked when the support task is completed.

⚠ Caution

The use of this function is only recommended in exceptional circumstances; it allows the support user to access all the data in the container, some of which could be private or confidential. Use of this function could also compromise the integrity of the container resulting in an unusable container or data loss. Enabling a support user to access a container raises a security alert to the database administrator.

Procedure

1. In an SQL console, connect to the database with the administrator of the HDI container group G.

2. Open the SQL editor for this database.
3. Insert the following SQL code into the editor:

Sample Code

```
CALL _SYS_DI#G.GRANT_CONTAINER_SUPPORT_PRIVILEGE( 'C', 'SELECT',
'CONTAINER_SUPPORT_USER', _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
```

4. Adjust the schema name of the container group G's API schema `_SYS_DI#G`.
5. Adjust the name of the container “C”.
6. Adjust the privilege as needed.

   Possible values are `SELECT`, `UPDATE`, `INSERT`, and `DELETE`.
7. Adjust the name of the user `CONTAINER_SUPPORT_USER`.
8. Execute the SQL code.

   Confirm that the SQL code completes successfully and displays the HDI return code 0.
9. (Optional) Confirm that the `CONTAINER_SUPPORT_USER` user can now access objects in the container C's API schema `C#DI` per the given privileges.

Related Information

Maintaining SAP HDI Container Groups [page 77]

Revoke Access to an SAP HDI Container from a Support User [page 86]

4.6.8 Revoke Access to an SAP HDI Container from a Support User

Revoke any privileges granted to members of the support teams for temporary access to SAP HDI containers.

Context

In the event of container-related problems in SAP HANA Deployment Infrastructure (HDI), it might be necessary for a support user to access HDI-internal objects in a container API schema, for example, `C#DI` for the container “C”. These privileges must be temporarily granted to an explicit support user and then revoked when the support task is completed.

Procedure

1. In an SQL console, connect to the database with the administrator of the HDI container group G.
2. Open the SQL editor for this database.
3. Insert the following SQL code into the editor:

Sample Code

```
CALL _SYS_DI#G.REVOKE_CONTAINER_SUPPORT_PRIVILEGE( 'C', 'SELECT',
'CONTAINER_SUPPORT_USER', _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
```

   a. Adjust the schema name of the container group G's API schema `_SYS_DI#G`.
   b. Adjust the name of the container “C”.
   c. Adjust the privilege as needed.

      Possible values are `SELECT`, `UPDATE`, `INSERT`, and `DELETE`.
   d. Adjust the name of the user `CONTAINER_SUPPORT_USER`.
4. Execute the SQL code.
5. (Optional) Confirm that the `CONTAINER_SUPPORT_USER` user can no longer access objects in the container C's API schema `C#DI` per the given privileges.

Related Information

Maintaining SAP HDI Container Groups [page 77]
Grant a Support User Access to an SAP HDI Container [page 85]

4.6.9 Export an SAP HDI Container for Copy Purposes

An HDI container and its dependencies can be exported to a table or a file, which can then be used to import the container into another database.

Context

The export of a source container, for example, `C1`, is performed by calling the built-in procedure `_SYS_DI#G.EXPORT_CONTAINER_FOR_COPY`. The procedure expects as inputs the name of the source container, the schema, and the names of two tables: the table into which the export data will be written (`EXPORT_TABLE`), and a table containing any parameters. After the export procedure has completed successfully, the specified table contains not only the objects of the source container's API schema (`C1#DI`) but also the deployed objects of the source container's run-time schema (`C1`), including all dependent data.

To export a container `C1` in container group `G` for copy purposes, perform the following steps:

Procedure

1. Open an SQL console and connect to the SAP HANA database with the permissions of the administrator of the HDI container group `G`.
2. Open the SQL editor for this database.
3. Paste the following SQL code into the SQL editor.

```
CREATE COLUMN TABLE EXPORT_TABLE LIKE _SYS_DI.TT_CONTAINER_EXPORT;
GRANT INSERT ON EXPORT_TABLE TO C1#DI;
GRANT SELECT ON EXPORT_TABLE TO C1#DI;
CALL _SYS_DI#G.EXPORT_CONTAINER_FOR_COPY
  ('C1', CURRENT_SCHEMA, 'EXPORT_TABLE', _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
EXPORT EXPORT_TABLE AS BINARY INTO '<export_path>'
  WITH REPLACE NO DEPENDENCIES;
```

   a. If necessary, modify the name of the container `C1` to the name of the source container you are exporting from.

      i Note

      If the exported container is copied within the same database, the `EXPORT`… statement can be skipped. If the target is another database, the `EXPORT`… statement creates a file on the file system that can be copied to the file system in the target database.

   b. If necessary, change the name of the target table ("`EXPORT_TABLE`" in the example above) to the name of the table that should receive the exported container. A check is performed to establish if `EXPORT_TABLE` exists. If it exists, has the correct format, and is empty, then the export operation runs. Otherwise, an error is returned.

      ! Restriction

      The target table cannot be a temporary table.

4. Execute the SQL code, and check that the operation completed successfully (HDI return code 0).
5. Confirm that the container has been exported to the `EXPORT_TABLE` (or to the specified target file).

Related Information

Maintaining SAP HDI Container Groups [page 77]
Import an SAP HDI Container for Copy Purposes [page 89]

4.6.10 Import an SAP HDI Container for Copy Purposes

An HDI container and its dependencies can be imported into the same (or another) database from a table or file.

Context

An HDI container and its dependencies can be exported to a table or a file and then imported to a new container. The import of a source container, for example, `C1` into a new container `C2` is performed by calling the built-in procedure `_SYS_DI#G.IMPORT_CONTAINER_FOR_COPY`. This procedure expects as input the name of an existing empty target container `C2`, the schema and table names of the table containing the exported results, and a table containing any necessary parameters.

The import procedure removes the contents of the target container, writes the data from the export table to the target container, and runs a make. Any existing objects and data in the target container will be deleted before the import procedure starts. After the import procedure has completed successfully, the target container will be a copy of the source container including all the data of the original run-time schema.

> **i Note**
>
> Only objects deployed by the source container's `MAKE` procedure are copied. Objects created manually in the source container's run-time schema are not copied.

To import a container `C1` into a new container `C2` for copy purposes, perform the following steps:

> **i Note**
>
> The container version of the target container must be the same as the container version of the source container. Containers with version greater than or equal to (>=) 40 can also be imported into containers with a higher container version. You can check the container version with the view `_SYS_DI#G.M_CONTAINER_VERSIONS`.

Procedure

1. Open an SQL console and connect to the SAP HANA database with the permissions of the administrator of the HDI container group `G`.
2. Open the SQL editor for this database.
3. If needed, create the new target container, for example, `C2`.

   ```
   CALL _SYS_DI#G.CREATE_CONTAINER('C2', _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
   ```

   > **i Note**
   >
   > The new container does not have to be empty for the purpose of an import. However, any existing objects and data in the target container will be deleted before the import.

4. If necessary, grant the object owner of the target container `C2#OO` any required privileges for all the external objects required by the container to be imported.

   > **i Note**
   >
   > All external objects referenced by the container's design-time objects must be available and accessible to the object owner of the target container.

5. Paste the following SQL code into the SQL editor.

   ```
   IMPORT EXPORT_TABLE FROM '<export_path>';
   CALL _SYS_DI#G.IMPORT_CONTAINER_FOR_COPY
     ('C2', CURRENT_SCHEMA, 'EXPORT_TABLE', _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
   ```

   > **i Note**
   >
   > If the imported container is copied from within the same database, the `IMPORT...` statement can be skipped.

6. Execute the SQL code, and check that the operation completed successfully (HDI return code 0).
7. Confirm that the container has been successfully imported.

   In this example, you can check that the expected run-time objects of source container `C1` are also present in the target container `C2`.

Related Information

Maintaining SAP HDI Container Groups [page 77]
Export an SAP HDI Container for Copy Purposes [page 87]

4.7 Maintaining SAP HDI Containers

An HDI container administrator configures and controls access to a SAP HDI container.

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 SAP HANA platform database features which describe the target (run-time) state of SAP HANA database artifacts, for example: tables, views, or procedures. These artifacts are modeled, staged (uploaded), built, and deployed into SAP HANA.

The SAP HANA service broker is used to create and destroy HDI containers; each HDI container comprises a design-time container (DTC), which is an isolated environment used to store design-time files, and a run-time container (RTC), which is used to store deployed objects built according to the specification stored in the corresponding design-time artifacts.

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.

The HDI container administrator manages one or more containers assigned by the container-group administrator. The role of the container-manager focuses primarily on configuring and controlling access to the HDI containers used to store the database objects deployed by the SAP HANA Deployment Infrastructure deploy service and repairing any problems that occur with run-time objects in the assigned HDI containers. An HDI container administrator can manage one or more containers in one HDI container group or multiple containers distributed across multiple container groups.

The configuration of HDI containers also 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 plug-in (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 run-time locations.

→ Tip

The APIs of a container named “C” are in the schema `C#DI`.

HDI-Container Maintenance Tasks

To manage an HDI container group, the administrator performs the following common tasks:

- Grant and revoke container administrator privileges
- Grant and revoke access to the container development API
- Grant and revoke access to a container's schema
- Grant and revoke a user role from the container's schema
- List all currently configured build plug-in libraries available to a container
- Configure the default set of build plug-in libraries available to a container
- Configure a custom set of build plug-in libraries available to a container
- Configure container parameters

Related Information

Maintaining SAP HDI Container Groups [page 77]
Maintaining the SAP HDI [page 46]
SAP HANA Deployment Infrastructure [page 7]

4.7.1 Grant SAP HDI Container Administrator Privileges to a User

Enable administrator access to an SAP HDI container.

Context

In SAP HANA Deployment Infrastructure (HDI), administrator privileges for an HDI container are initially granted to a user by an administrator of the container group that the HDI container belongs to. If these privileges have been granted "with grant option", the HDI container administrator can also grant these privileges to another user as described below:

> **i Note**
>
> The predefined table `_SYS_DI.T_DEFAULT_CONTAINER_ADMIN_PRIVILEGES` used in this task contains the largest possible set of privileges that can be granted for a user of this type. You can reduce the set of privileges granted by explicitly specifying the desired set of privileges and not using this default table.

Procedure

1. In an SQL console, connect to the database with an administrator of the HDI container “C”.
2. Open the SQL editor for this database.
3. Insert the following SQL code into the editor:

   Sample Code

   ```
   CREATE LOCAL TEMPORARY COLUMN TABLE #PRIVILEGES LIKE
   _SYS_DI.TT_API_PRIVILEGES;
   INSERT INTO #PRIVILEGES (PRINCIPAL_NAME, PRIVILEGE_NAME, OBJECT_NAME)
   SELECT 'NEW_CONTAINER_ADMIN', PRIVILEGE_NAME, OBJECT_NAME FROM
   _SYS_DI.T_DEFAULT_CONTAINER_ADMIN_PRIVILEGES;
   CALL C#DI.GRANT_CONTAINER_API_PRIVILEGES(#PRIVILEGES,
   _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
   DROP TABLE #PRIVILEGES;
   ```

   a. Adjust the name of the user “`NEW_CONTAINER_ADMIN`” in line 2 to reflect the name of the user to whom the API privileges should be granted.
   b. Adjust the name of the container's API schema `C#DI` to suit your needs.
   c. If the target user should also be able to grant another user the container administration privileges, replace the procedure `C#DI.GRANT_CONTAINER_API_PRIVILEGES` in the SQL statement above with the procedure `C#DI.GRANT_CONTAINER_API_PRIVILEGES_WITH_GRANT_OPTION`.

> i Note
>
> The procedure `C#DI.GRANT_CONTAINER_API_PRIVILEGES_WITH_GRANT_OPTION` should only be used in special scenarios, for example, when building a custom SQL API by wrapping the HDI SQL API in SQLScript procedures.

4. Execute the SQL code.

   Confirm that the SQL code completes successfully and displays the HDI return code 0.
5. (Optional) Confirm that the `NEW_CONTAINER_ADMIN` user is now able to call HDI container API procedures in the containers API schema (for example, `C#DI` in container “C”).

Related Information

Revoke SAP HDI Container Administrator Privileges from a User [page 93]
Maintaining SAP HDI Containers [page 90]

4.7.2 Revoke SAP HDI Container Administrator Privileges from a User

Disable administrator access to an SAP HDI container.

Context

In SAP HANA Deployment Infrastructure (HDI), each HDI container can have its own set of administrators, and administrative privileges for an HDI container must be explicitly granted or revoked either by an HDI container-group administrator or by an HDI container administrator with the necessary privileges.

> i Note
>
> The predefined table `_SYS_DI.T_DEFAULT_CONTAINER_ADMIN_PRIVILEGES` used in this task contains the largest possible set of privileges that can be granted for a user of this type. You can reduce the set of privileges granted by explicitly specifying the desired set of privileges and not using this default table.

Procedure

1. In an SQL console, connect to the database with an administrator of the HDI container “C” (or the name of the container whose administrator privileges you want to change).
2. Open the SQL editor for this database.

3. Revoke container-administrator privileges from a user.

   Insert the following SQL code into the SQL editor:

   Sample Code

   ```
   CREATE LOCAL TEMPORARY COLUMN TABLE #PRIVILEGES LIKE
   _SYS_DI.TT_API_PRIVILEGES;
   INSERT INTO #PRIVILEGES (PRINCIPAL_NAME, PRIVILEGE_NAME, OBJECT_NAME)
   SELECT 'NEW_CONTAINER_ADMIN', PRIVILEGE_NAME, OBJECT_NAME FROM
   _SYS_DI.T_DEFAULT_CONTAINER_ADMIN_PRIVILEGES WHERE NOT (PRIVILEGE_NAME =
   'SELECT' AND OBJECT_NAME LIKE '_SYS_DI.T%');
   CALL C#DI.REVOKE_CONTAINER_API_PRIVILEGES(#PRIVILEGES,
   _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
   DROP TABLE #PRIVILEGES;
   ```

   a. In the `INSERT` statement in line 2, adjust the name of the user “`NEW_CONTAINER_ADMIN`” to reflect the name of the user from whom the API privileges should be revoked.
   b. In the `CALL` statement in line 3, adjust the name of the container's API schema `C#DI` to suit your needs.
4. Execute the SQL code.

   Confirm that the SQL code completes successfully and displays the HDI return code 0.
5. (Optional) Confirm that the `NEW_CONTAINER_ADMIN` user is no longer able to call HDI container API procedures in the containers API schema (for example, `C#DI` in container “C”).

Related Information

Maintaining SAP HDI Containers [page 90]
Grant SAP HDI Container Administrator Privileges to a User [page 92]

4.7.3 Grant Access to the SAP HDI Container Content-Development API

Enable access to the content-development application programming interface (API) for an SAP HDI container.

Context

In SAP HANA Deployment Infrastructure (HDI), the content-development API in a container's `#DI` schema (for example, schema `C#DI` in container “C”) is intended for use by developers of HDI database artifacts. Developers use the container content-development API to write design-time artifacts to the container; the design-time artifacts are then used to generate database objects in the container's schema (for example, schema `C` in container “C”). A container administrator must first grant the necessary privileges to the developer before the API can be used, as described below.

i Note

The predefined table `_SYS_DI.T_DEFAULT_CONTAINER_USER_PRIVILEGES` used in this task contains the largest possible set of privileges that can be granted for a user of this type. You can reduce the scope of the privileges granted by not using this default table and explicitly specifying the desired set of privileges instead.

Procedure

1. In an SQL console, connect to the database with an administrator of the target HDI container (for example, “C”).
2. Open the SQL editor for this database.
3. Grant access to the HDI container content-development API.

   Insert the following SQL code into the editor:

   Sample Code

   ```
   CREATE LOCAL TEMPORARY COLUMN TABLE #PRIVILEGES LIKE
   _SYS_DI.TT_API_PRIVILEGES;
   INSERT INTO #PRIVILEGES (PRINCIPAL_NAME, PRIVILEGE_NAME, OBJECT_NAME)
   SELECT 'NEW_CONTAINER_CONTENT_DEVELOPER', PRIVILEGE_NAME, OBJECT_NAME FROM
   _SYS_DI.T_DEFAULT_CONTAINER_USER_PRIVILEGES;
   CALL C#DI.GRANT_CONTAINER_API_PRIVILEGES(#PRIVILEGES,
   _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
   DROP TABLE #PRIVILEGES;
   ```

   a. Adjust the name of the new content-development user “`NEW_CONTAINER_CONTENT_DEVELOPER`” in the `INSERT` statement in line 2 to reflect the name of the user to whom the API privileges should be granted.
   b. Adjust the name of the container's API schema `C#DI` in the `CALL` statement to suit your needs.
4. Execute the SQL code.

   Confirm that the SQL code completes successfully and displays the HDI return code 0.
5. (Optional) Confirm that the `NEW_CONTAINER_CONTENT_DEVELOPER` user is now able to call the HDI container content-development API in the containers API schema (for example, `C#DI` in container “C”).

Related Information

Maintaining SAP HDI Containers [page 90]
Revoke Access to the SAP HDI Container Content-Development API [page 96]

4.7.4 Revoke Access to the SAP HDI Container Content-Development API

Disable access to the content-development application programming interface (API) for an SAP HDI container.

Context

In SAP HANA Deployment Infrastructure (HDI), the content-development API in an HDI container's schema (for example, schema C#DI in container “C”) is intended for use by developers of HDI database artifacts. Developers use the container content-development API to write design-time artifacts to the container; the design-time artifacts are then used to generate database objects in the container's schema (for example, schema C#DI in container “C”. A container administrator can revoke the privileges that enable access to the container content-development API at any time, as described below.

> **Note**
>
> The predefined table _SYS_DI.T_DEFAULT_CONTAINER_USER_PRIVILEGES used in this task contains the largest possible set of privileges that can be granted for a user of this type. You can reduce the scope of the privileges granted by not using this default table and explicitly specifying the desired set of privileges instead.

Procedure

1. In an SQL console, connect to the database as an administrator of the HDI container whose development API you want to enable (for example, “C”).
2. Open the SQL editor for this database.
3. Revoke access to the HDI container content-development API.

   Insert the following SQL code into the editor:

   Sample Code

   ```
   CREATE LOCAL TEMPORARY COLUMN TABLE #PRIVILEGES LIKE
   _SYS_DI.TT_API_PRIVILEGES;
   INSERT INTO #PRIVILEGES (PRINCIPAL_NAME, PRIVILEGE_NAME, OBJECT_NAME)
   SELECT 'NEW_CONTAINER_CONTENT_DEVELOPER', PRIVILEGE_NAME, OBJECT_NAME FROM
   _SYS_DI.T_DEFAULT_CONTAINER_USER_PRIVILEGES WHERE NOT (PRIVILEGE_NAME =
   'SELECT' AND OBJECT_NAME LIKE '_SYS_DI.T%');
   CALL C#DI.REVOKE_CONTAINER_API_PRIVILEGES(#PRIVILEGES,
   _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
   DROP TABLE #PRIVILEGES;
   ```

   a. In the INSERT statement in line 2, adjust the name of the new content-development user “NEW_CONTAINER_CONTENT_DEVELOPER” to reflect the name of the user from whom the API privileges should be revoked.

b. In the CALL statement in line 3, adjust the name of the container's API schema C#DI to suit your needs.

4. Execute the SQL code.

   Confirm that the SQL code completes successfully and displays the HDI return code 0.
5. (Optional) Confirm that the NEW_CONTAINER_CONTENT_DEVELOPER user is no longer able to call the HDI container content-development API in the containers API schema (for example, C#DI in container “C”).

Related Information

Maintaining SAP HDI Containers [page 90]
Grant Access to the SAP HDI Container Content-Development API [page 94]

4.7.5 Grant Access to an SAP HDI Container's Schema

Enable access to the schema of an SAP HDI container of individual objects in the target schema.

Context

In SAP HANA Deployment Infrastructure (HDI), users that would like to consume objects deployed to an HDI container need to be granted the appropriate privileges. The privileges can be granted to specific objects in the schema (for example, C) by use of a role that has been deployed to the target container, for example, “C”, or by granting privileges for the entire schema.

To grant access privileges for the entire container schema where the database objects are located to a database object consumer NEW_CONTAINER_CONSUMER, perform the following steps:

Procedure

1. In an SQL console, connect to the database with an administrator of the target HDI container (for example, “C”).
2. Open the SQL editor for this database.
3. Grant access to the HDI container's schema.

   Insert the following SQL code into the editor:

   Sample Code

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PRIVILEGES LIKE
_SYS_DI.TT_SCHEMA_PRIVILEGES;
INSERT INTO #PRIVILEGES ( PRIVILEGE_NAME, PRINCIPAL_SCHEMA_NAME,
PRINCIPAL_NAME ) VALUES ( 'SELECT', '', 'NEW_CONTAINER_CONSUMER' );
```

```
CALL C#DI.GRANT_CONTAINER_SCHEMA_PRIVILEGES( #PRIVILEGES,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PRIVILEGES;
```

a. Adjust the name of the consumer user “`NEW_CONTAINER_CONSUMER`” in the `INSERT` statement in the `INSERT`statement in line 2 to reflect the name of the user who requires access to the container.
b. Adjust the name of the container's API schema (`C#DI`) in the `CALL` statement to suit your needs.

4. Execute the SQL code.

   Confirm that the SQL code completes successfully and displays the HDI return code 0.
5. (Optional) Confirm that the `NEW_CONTAINER_CONSUMER` can now access the database objects in the container's schema.

Related Information

Maintaining SAP HDI Containers [page 90]
Revoke Access to an SAP HDI Container's Schema [page 98]
SAP HDI Container Schemas [page 48]

4.7.6 Revoke Access to an SAP HDI Container's Schema

Revoke access to the schema of an SAP HDI container or specific objects in the target schema.

Context

In SAP HANA Deployment Infrastructure (HDI), users that need to consume objects deployed in a container (for example, “C”) must be granted the appropriate privileges to access the target container's schema (for example, `C#DI`). The privileges for the entire schema can be revoked with the `REVOKE_CONTAINER_SCHEMA_PRIVILEGES` API.

To revoke privileges for the entire container schema where the database objects are located from a database object consumer `NEW_CONTAINER_CONSUMER` perform the following steps:

Procedure

1. In an SQL console, connect to the database with an administrator of the target HDI container (for example, “C”).
2. Open the SQL editor for this database.
3. Revoke access to the HDI container's schema.

   Insert the following SQL code into the editor:

Sample Code

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PRIVILEGES LIKE
_SYS_DI.TT_SCHEMA_PRIVILEGES;
INSERT INTO #PRIVILEGES ( PRIVILEGE_NAME, PRINCIPAL_SCHEMA_NAME,
PRINCIPAL_NAME ) VALUES ( 'SELECT', '', 'NEW_CONTAINER_CONSUMER' );
CALL C#DI.REVOKE_CONTAINER_SCHEMA_PRIVILEGES( #PRIVILEGES,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PRIVILEGES;
```

  a. In the `INSERT` statement in line 2, adjust the name of the consumer user "`NEW_CONTAINER_CONSUMER`" to reflect the name of the user whose container-access privileges must be revoked.
  b. In the `CALL` statement, adjust the name of the container's API schema (`C#DI`) to suit your needs.
4. Execute the SQL code.

   Confirm that the SQL code completes successfully and displays the HDI return code 0.
5. (Optional) Confirm that the `NEW_CONTAINER_CONSUMER` can no longer access the database objects in the container's schema.

Related Information

Maintaining SAP HDI Containers [page 90]
Grant Access to an SAP HDI Container's Schema [page 97]
SAP HDI Container Schemas [page 48]

4.7.7 Grant a User a Role from the SAP HDI Container's Schema

Use a role to enable access to an SAP HDI container's schema objects.

Context

In SAP HANA Deployment Infrastructure (HDI), users who would like to consume objects deployed to an SAP HDI container (for example, named "C") need to be granted the appropriate access privileges for the container's schema. The access privileges can be granted to specific objects in the schema (for example, `C#DI`) by use of a role that has been deployed to the container, or by granting privileges for the entire schema.

To grant privileges for specific objects in the container to a user `NEW_CONTAINER_CONSUMER` by use of a role in the target container, perform the following steps:

Procedure

1. In an SQL console, connect to the database with an administrator of the target HDI container (for example, “C”).
2. Open the SQL editor for this database.
3. Grant the role to the target user.

   Insert the following SQL code into the editor:

   > **i Note**
   >
   > The temporary table `#ROLES` must be placed in a schema where the current user has the privileges required to create a temporary table.

   > **Sample Code**
   >
   > ```
   > CREATE LOCAL TEMPORARY COLUMN TABLE #ROLES LIKE _SYS_DI.TT_SCHEMA_ROLES;
   > INSERT INTO #ROLES ( ROLE_NAME, PRINCIPAL_SCHEMA_NAME, PRINCIPAL_NAME )
   > VALUES ( 'myrole', '', 'NEW_CONTAINER_CONSUMER' );
   > CALL C#DI.GRANT_CONTAINER_SCHEMA_ROLES(.#ROLES,
   > _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
   > DROP TABLE .#ROLES;
   > ```

   a. Adjust the name of the consumer user `NEW_CONTAINER_CONSUMER` in the `INSERT` statement in line 2
   b. Adjust the name of the role "`myrole`" in the `INSERT` statement in line 2.
   c. Adjust the schema name of the container C's API schema `C#DI` in the `CALL` statement in line 3.
4. Execute the SQL code.

   Confirm that the SQL code completes successfully and displays the HDI return code 0.
5. (Optional) Confirm that the `NEW_CONTAINER_CONSUMER` can now access the database objects specified by the role in the container's schema.

Related Information

Maintaining SAP HDI Containers [page 90]
Revoke a Role from the SAP HDI Container's Schema [page 101]

4.7.8 Revoke a Role from the SAP HDI Container's Schema

Use a role to disable access to an SAP HDI container's schema objects.

Context

In SAP HANA Deployment Infrastructure (HDI), users who would like to consume objects deployed to an SAP HDI container (for example, named “C”) need to be granted the appropriate access privileges for the container's schema. The privileges for specific objects in the target schema (for example, `C#DI`), granted by a role in the schema, can be revoked with the `REVOKE_CONTAINER_SCHEMA_ROLES` API:

To revoke a role and disable access privileges for specific objects in the container from a user, perform the following steps:

Procedure

1. In an SQL console, connect to the database with an administrator of the target HDI container (for example, “C”).
2. Open the SQL editor for this database.
3. Grant the role to the target user.

   Insert the following SQL code into the editor:

   Sample Code

   ```
   CREATE LOCAL TEMPORARY COLUMN TABLE #ROLES LIKE _SYS_DI.TT_SCHEMA_ROLES;
   INSERT INTO #ROLES ( ROLE_NAME, PRINCIPAL_SCHEMA_NAME, PRINCIPAL_NAME )
   VALUES ( 'myrole', '', 'NEW_CONTAINER_CONSUMER' );
   CALL C#DI.REVOKE_CONTAINER_SCHEMA_ROLES(#ROLES,
   _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
   DROP TABLE #ROLES;
   ```

   a. In the `INSERT` statement in line 2, adjust the name of the consumer user `NEW_CONTAINER_CONSUMER` and the role "`myrole`" to suit your needs.
   b. In the `CALL` statement in line 3, adjust the schema name of the container “C”'s API schema C#DI.
4. Execute the SQL code.

   Confirm that the SQL code completes successfully and displays the HDI return code 0.
5. (Optional) Confirm that the `NEW_CONTAINER_CONSUMER` can no longer access the database objects specified by the role in the container's schema.

Related Information

Maintaining SAP HDI Containers [page 90]

[Grant a User a Role from the SAP HDI Container's Schema [page 99]]

4.7.9 List All Currently Configured Build Plug-in Libraries Available to an SAP HDI Container

Display a list of all the build plug-in libraries available for use in an SAP HDI container.

Context

In SAP HANA Deployment Infrastructure (HDI), administrators of an HDI container (for example, container "C") can view all currently configured HDI build plug-in libraries and their versions for any container where they have the container-administration permissions, as described in the following procedure:

Procedure

1. In an SQL console, connect to the database with an administrator of the target HDI container, for example, "C".
2. Open the SQL editor for this database.
3. Display a list of all currently available plug-ins in the specified container.

   Insert the following SQL statement into the SQL editor:

   Sample Code

   ```
   CALL C#DI.LIST_CONFIGURED_LIBRARIES(_SYS_DI.T_NO_PARAMETERS, ?, ?, ?, ?);
   ```

   a. Replace the name of the container "C" in the `CALL` statement with the name of your container.
4. Execute the SQL code.

   Confirm that the SQL code completes successfully and displays the HDI return code 0.
5. A result set with the requested plug-in information is returned.

   The following example shows an excerpt from the result set returned:

   Output Code

   ```
   LIBRARY_NAME;LIBRARY_VERSION;PLUGIN_ID;PLUGIN_VERSION
   com.sap.hana.di.afllangprocedure;0.0;com.sap.hana.di.afllangprocedure;
   2.0.10.0
   com.sap.hana.di.analyticprivilege;0.0;com.sap.hana.di.analyticprivilege;
   2.0.10.0
   com.sap.hana.di.calculationview;0.0;com.sap.hana.di.calculationview;
   2.0.10.0
   com.sap.hana.di.cds;0.3;com.sap.hana.di.cds;2.0.10.0
   com.sap.hana.di.constraint;0.0;com.sap.hana.di.constraint;2.0.10.0
   com.sap.hana.di.copyonly;0.0;com.sap.hana.di.copyonly;2.0.10.0
   ```

```
com.sap.hana.di.dropcreatetable;0.0;com.sap.hana.di.dropcreatetable;
2.0.10.0
com.sap.hana.di.flowgraph;0.0;com.sap.hana.di.flowgraph;2.0.10.0
```

Related Information

Configure the Default Build Plug-in Libraries Available to an SAP HDI Container [page 103]
Configure a Custom Set of Build Plug-in Libraries Available to an SAP HDI Container [page 104]
Maintaining SAP HDI Containers [page 90]

4.7.10 Configure the Default Build Plug-in Libraries Available to an SAP HDI Container

Maintain the set of plug-in libraries available by default in an SAP HDI container.

Context

In SAP HANA Deployment Infrastructure (HDI), HDI container administrators can configure a default set of commonly used SAP HDI plug-in libraries for any HDI container for which they have responsibility.

→ Tip

The table `_SYS_DI.T_DEFAULT_LIBRARIES` contains the names of a default set of plug-in libraries used for HDI artifacts. The contents of the table are passed to the configure-libraries procedure (`DI.CONFIGURE_LIBRARIES`) as input. The content of `_SYS_DI.T_DEFAULT_LIBRARIES` is not modified

Procedure

1. In an SQL console, connect to the database as an administrator of the target HDI container, for example, “C”.
2. Open the SQL editor for this database.
3. Set the list of plug-in libraries available by default in the specified target container.

   Insert the following SQL statement into the SQL editor:

   Sample Code

   ```
   CALL C#DI.CONFIGURE_LIBRARIES(_SYS_DI.T_DEFAULT_LIBRARIES,
   _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
   ```

a. Replace the name of the container “C” in the `CALL` statement with the name of your container.

4. Execute the SQL code.

   Confirm that the SQL code completes successfully and displays the HDI return code 0.
5. (**Optional**) Confirm that the default libraries have been configured successfully by listing all currently configured build plug-in libraries available to the container.

Related Information

List All Currently Configured Build Plug-in Libraries Available to an SAP HDI Container [page 102]
Configure a Custom Set of Build Plug-in Libraries Available to an SAP HDI Container [page 104]
Maintaining SAP HDI Containers [page 90]

4.7.11 Configure a Custom Set of Build Plug-in Libraries Available to an SAP HDI Container

Maintain a custom set of plug-in libraries available in an SAP HDI container.

Context

In SAP HANA Deployment Infrastructure (HDI), HDI container administrators can configure a custom set of commonly used HDI plug-in libraries for any container for which they are responsible. It is possible to define the full set of libraries, or incrementally add or remove certain libraries to the currently configured set.

Procedure

1. In an SQL console, connect to the database as an administrator of the target HDI container, for example, “C”.
2. Open the SQL editor for this database.
3. Define a custom list of plug-in libraries available in the specified target container.

   Insert the following SQL statement into the SQL editor:

   Sample Code

   ```
   CREATE LOCAL TEMPORARY COLUMN TABLE #LIBRARY_CONFIGURATION LIKE
   _SYS_DI.TT_LIBRARY_CONFIGURATION;
   INSERT INTO #LIBRARY_CONFIGURATION ( ACTION, LIBRARY_NAME ) VALUES
   ( 'ADD', 'com.sap.hana.di.calculationview' );
   INSERT INTO #LIBRARY_CONFIGURATION ( ACTION, LIBRARY_NAME ) VALUES
   ( 'ADD', 'com.sap.hana.di.cds' );
   ```

```
INSERT INTO #LIBRARY_CONFIGURATION ( ACTION, LIBRARY_NAME ) VALUES
( 'ADD', 'com.sap.hana.di.synonym' );
INSERT INTO #LIBRARY_CONFIGURATION ( ACTION, LIBRARY_NAME ) VALUES
( 'REMOVE', 'com.sap.hana.di.view' );
CALL C#DI.CONFIGURE_LIBRARIES(#LIBRARY_CONFIGURATION,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #LIBRARY_CONFIGURATION;
```

   a. Replace the name of the container “C” in the `CALL` statement with the name of your container.
   b. Adjust the set of libraries to be added or removed in the `INSERT` statements as needed. This example adds three libraries and removes one.
4. Execute the SQL code.

   Confirm that the SQL code completes successfully and displays the HDI return code 0.
5. (**Optional**) Confirm that the custom set of libraries has been configured successfully by listing all currently configured build plug-in libraries available in the container.
6. (**Optional**) Customize the configuration of the build plug-in library if necessary.

   The behavior of the build plug-ins library configuration can be modified by supplying custom parameters in the call to `CONFIGURE_LIBRARIES` instead of `_SYS_DI.T_NO_PARAMETERS`.

   For example, to remove a build plug-in library and **undeploy** all files `('undeploy', 'true')` corresponding to that library, use the following SQL code:

i Note

The default value for the `undeploy` parameter is `'false'`

Sample Code

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PARAMETERS LIKE _SYS_DI.TT_PARAMETERS;
INSERT INTO #PARAMETERS (KEY, VALUE) VALUES ('undeploy', 'true');
CREATE LOCAL TEMPORARY COLUMN TABLE #LIBRARY_CONFIGURATION LIKE
_SYS_DI.TT_LIBRARY_CONFIGURATION;
INSERT INTO #LIBRARY_CONFIGURATION ( ACTION, LIBRARY_NAME ) VALUES
( 'REMOVE', 'com.sap.hana.di.view' );
CALL C#DI.CONFIGURE_LIBRARIES(#LIBRARY_CONFIGURATION,
#PARAMETERS, ?, ?, ?);
DROP TABLE #LIBRARY_CONFIGURATION;
DROP TABLE #PARAMETERS;
```

Related Information

List All Currently Configured Build Plug-in Libraries Available to an SAP HDI Container [page 102]
Configure the Default Build Plug-in Libraries Available to an SAP HDI Container [page 103]
Maintaining SAP HDI Containers [page 90]

4.7.12 Configure SAP HDI Container Parameters

Maintain the parameters used to configure an SAP HDI container.

Context

In SAP HANA Deployment Infrastructure (HDI), the HDI container administrator can use configuration parameters to maintain some of the more general aspects of an HDI container. Container-specific configuration parameters are used to control the behavior of a single container, for example, they can be used to specify the time a container operation waits for a locking conflict to clear, or the maximum number of parallel jobs to be spawned during a make operation.

> → Tip
>
> For more information about HDI configuration parameters, see *Related Information*.

To configure container-specific parameters, perform the following steps:

Procedure

1. In an SQL console, connect to the database with an administrator of the target HDI container (for example, “C”).
2. Open the SQL editor for this database.
3. Define and set the parameters required for the configuration of the target HDI container.

   Sample Code

   ```
   CREATE LOCAL TEMPORARY COLUMN TABLE #CONFIG_PARAMETERS LIKE
   _SYS_DI.TT_PARAMETERS;
   INSERT INTO #CONFIG_PARAMETERS (KEY, VALUE) VALUES
   ('make.max_parallel_jobs', '8');
   CALL C#DI.CONFIGURE_CONTAINER_PARAMETERS(#CONFIG_PARAMETERS,
   _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
   DROP TABLE #CONFIG_PARAMETERS;
   ```

   a. Add more parameters and values as required.

   > → Tip
   >
   > A new `INSERT` statement is required for each new parameter.

   ```
   INSERT INTO #CONFIG_PARAMETERS (KEY, VALUE) VALUES
   ('make.max_parallel_jobs', '8');
   INSERT INTO #CONFIG_PARAMETERS (KEY, VALUE) VALUES
   ('connection.transaction_lock_wait_timeout', '1,000');
   INSERT INTO #CONFIG_PARAMETERS (KEY, VALUE) VALUES
   ('messages.days_to_keep', '30');
   ...
   ```

b. Adjust the schema name of the container (“C”) in the `CALL` statement.

4. Execute the SQL code.

   Confirm that the SQL code completes successfully and displays the HDI return code 0.

Related Information

SAP HDI Parameters [page 50]
SAP HDI Configuration Parameters [page 66]

4.7.13 Cancel a Running Make Operation in an SAP HDI Container

An HDI container administrator can cancel a `make` operation that is running in an SAP HDI container.

Context

In SAP HANA Deployment Infrastructure (HDI), an HDI container administrator can cancel a running make in a container `C`, for example, if it is taking longer to complete than expected. This is mainly intended for canceling asynchronous makes (`C#DI.MAKE_ASYNC`) but also works for synchronous ones (`C#DI.MAKE`). When an asynchronous make is started, it returns a request ID identifying that make process. This request ID can then be used in a call to the `C#DI.CANCEL` procedure for canceling this process.

Procedure

1. In an SQL console, connect to the database as the administrator of the target HDI container (for example, “C”).
2. Open the SQL editor for this database.
3. Cancel the make operation.

   Enter the following SQL code:

   Sample Code

```
CALL C#DI.CANCEL(12345, _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
```

   a. Adjust the name of the container (“C”) in the `CALL` statement to reflect your target container.
   b. Replace “`12345`” in the `CALL` statement with the request ID of the make process you want to cancel.

> → Tip
>
> The procedure `C#DI.MAKE` returns one or more result sets, which contain the request ID and the return code.

4. Execute the SQL code.

   Confirm that the SQL code completes successfully and displays the HDI return code 0.

Related Information

Maintaining SAP HDI Containers [page 90]

4.7.14 Export an SAP HDI Container for Copy Purposes

An HDI container administrator can export an HDI container and its dependencies to a table or a file, which can then be used to import the container into another database.

Context

The export of a source HDI container, for example, `C1`, is performed by calling the built-in procedure `C1#DI.EXPORT_CONTAINER_FOR_COPY`. The procedure expects as input the schema, and the names of two tables used for the export operation: the table into which the export data will be written (`EXPORT_TABLE`), and a table containing any parameters. After the export procedure has completed successfully, the specified table contains not only the objects of the source container's API schema (`C1#DI`) but also the deployed objects of the source container's run-time schema (`C1`), including all dependent data.

To export a container `C1` for copy purposes, perform the following steps:

Procedure

1. Open an SQL console and connect to the SAP HANA database with the permissions of the administrator of the HDI container "C1".
2. Open the SQL editor for this database.
3. Paste the following SQL code into the SQL editor.

```
CREATE COLUMN TABLE EXPORT_TABLE LIKE _SYS_DI.TT_CONTAINER_EXPORT;
GRANT INSERT ON EXPORT_TABLE TO C1#DI;
GRANT SELECT ON EXPORT_TABLE TO C1#DI;
CALL C1#DI.EXPORT_CONTAINER_FOR_COPY
  (CURRENT_SCHEMA, 'EXPORT_TABLE', _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
EXPORT EXPORT_TABLE AS BINARY INTO '<export_path>'
```

```
WITH REPLACE NO DEPENDENCIES;
```

a. If necessary, modify the name of the container `C1` to the name of the source container you are exporting from.

> **i Note**
>
> If the exported container is copied within the same database, the `EXPORT`… statement can be skipped. If the target is another database, the `EXPORT`… statement creates a file on the file system that can be copied to the file system in the target database.

b. If necessary, change the name of the target table (“`EXPORT_TABLE`” in the example above) to the name of the table that should receive the exported container.

> **i Note**
>
> The target table cannot be a temporary table.

c. The `EXPORT`… statement can be skipped if the container should be copied in the same database. If the target is another database, the `EXPORT`… statement creates a file on the file system which can be copied to the target database file system.

4. Execute the SQL code, and check that the operation completed successfully (HDI return code 0).
5. Confirm that the container has been exported to the `EXPORT_TABLE` (or to the specified target file).

Related Information

Maintaining SAP HDI Container Groups [page 77]
Import an SAP HDI Container for Copy Purposes [page 109]

4.7.15 Import an SAP HDI Container for Copy Purposes

An HDI container administrator can import an HDI container and its dependencies into the same (or another) database from a table or file.

Context

An HDI container and its dependencies can be exported to a table or a file and then imported to a new container. The import of a source container, for example, `C1` into a new container `C2` is performed by calling the built-in procedure `C2#DI.IMPORT_CONTAINER_FOR_COPY`. This procedure expects as input the schema and the names of two tables: the table containing the exported results and a table containing any necessary parameters.

The import procedure removes the contents of the target container, writes the data from the export table to the target container, and runs a make. Any existing objects and data in the target container will be deleted before the import procedure starts. After the import procedure has completed successfully, the target container will be a copy of the source container including all the data of the original run-time schema.

> **i Note**
>
> Only objects deployed by the source container's `MAKE` procedure are copied. Objects created manually in the source container's run-time schema are not copied.

To import a container `C1` into a new container `C2` for copy purposes, perform the following steps:

> **i Note**
>
> The container version of the target container must be the same as the container version of the source container. Containers with version greater than or equal to (>=) 40 can also be imported into containers with a higher container version. You can check the container version with the view `_SYS_DI#G.M_CONTAINER_VERSIONS`.

Procedure

1. Open an SQL console and connect to the SAP HANA database with the permissions of the administrator of the HDI container "C1".
2. Open the SQL editor for this database.
3. Ensure the target container exists, for example, `C2`.

   The target container ("C2" in this example) must already exist before the import operation starts.

   ```
   CALL _SYS_DI#G.CREATE_CONTAINER('C2', _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
   ```

   > **i Note**
   >
   > An HDI container must be created by an HDI **container-group** administrator. The new container does not have to be empty for the purpose of an import. However, any existing objects and data in the target container will be deleted before the import.

4. If necessary, grant the object owner of the target container `C2#OO` any privileges for all the external objects required by the container ("C1") to be imported.

   > **i Note**
   >
   > All external objects referenced by the container's design-time objects must be available and accessible to the object owner of the target container.

5. Paste the following SQL code into the SQL editor.

   ```
   IMPORT EXPORT_TABLE FROM '<export_path>';
   CALL C2#DI.IMPORT_CONTAINER_FOR_COPY
     (CURRENT_SCHEMA, 'EXPORT_TABLE', _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
   ```

   > **i Note**
   >
   > If the imported container is copied from within the same database, the `IMPORT`... statement can be skipped.

6. Execute the SQL code, and check that the operation completed successfully (HDI return code 0).
7. Confirm that the container has been successfully imported.

   In this example, you can check that the expected run-time objects of source container `C1` are also present in the target container `C2`.

Related Information

Maintaining SAP HDI Container Groups [page 77]

Export an SAP HDI Container for Copy Purposes [page 108]

5 Content Development in SAP HDI

Develop and deploy database content in the SAP HANA Deployment Infrastructure (HDI).

If you are planning to develop content for deployment to HDI, you need to read and understand the high-level information included in the following sections where you can also find links to more detailed information:

- Content Development with the SAP HDI API [page 112]
- Content Development with the SAP HDI Deployer for XS Advanced [page 112]
- Container Configuration and Name-space Rules [page 113]

i Note

SAP HDI focuses strictly on deployment; HDI does not include any version-control tools, nor does it provide any tools for life-cycle management.

Content Development with the SAP HDI API

HDI container content developers can use the content development API to maintain design-time artifacts in the container's virtual file systems and to create database objects from them. HDI automatically takes care of the dependencies between artifacts and creates the database objects in the correct order. A variety of artifacts types are available from which database objects can be created by HDI. These artifacts are handled by specialized build plug-ins. For more information about developing content with the HDI APIs, see *SAP HDI Container Content Development with the HDI API* and *HDI Artifact Types and Build Plug-ins* in *Related Information* below.

The content development API of an HDI container consists of stored procedures and views located in the container's `#DI` schema (for example, schema `C#DI` in container “`C`”). Before this container API can be called by a content developer, the container administrator must first grant the required privileges to the developer requiring access. For more information about the roles and privileges used to control access to HDI containers, see *Maintaining SAP HDI Containers* in *Related Information*.

Content Development with the SAP HDI Deployer for XS Advanced

In the context of XS advanced, SAP HDI provides a service that enables you to deploy database development artifacts to HDI containers. This service includes a family of consistent design-time artifacts for all key SAP HANA platform database features which describe the target (run-time) state of SAP HANA database artifacts, for example: tables, views, or procedures. These artifacts are modeled, staged (uploaded), built, and deployed into SAP HANA.

The SAP HANA service broker is used to create and destroy HDI containers for XS advanced applications, and the HDI deployment tools (the SAP Node.js application named `@sap/hdi-deploy`), which are based on the HDI's SQL interface, enable the deployment of an XS advanced application's database artifacts to the application's HDI container. The `@sap/hdi-deploy` application is included in the deployed XS advanced application; `@sap/hdi-deploy` must be deployed first so that it can "prepare" the SAP HANA persistence, for example, by creating the required HDI containers. Each HDI container comprises a design-time container for design-time artifacts and a run-time container for the corresponding deployed objects.

The HDI deployer makes assumptions about where it expects to find the source and configuration files it requires to deploy the database components of XS advanced applications. For example, design-time database objects are typically located in the `db/` folder of the XS advanced application's design-time hierarchy. For more information about the HDI deployer, `@sap/hdi-deploy`, see the *JavaScript Run Time* section of the *SAP HANA Developer Guide for XS Advanced*.

Container Configuration and Name-space Rules

In SAP HDI, 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`. For more information about the container-configuration file `.hdbconfig`, see *Related Information* below.

HDI allows developers to configure the naming of generated database objects; the rules for object names and (if desired) name spaces are configured in the `.hdbnamespace` file. The naming rules do not directly relate to the folder structure of the corresponding design-time files. Developers can put a name-space configuration file in a design-time folder, which specifies how the generated database objects are named in the folder where the configuration file is located and also in any sub-folders.

When configuring the name-space rules, you can choose to use the plain design-time object name or the object name with a common name-space prefix. You can also use the object name with a prefix that is a concatenation of a common name-space and the design-time folder path. Choosing a folder-independent naming rule gives you the freedom to change your design-time folder structure without causing incompatible changes to run-time names. For more information about the name-space configuration file `.hdbnamespace`, see *Related Information* below.

Related Information

Managing SAP HDI Containers and Artifacts [page 114]
SAP HDI in the Context of XS Advanced [page 130]
SAP HDI Container Content Development with the HDI API [page 135]
Maintaining SAP HDI Containers [page 90]
The SAP HDI Container Configuration File [page 118]
The HDI Name-Space Configuration File [page 127]
SAP HANA Developer Guide for XS Advanced Model

5.1 Managing SAP HDI Containers and Artifacts

In SAP HANA Deployment Infrastructure (HDI), database development artifacts are deployed to so-called containers.

SAP HDI provides a service that enables you to deploy database development artifacts to HDI containers. This service includes a family of consistent design-time artifacts for all key SAP HANA platform database features which describe the target (run-time) state of SAP HANA database artifacts, for example: tables, views, or procedures. These artifacts are modeled, staged (uploaded), built, and deployed into SAP HANA.

i Note

The HDI focuses strictly on deployment; HDI does not include any version-control tools, nor does it provide any tools for life-cycle management.

In the context of XS advanced, the SAP HANA service broker is used to create and drop HDI containers for XS advanced applications; for pure HDI content development outside the scope of XS advanced, the HDI SQL APIs also enable the maintenance and management of HDI containers and the creation of content in the containers.

The HDI deployment tools deploy database artifacts to an HDI container. Each HDI container comprises a design-time container (DTC) and a run-time container (RTC). Design-time database objects are typically located in the `db` folder of the XS advanced application's design-time hierarchy, as illustrated in the following example:

Output Code

```
app1-hello-world
|- db
   \- src
      |- .hdiconfig                          # HDI build-plugin configuration
      |- .hdinamespace                       # Run-time name-space configuration
      |- myCDSdoc.hdbcds                     # CDS document
      |- myTable.hdbtable                    # DB table definition
      |- mySQLView.hdbview                   # SQL view definition
      |- mySynonym.hdbsynonym                # DB synonym definition
      |- mySynoConfig.hdbsynonymconfig       # DB synonym configuration file
      |- myProcedure.hdbprocedure            # DB procedure definition
      \- myImportData.hdbtabledata           # Data-import definition
```

The deployment process in XS advanced and the build-and-make process in HDI API enable you to populate the database run-time with the specified catalog objects. In addition to database artifacts, HDI also enables you to import and export table content such as business configuration data and translatable texts.

! Restriction

HDI enables you to deploy database objects only; it is not possible (or necessary) to deploy application-layer artifacts such as JavaScript programs or OData objects.

As part of the maintenance and management of containers, system administrators typically perform the following operations:

- Configure access to the SAP HANA Deployment Infrastructure (including containers)
- Create and remove HDI containers, each consisting of both a design-time and a run-time container

The configuration of HDI containers involves the creation and configuration of the following design-time artifacts:

- Container deployment configuration (`.hdiconfig`)
  A JSON file containing a list of the bindings between database artifact types (for example, sequence, procedure, table) and the corresponding build and deployment plug-in (and version).
- Run-time container name-space rules (`.hdinamespace`)
  A JSON file containing a list of design-time file suffixes and the naming rules for the corresponding runtime locations.

> → Tip
>
> You can apply the rules defined in the `.hdinamespace` file either exclusively to the folder it is located in or to both the folder it is located in **and** its subfolders.

Related Information

SAP HDI Containers [page 115]
SAP HDI in the Context of XS Advanced [page 130]
SAP HDI Container Content Development with the HDI API [page 135]

5.1.1 SAP HDI Containers

An SAP HANA HDI container consists of a design-time container and a corresponding run-time container.

The SAP HANA Deployment Infrastructure (HDI) uses so-called containers to store design-time artifacts and the corresponding deployed run-time (catalog) objects. The HDI makes a strict separation between design-time and run-time objects by introducing the following, distinct container types:

- **Design-time** container (DTC)
  Provides an isolated environment for the storage of design-time files
- **Run-time** container (RTC)
  Stores the deployed objects built according to the specification stored in the corresponding design-time artifacts

> → Tip
>
> HDI uses so-called build plug-ins to create run-time objects from the corresponding design-time files, for example, a procedure object in the database catalog from a design-time definition of a stored procedure.

Design-Time Containers

In SAP HDI, the design-time container is used to store the design-time representations of the catalog objects that you want to create during the deployment process. The design-time container does not provide direct access to its database storage; it contains meta-data that can only be accessed by means of the HDI application-programming interface (API). The HDI API enables access to design-time resources inside a design-time container by means of the following file systems:

- work
  A read-write file system where developers can create, modify, and delete design-time resources. The resources in the **work** file system are not modified by HDI, so undeploying a resource from the **deployed** file system leaves the corresponding resource in the work file system untouched. This is because deployed resources are handled by the **deployed** file system.
- deployed
  A read-only file system containing the design-time resources that have been **deployed**. The deployed file system can only be changed by HDI during the deployment and undeployment process.

! Restriction

For maximum isolation and increased security, content stored in a design-time container is owned by a dedicated technical user.

Build Plug-ins are used to transform a certain type of design-time file into the corresponding run-time objects. For this reason, each design-time resource type must be mapped to a build plug-in for each container that is created. It is not possible to create a catalog object from a design-time resource without having a plug-in which can read and transform this source definition during deployment.

A design-time container is always attached to a corresponding **target** run-time container.

Run-Time Containers

In SAP HANA, the database run-time container stores the deployed objects built according to the specification stored in the corresponding design-time artifacts. During the deployment and undeployment operations, all interaction with the run-time container is performed by build plug-ins, which are mapped to distinct design-time artifact types in the container-configuration file (`.hdiconfig`).

A run-time container does not know about the concept of design-time containers. As a consequence, the run-time container can be identified with a standard database schema without any need for HDI meta-data.

The creation of a run-time container results in a new technical database user and a schema with the same name as the one chosen for the name of the run-time container itself. Access to content in the run-time container is controlled by database privileges, for example, by granting SELECT or EXECUTE privileges.

Build Plug-ins

The transformation of design-time resources into corresponding run-time (catalog) objects as well as the extraction of dependency information is performed by so-called build plug-ins. A newly created design-time container cannot access any plug-ins; this access must be defined in the HDI configuration file (`.hdiconfig`). In addition, since there is no default behavior configured, it is not possible to deploy any design-time files (except the two HDI container-configuration and name-space configuration files `.hdiconfig` and `.hdinamespace` respectively) without configuring a build plug-in first. To configure a design-time container to use certain build plug-ins, you must set up and deploy a `.hdiconfig` container-configuration file.

→ Tip

By default, a file suffix (for example, `.hdbprocedure`) is not bound to a specific version of a known build plug-in. Instead, the binding is established by the contents of the `.hdiconfig` file, which must be deployed into a container.

Related Information

The SAP HDI Container Configuration File [page 118]
Managing SAP HDI Containers and Artifacts [page 114]

5.1.2 Naming Conventions in SAP HDI

Rules and restrictions apply to the names of artifacts in the SAP HDI virtual file system

In SAP HANA Deployment Infrastructure (HDI), bear in mind the rules listed below when specifying the name of a database object that is deployed to a HDI container and written to the HDI virtual file system. For HDI run-time objects and their name spaces, valid characters are those that are also valid for catalog names (described in the *SAP HANA SQL and System Views Reference*), except for the following restrictions:

- Name space names cannot contain the forward-slash character (/).
- The backslash (\) can be used but must be escaped in the `.hdbnamespace` file with a second backslash.
- Name space and object names cannot contain the colon character (:)
- The double-colon (::) is used as separator between the name space and the object name.

  i Note

  HANA 2 SPS01 Rev 11: For synonyms, the separator can also be the forward-slash character (/). The forward slash should only be used for migration scenarios where the double-colon cannot be used.
- The name space name can be empty, in which case no separator is used. If a name space **is** used, then object names must include the declared name space as a prefix, for example, `<namespace>::myObject`

! Restriction

Due to catalog rules, the maximum length of the fully qualified object name (including the name space) is 127 characters.

In the HDI virtual file system, the following restrictions apply to the names of files and folders:

- Names can include only a subset of ASCII, namely, the following characters:
  - Lower- or upper-case letters (aA-zZ)
  - Digits (0-9)
  - In addition to the standard characters listed above, HDI files and folders can also include the following special characters:
    ! # % & ' ( ) + , - . ; = @ [ ] ^ _ ` { } ~

- The maximum length of the object's name (including the path to the source file) is 255 characters in the HDI virtual file system.
- The forward slash (/) is used as the folder delimiter in the HDI path:
  `path/to/HDI/object.hdbsuffix`
- An HDI path must **not** start with a forward slash (/). Only relative paths are permitted, for example:
  `path/to/HDI/object.hdbsuffix`
- Trailing spaces ' ' or periods '.' are not allowed in the name of an HDI file or folder
  - Trailing space:
    `path/to/HDI/object.hdbsuffix<trailing space>`
  - Trailing period:
    `path/to/HDI/object.hdbsuffix.`
- For HDI folders:
  - An empty path denotes the root folder
  - Other paths must end with a forward slash (/)
    - `path/`
    - `path/to/`
    - `path/to/HDI/object/`
- Core Data Services (CDS)
  Additional restrictions apply to the names of CDS artifacts deployed to an HDI container. For more details, see *Related Information*.
- SAP Web IDE for SAP HANA
  If you are using SAP Web IDE for SAP HANA to develop applications for XS advanced, additional restrictions apply to the names of design-time artifacts. For more details, see *Related Information*.

Related Information

Naming Conventions in CDS (SAP HANA Developer Guide)
SAP Web IDE for SAP HANA Reference

5.1.3 The SAP HDI Container Configuration File

Bind design-time file types to the corresponding build plug-in required in the SAP HANA Deployment Infrastructure (HDI).

In SAP HDI, a design-time container defines an isolated environment for design-time files. To create run-time objects from the corresponding design-time files (for example, a catalog object from a source definition of a stored procedure), HDI uses so-called Build Plug-ins. It is not possible to create a catalog object from a design-time resource without having a plug-in which can read and transform this source definition. For this reason, each design-time resource type must be mapped to a build plug-in for each container that is created. This mapping is configured in an HDI container-configuration file: a file with no name but the mandatory file suffix `.hdiconfig`.

Note

The `.hdiconfig` file must be located in the root folder of the database module. However, you can include additional `.hdiconfig` files further down the source-file hierarchy, for example, if you want to change (expand/restrict) the definitions at a certain point in the hierarchy.

By default, a file suffix (for example, `.hdbprocedure`) is not bound to a specific version of a known build plug-in. Instead, the binding is established by the contents of the `.hdiconfig` file, which must be deployed into the container along with the artifacts it describes. An `.hdiconfig` file is a JSON file with the following structure:

Sample Code

```
{
"plugin_version": "2.0.30.0",
"file_suffixes" : {
   "hdbsynonym" : {
      "plugin_name" : "com.sap.hana.di.synonym"},
   "hdbview" : {
      "plugin_name" : "com.sap.hana.di.view"},
   "hdbcalculationview" : {
      "plugin_name" : "com.sap.hana.di.calculationview"},
   "hdbprocedure" : {
      "plugin_name" : "com.sap.hana.di.procedure"},
   "<hdb_file_suffix_#>" : {
      "plugin_name" : "<plugin_NAME>" }
    }
}
```

It is also possible to define the plug-in version for individual artifact plug-ins, as illustrated in the following example:

Sample Code

```
{
"file_suffixes" : {
   "hdbsynonym" : {
      "plugin_name"    : "com.sap.hana.di.synonym",
      "plugin_version": "2.0.30.0"},
   "hdbview" : {
      "plugin_name"    : "com.sap.hana.di.view",
      "plugin_version": "2.0.30.0" },
   "hdbcalculationview" : {
      "plugin_name"    : "com.sap.hana.di.calculationview",
      "plugin_version": "2.0.30.0"},
   "hdbprocedure" : {
      "plugin_name"    : "com.sap.hana.di.procedure",
      "plugin_version": "2.0.30.0"},
   "<hdb_file_suffix_#>" : {
      "plugin_name"    : "<plugin_NAME>","<plugin_VERSION>" }
    }
}
```

Note

If the plug-in version is defined at both the global and local (individual) level, as illustrated in the following example, then the version specified for an HDI build plug-in at the individual level takes precedence.

```
{
```

```
"plugin_version": "2.0.30.0",
"file_suffixes" : {
   "hdbsynonym" : {
      "plugin_name"   : "com.sap.hana.di.synonym"},
   "hdbview" : {
      "plugin_name"   : "com.sap.hana.di.view"},
   "hdbcalculationview" : {
      "plugin_name"   : "com.sap.hana.di.calculationview",
      "plugin_version": "2.0.20.0"},
   "hdbprocedure" : {
      "plugin_name"   : "com.sap.hana.di.procedure",
      "plugin_version": "2.0.20.0"}
    }
}
```

The plug-in **name** corresponds to the plug-in name in reverse-URL notation, for example, `com.sap.hana.di.procedure` or `com.sap.hana.di.view`.

The plug-in **version** is described as follows: **major** version, **minor** version, **revision** version, and an optional **patch** version, for example, `2.0.20.0` signifies the SAP HANA version 2.0 SPS 02. Similarly, `1.0.12.0` signifies the SAP HANA version 1.0 SPS 12.

Note

The three level plug-in version format previously used in SAP HANA 1.0, for example, "`13.1.0`", is still valid. Values in the old format will be internally converted to the appropriate four level format for SAP HANA 1.0 revisions.

Related Information

Managing SAP HDI Containers and Artifacts [page 114]
SAP HDI Artifact Types and Build Plug-ins Reference [page 210]

5.1.3.1 SAP HDI Container Configuration File Syntax

In SAP HANA Deployment Infrastructure (HDI), the JSON syntax is used to format the content of the HDI container-configuration file (`.hdiconfig`).

Sample Code

```
{
  "plugin_version": "<plugin_VERSION>"
  "file_suffixes" : {
    "<file_suffix_1>" : {
      "plugin_name"    : "<plugin_NAME>",
      "plugin_version": "<plugin_VERSION>" // If different from global version
    },
    "<file_suffix_2>" : {
      "plugin_name"    : "<plugin_NAME>"
    },
    "<file_suffix_#>" : {
      "plugin_name"    : "<plugin_NAME>",
      "plugin_version": "<plugin_VERSION>" // If different from global version
      },
      {
        [...]
      }
    }
}
```

file_suffixes

Define one or more file suffixes which you want to bind to a build plug-in for deployment to an HDI container. The file suffix specifies the **type** of database artifact, for example, `.hdbsequence` or `.hdbview`.

Sample Code

```
{
  "file_suffixes" : {
    "<file_suffix_1>" : {
      "plugin_name"    : "<plugin_NAME>",
      "plugin_version": "<plugin_VERSION>"  //optional
    }
  }
}
```

You can use the asterisk character (*) to define a **global** file suffix which binds all remaining, unbound file suffixes to a specific build plug-in, for example, to bind all unbound file suffixes to the `Copy Only` plug-in.

! Restriction

All “`hdi*`” file suffixes (for example, `.hdiconfig` or `hdinamespace`) are reserved for use with HDI; these file suffixes cannot be (re)configured.

The following example of an abbreviated HDI configuration file illustrates how the design-time artifact types (for example, `.hdbsynonym`, `.hdbtable`, `.hdbcalculationview`) are mapped to their corresponding HDI build plug-in:

Code Syntax

`.hdiconfig`

```
{
  "plugin_version" : "2.0.20.0",    // global, optional, defaults to 0.0.0.0
  "file_suffixes" : {
      "hdbsynonym" : {
        "plugin_name" : "com.sap.hana.di.synonym"
      },
      "hdbsynonymconfig" : {
        "plugin_name" : "com.sap.hana.di.synonym.config"
      },
      "hdbtable" : {
        "plugin_name" : "com.sap.hana.di.table"
      },
      "hdbdropcreatetable" : {
        "plugin_name" : "com.sap.hana.di.dropcreatetable"
      },
      "hdbcalculationview" : {
        "plugin_name" : "com.sap.hana.di.calculationview"
      },
      "hdbcds" : {
        "plugin_name"  : "com.sap.hana.di.cds",
        "plugin_version": "2.0.19.0"
      },
      "<file suffix 2>" : {
       "plugin_name"   : "<plugin name>",
       "plugin_version": "<plugin_version>" //local, optional
      }
}
```

plugin_name

Use the `plugin_name` key to specify the name of the HDI build plug-in that you want to associate with a specific HDI artifact type.

Sample Code

```
"plugin_name"   : "<plugin_NAME>",
```

For example, you can associate the artifact suffix for the HDB SQL view (`.hdbview`) with the build plug-in `com.sap.hana.di.view`.

Sample Code

```
"plugin_name"   : "com.sap.hana.di.view",
```

plugin_version (optional)

Use the optional `plugin_version` key to specify the version number of the HDI build plug-in that you want to associate with one or more HDI plug-ins.

Sample Code

```
"plugin_version":
"<Major_Version>.<Minor_Version>.<Revision_Version>.<Patch_Version>"
```

→ Tip

To specify one plug-in version for **all** configured plug-ins, define `"plugin_version"` globally at the start of the configuration file. To specify a specific version for a particular plug-in, define `"plugin_version"` locally alongside the plug-in name to which the specific version refers. The locally defined plug-in version takes precedence over the globally defined version.

The value of the `plugin_version` key is interpreted as a minimal required version of the build plug-in. Version numbers follow the format `<major>.<minor>.<Revision_Version>.<patch>`. An installed build plug-in with the same major version is considered to be compatible if its minor version is greater than the requested minor version, or its minor version is equal to the requested version and the revision or patch number is greater than (or equal to) the requested version.

Sample Code

HDI Plug-in Version for SAP HANA 2.0 SPS 00 Revision 0

```
"plugin_version": "2.0.0.0"
```

i Note

The three-level plug-in format used in SAP HANA 1.0 (for example, “`11.1.0`”) is still valid; this old, three-level version format is converted internally to the appropriate four-level format required for newer versions of SAP HANA revisions.

In SAP HANA 1.0 there was no top-level (global) `"plugin_version"` entry, and the bottom-level (local) `"plugin_version"` entries were mandatory. In SAP HANA 2.0, the version of **all** plug-ins shipped with SAP HANA is the same as (and equal to) the SAP HANA version. For this reason, there should normally be no need for the local `"plugin_version"` entries since the result of the compatibility check would depend only on the largest entry in use. This means that it is sufficient to provide the top-level (global) entry or no entry at all if no particular minimum SAP HANA version is required. However, if present, bottom-level (local) entries override the top-level (global) entry or the default `"plugin_version"` of 0.0.0.0, respectively.

Related Information

Set up an SAP HDI Container for XS Advanced Applications [page 133]
SAP HDI Artifact Types and Build Plug-ins Reference [page 210]

5.1.4 Define Run-Time Name-Space Rules for SAP HDI

Define rules for run-time name spaces in SAP HANA Deployment Infrastructure (HDI) containers.

Prerequisites

For the definition and deployment of name space rules for run-time container objects in SAP HDI, the following prerequisites apply:

- The design-time folder hierarchy contains application artifacts
- The design-time containers exist
- The corresponding run-time containers exist

Context

In SAP HANA Deployment Infrastructure (HDI), the application objects deployed to an HDI container must also be assigned to a run-time name space, which is defined in the `.hdinamespace` file.

→ Tip

In the following examples, use **either** the `xs` command (on-premise environments) **or** the `cf` command (SAP Cloud Foundry).

Procedure

1. Create the name-space configuration file (`.hdinamespace`) in the design-time location of your choice.

   Design-time database objects are typically located in the `db/` folder of the application's design-time hierarchy, as illustrated in the following example:

   Output Code

   ```
   app1-hello-world
   |- db
      \- src
         |- .hdiconfig                          # HDI build-plugin configuration
         |- .hdinamespace                       # Run-time name-space
   configuration
         |- myCDSdoc.hdbcds                     # CDS document
         |- myTable.hdbtable                    # DB table definition
         |- mySQLView.hdbview                   # SQL view definition
         |- mySynonym.hdbsynonym                # DB synonym definition
         |- mySynoConfig.hdbsynonymconfig       # DB synonym configuration file
         |- myProcedure.hdbprocedure            # DB procedure definition
         \- myImportData.hdbtabledata           # Data-import definition
   ```

   i Note

   If you are using the SAP HDI API to create content, you first use the `WRITE` API to create the `.hdinamespace` in the HDI container's "work" file system and then deploy the name-space configuration to the "deployed" file system of the same HDI container, for example, with the `MAKE` command as described below.

2. Configure the run-time name spaces for the objects added to the run-time container during application deployment.

   In SAP HANA, name-space rules are defined in one or more file resources named `.hdinamespace`, which must be located in the design-time folder to which the naming rules apply - or the root folder of a hierarchy to which the naming rules apply. The content of the `.hdinamespace` file is specified according to the JSON format with the following structure:

> Sample Code
>
> `src/.hdinamespace`
>
> ```
> {
>    "name"      : "com.sap.hana.example",
>    "subfolder" : "<[append | ignore]>"
> }
> ```

> → Tip
>
> Change the path defined in `name` and, for `subfolder`, choose between `append` (add sub-folder name to object name space) and `ignore` (do **not** add sub-folder name to the object name space).

3. Deploy the name-space configuration to the container where the rules apply.

   The deployment operation pushes the specified application's database objects to the HDI container to which it is bound.

   - Command-line interface:
     `[xs | cf] deploy`
   - SAP Web IDE:
     Build > *Build*

     > i Note
     >
     > You can use the *Database Explorer* tool to view the contents of the XS application's run-time container.

   - HDI SQL API
     To deploy the newly created files with the HDI `MAKE` API, you must provide the file name in the `PATH` parameter.

     > ! Restriction
     >
     > This step requires the privileges of an HDI container administrator. Only files can be deployed; it is not possible to deploy folders.

     > Sample Code
     >
     > Deploy Files to the Work File System of Container "C"
     >
     > ```
     > CREATE LOCAL TEMPORARY COLUMN TABLE #DEPLOY_PATHS LIKE
     > _SYS_DI.TT_FILESFOLDERS;
     > INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('.hdinamespace');
     > CREATE LOCAL TEMPORARY COLUMN TABLE #UNDEPLOY_PATHS LIKE
     > _SYS_DI.TT_FILESFOLDERS;
     > CREATE LOCAL TEMPORARY COLUMN TABLE #PATH_PARAMETERS LIKE
     > _SYS_DI.TT_FILESFOLDERS_PARAMETERS;
     > CALL C#DI.MAKE(#DEPLOY_PATHS, #UNDEPLOY_PATHS, #PATH_PARAMETERS,
     > _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
     > DROP TABLE #DEPLOY_PATHS;
     > DROP TABLE #UNDEPLOY_PATHS;
     > DROP TABLE #PATH_PARAMETERS;
     > ```

Related Information

The HDI Name-Space Configuration File [page 127]
The SAP HDI Name-space Configuration Syntax [page 128]

5.1.4.1 Run-Time Name Spaces in SAP HDI

SAP HDI defines a strict separation between the naming of run-time objects and the organization of design-time files.

In SAP HANA Deployment Infrastructure (HDI), there is no simple, one-to-one mapping between the file-system structure and the names of the resulting run-time objects. Instead, the names of run-time objects are determined by naming rules defined in HDI name-space files. If desired, the file-system structure can still be used to derive the run-time name space of an object. However, decoupling design-time file structure and run-time name-space names allows the file-system structure to reflect the organization of work, and it is easy to relocate design-time resources without changing run-time name spaces. Note that if design-time folder names do not contribute to run-time names, you avoid problems with limitations on name length.

In SAP HANA, name-space rules for run-time objects are defined in one or more files named `.hdinamespace`, which must be located in the folder to which the naming rules apply - or the root folder of a folder structure to which the naming rules apply. The name-space rules allow you to specify the following elements:

- A name-space prefix, for example, `com.acme.hana.example`
- The name of design-time folder(s) containing the deployed objects

Code Syntax

Run-Time Name Space with Appended Folder Name

```
com.acme.hana.example.<Design-Time_Folder_Name>::<Run-Time_Object_Name>
```

The following table illustrates how the rules specified in the name-space configuration file influence the way run-time objects are named. If the name-space rules specify that the name of the design-time sub-folder should be **appended** to the run-time name-space prefix, for example, `com.acme.hana.example`:

Folder Names Appended to the Run-Time Name Space

Design-time Resource Path	Name of Run-Time Object (append)
`/src/`	`com.acme.hana.example::<Object_Name>`
`/src/db`	`com.acme.hana.example.db::<Object_Name>`
`/src/data`	`com.acme.hana.example.data::<Object_Name>`

Related Information

The HDI Name-Space Configuration File [page 127]
The SAP HDI Name-space Configuration Syntax [page 128]

5.1.4.2 The HDI Name-Space Configuration File

The SAP HANA Deployment Infrastructure (HDI) uses a JSON resource to define naming rules for run-time objects.

In SAP HDI, name-space rules are defined in one or more file resource named `.hdinamespace`, which must be located in the folder to which the naming rules apply - or the root folder of a hierarchy to which the naming rules apply. The content of the `.hdinamespace` file is specified according to the JSON format with the following structure:

Sample Code

src/.hdinamespace

```
{
   "name"       : "com.sap.hana.example",
   "subfolder" : "<[append | ignore]>"
}
```

Every folder can contain its own name-space file. For a name-space file to take effect, it has to be deployed in the same way as any other design-time file. After deployment, a name-space file provides run-time name-space information for all files in the same folder.

The `.hdinamespace` file provides name-space information for all sub-folders where no additional name-space file is present. For those sub-folders without a name-space configuration file, the name space depends on the value of the `subfolder` key in the name-space-configuration file in the parent folder.

- `append`
  Add the name of the sub-folder to the run-time name space for the deployed objects
- `ignore`
  Do **not** add the name of the sub-folder to the run-time name space for the deployed objects

Note

Name spaces and the naming rules are configurable and do not directly relate to the folder structure of the design-time files.

Developers can place a name-space configuration file (`.hdinamespace`) in a design-time folder, which specifies how the generated database objects are named in the sub-hierarchy underneath this folder. For example, you can just use the plain design-time object name, or the object name with a common name-space prefix, or even the object name with a prefix that is a concatenation of a common name space and the design-time folder path. Choosing a folder-independent naming rule provides the freedom to change the design-time folder structure at a later point in time without causing incompatible changes to run-time names.

Related Information

Run-Time Name Spaces in SAP HDI [page 126]
The SAP HDI Name-space Configuration Syntax [page 128]
Define Run-Time Name-Space Rules for SAP HDI [page 123]

5.1.4.2.1 The SAP HDI Name-space Configuration Syntax

In SAP HANA Deployment Infrastructure (HDI), the contents of the `.hdinamespace` file are formatted with the JSON syntax

Sample Code

src/.hdinamespace

```
{
    "name"      : "com.sap.hana.example",
    "subfolder" : "<[append | ignore]>"
}
```

name

Use the `name` property in the `.hdinamespace` configuration file to specify the name of run-time space to map to the name of (and objects in) the file-system folder where the `.hdinamespace` file is located.

Sample Code

```
"name" : "com.sap.hana.example",
```

subfolder

Use the `subfolder` key in the `.hdinamespace` configuration file to specify if the name of the file-system folder where the `.hdinamespace` file is located should be **appended** to the name of the corresponding run-time space or **ignored**.

Sample Code

```
{
"name" : "com.sap.hana.example",
"subfolder" : "<[append | ignore]>"
}
```

For those sub-folders without a name-space configuration file, the name space depends on the value of the `subfolder` key in the name-space configuration file in the parent folder, for example: `append` or `ignore`.

- `"subfolder" : "ignore"`
  The sub-folder inherits the name-space information from the parent folder. In this case, all sub-folders use the same run-time name space.

  Design-Time Folder Names Ignored in Run-Time Name Space

Design-time Resource Path	Name of Run-time Object (ignore)
`/src/`	`com.sap.hana.example::<Object_Name>`
`/src/db`	`com.sap.hana.example::<Object_Name>`
`/src/db/proc`	`com.sap.hana.example::<Object_Name>`
`/src/data`	`com.sap.hana.example::<Object_Name>`
`/src/data/import`	`com.sap.hana.example::<Object_Name>`

- `"subfolder" : "append"`
  The run-time name-space for a sub-folder is derived by appending the sub-folder name to the name-space name of the parent folder.

  Design-Time Folder Names Appended to Run-time Name Space

Design-time Resource Path	Name of Run-Time Object (append)
`/src/`	`com.sap.hana.example::<Object_Name>`
`/src/db`	`com.sap.hana.example.db::<Object_Name>`
`/src/db/proc`	`com.sap.hana.example.db.proc::<Object_Name>`
`/src/data`	`com.sap.hana.example.data::<Object_Name>`
`/src/data/import`	`com.sap.hana.example.data.import::<Object_Name>`

Related Information

Run-Time Name Spaces in SAP HDI [page 126]
The HDI Name-Space Configuration File [page 127]
Define Run-Time Name-Space Rules for SAP HDI [page 123]

5.2 SAP HDI in the Context of XS Advanced

Understand the role and scope of the HDI in XS advanced model.

In SAP HANA Deployment Infrastructure (HDI), database development artifacts are deployed to so-called containers. In the context of XS advanced, SAP HDI provides a service that enables you to deploy database development artifacts to HDI containers. This service includes a family of consistent design-time artifacts for all key SAP HANA platform database features which describe the target (run-time) state of SAP HANA database artifacts, for example: tables, views, or procedures. These artifacts are modeled, staged (uploaded), built, and deployed into SAP HANA.

Note

The HDI focuses strictly on deployment; HDI does not include any version-control tools, nor does it provide any tools for life-cycle management.

In the context of XS advanced, the SAP HANA service broker is used to create and destroy HDI containers for XS advanced applications, and the HDI deployment tools deploy an XS advanced application's database artifacts to the appropriate HDI container. Each HDI container comprises a design-time container (DTC) and a run-time container (RTC). Design-time database objects are typically located in the `db` folder of the XS advanced application's design-time hierarchy, as illustrated in the following example:

Output Code

```
app1-hello-world
|- db
|   |- src
|      |- .hdiconfig                        # HDI build-plugin configuration
|      |- .hdinamespace                     # Run-time name-space configuration
|      |- myCDSdoc.hdbcds                   # CDS document
|      |- myTable.hdbtable                  # DB table definition
|      |- mySQLView.hdbview                 # SQL view definition
|      |- mySynonym.hdbsynonym              # DB synonym definition
|      |- mySynoConfig.hdbsynonymconfig     # DB synonym configuration file
|      |- myProcedure.hdbprocedure          # DB procedure definition
|      \- myImportData.hdbtabledata         # Data-import definition
\- 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.

Related Information

SAP HDI Containers [page 115]

5.2.1 The SAP HDI Deployer

SAP HDI provides dedicated tools to enable the deployment of design-time database artifacts to the SAP HANA database.

For XS advanced and Cloud Foundry environments, the SAP HANA Deployment Infrastructure (HDI) includes a so-called "Deployer", which is a Node application that is included in a Multi-Target Application (MTA) for the purpose of deploying an XS advanced application's HDI database-related **design-time** artifacts to the application's corresponding HDI containers. When an MTA is deployed to the target run-time platform, for example, XS advanced or SAP Cloud Platform @ Cloud Foundry, the HDI Deployer application `@sap/hdi-deploy` is deployed first so that it can “prepare” the SAP HANA persistence; by the time the services defined in the MTA's deployment descriptor are started in the XS advanced (or Cloud Foundry) run-time environment, the HDI container is ready for use.

You can install the HDI Deployer by including the Node.js application “`@sap/hdi-deploy`” in the `package.json` file that defines the dependencies inside your XS advanced application's `db/` module, as illustrated in the following example:

Code Syntax

`myApp/db/package.json`

```
{
  "name": "deploy",
  "dependencies": {
    "@sap/hdi-deploy": "3.9.1"
  },
  "scripts": {
    "start": "node node_modules/@sap/hdi-deploy/"
  }
}
```

i Note

Java applications also require the HDI Deployer `@sap/hdi-deploy` for the deployment to HDI of database-related design-time artifacts. However, after deployment Java applications typically connect to the run-time schema of the application's corresponding HDI container using JDBC. Access to the HDI run-time containers is also possible with the SAP SQL APIs for SAP HANA DI (HDI).

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

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

Dynamic Deployment to HDI

The Node.js module `@sap/hdi-dynamic-deploy` provides a Node.js-based HTTP server that can be used to deploy database content to dynamically created HDI containers, for example, containers created by means of the XS advanced Instance Manager. The HDI Dynamic Deployer is intended for use in not only in XS Advanced on-premise scenarios but also in the SAP Cloud Platform Cloud Foundry (CF) environment.

i Note

`@sap/hdi-dynamic-deploy` is based on `@sap/hdi-deploy` which should be used if a static deployment at deployment time is sufficient.

Typically, `@sap/hdi-dynamic-deploy` is installed by including a dependency in the `package.json` file of your XS advanced application's database module, as illustrated in the following example:

Sample Code

`myApp/db/package.json`

```
{
  "name": "deploy",
  "dependencies": {
     "@sap/hdi-dynamic-deploy": "1.5.0"
  },
  "scripts": {
     "start": "node node_modules/@sap/hdi-dynamic-deploy/"
  }
}
```

For more detailed information about using the `@sap/hdi-dynamic-deploy`, see the `README.md` file included in the application's corresponding NPM package.

Run-time Access to HDI Containers

If XS advanced applications need access to their HDI run-time containers outside the scope of the HDI deployer, they can use the SAP SQL APIs for SAP HANA DI, which include an API specifically intended for content development in HDI containers.

i Note

When working with the SAP SQL API for HDI, the HDI Deployer "`@sap/hdi-deploy`" described above is **not** required.

For example, for Node.js applications that need to create database objects at run time, the Node.js module `@sap/hdi` provides an HDI client library that includes a selection of asynchronous methods which enable access to SAP HANA deployment infrastructure functionality, for example: `connect()` and `disconnect()`, `createContainer()` and `dropContainer()`, and `configureLibraries()`.

Similarly, the `sap-java-hdi` client library enables access to the SAP HANA Deployment Infrastructure (HDI) for Java applications that need to create database objects at run time. Version 2.0 of the SAP HANA DI (HDI) client library for Java applications wraps the HDI SQL APIs in Java classes and methods which enable access to the APIs that are provided for managing HDI, its containers, and any container groups. Assuming the `hdiUser` has the privileges required to access `_SYS_DI`, the `sap-java-hdi` client library can be used to connect to the SAP HANA database, configure HDI, create and drop HDI containers, configure container build plug-in libraries, and so on.

i Note

For more information about the standard and additional Java and Node.js libraries that you can use in your XS advanced applications, see *Related Information* below.

Related Information

Standard Node.js Packages for XS Advanced (Developer Guide)
Standard Java Packages for XS Advanced (Developer Guide)
SAP HANA Developer Guide for XS Advanced Model

5.2.2 Set up an SAP HDI Container for XS Advanced Applications

Set up the environment required for the deployment to the SAP HANA Deployment Infrastructure (HDI) of an XS advanced application's database artifacts.

Context

In SAP HDI, the design-time artifact types for XS advanced applications are distinguished by the artifact's file suffix (for example, `.hdbtable`). These artifact types must be associated with a corresponding build plug-in to ensure the successful build and deployment. The binding between an artifact type and a corresponding build plug-in is established by the contents of the `.hdiconfig` file, which must be deployed into the target container, too.

In the context of XS advanced, the service broker is used to create HDI containers for the deployed application. An HDI container is a set of schemata and a set of users that, together, enable an isolated deployment of database artifacts. From an application perspective, it is just a technical user and a schema.

For backward compatibility with legacy applications, the deployed objects can also be assigned to a run-time name space, which is defined in the `.hdinamespace` file that also has to be deployed to the target HDI container.

→ Tip

If you are using SAP Web IDE, the configuration described here is available by default. If you are working in a command shell, use **either** the `xs` command (on-premise environments) **or** the `cf` command (SAP Cloud Foundry) in the following steps.

Procedure

1. Check that the HDI service plan `hdi-shared` is available for your organization or space.

```
[xs | cf] marketplace
Getting services from marketplace...
service        plans                                  description
------------------------------------------------------------------------------
---
fs-storage     free                                   xs file service
provides...
hana           hdi-shared, sbss, schema, securestore  SAP HANA database
xsuaa          default, devuser, space                Manage app
authorization...
managed-hana   hdi-shared, schema, securestore        Creates service
instance...
auditlog       free                                   Broker for Audit log
Service
```

2. Create an HDI container using the service broker.

   `[xs | cf] create-service hana hdi-shared <myHDIcontainer>`

3. Bind the new HDI container to the application that you want to use the container.

   `[xs | cf] bind-service <myApplication> <myHDIcontainer>`

4. Configure the build plug-ins required for the artifacts deployed to the HDI container.

   By default, a file suffix (for example, `.hdbprocedure`) is not bound to a specific version of a build plug-in. Instead, the binding is established by the contents of the `.hdiconfig` file, which must be deployed into a container along with the database artifacts it lists. An `.hdiconfig` file is a JSON file with the structure illustrated in the following example:

   Sample Code

   Example `.hdiconfig` File

```
{
"plugin_version": "2.0.30.0"
"file_suffixes" : {
    "hdbtable" : {
        "plugin_name" : "com.sap.hana.di.table"},
    "hdbview" : {
        "plugin_name" : "com.sap.hana.di.view"},
    "hdbprocedure" : {
        "plugin_name" : "com.sap.hana.di.procedure"},
    "<hdb_file_suffix_#>" : {
        "plugin_name"   : "<plugin_NAME>",
        "plugin_version": "<local_plugin_VERSION>"}
    }
}
```

   → Tip

   Add to the `.hdiconfig` file, the file-suffixes of the design-time artifacts deployed by your applications. The `.hdiconfig` file must be placed in the root folder of the database source-files, for example, `app1/db/.hdiconfig`.

5. Configure the run-time name spaces for the objects added to the run-time container during application deployment.

In SAP HANA HDI, name-space rules are defined in one or more file resources named `.hdinamespace`, which must be located in the design-time folder to which the naming rules apply - or the root folder of a hierarchy to which the naming rules apply. The content of the `.hdinamespace` file is specified according to the JSON format with the following structure:

→ Tip

Change the path defined in `name` and, for `subfolder`, choose between `append` (add sub-folder name to object name space) and `ignore` (do **not** add sub-folder name to the object name space).

Sample Code

`src/.hdinamespace`

```
{
   "name"       : "com.sap.hana.example",
   "subfolder" : "<[append | ignore]>"
}
```

6. Deploy the application.

   The deployment operation pushes the specified application's database objects to the HDI container to which it is bound.

   - Command-line interface:
     `[xs | cf] deploy`
   - SAP Web IDE:
     Build > *Build*

   i Note

   You can use the *Database Explorer* tool to view the contents of the XS application's run-time container.

Related Information

The SAP HDI Container Configuration File [page 118]
SAP HDI Container Configuration File Syntax [page 120]
SAP HDI Artifact Types and Build Plug-ins Reference [page 210]
Managing SAP HDI Containers and Artifacts [page 114]

5.3 SAP HDI Container Content Development with the HDI API

SAP HDI includes an SQL API for the development of content in SAP HDI containers.

HDI container content developers can use the content development API to maintain design-time artifacts in the container's virtual file systems and to create database objects from them. HDI automatically takes care of the dependencies between artifacts and creates the database objects in the correct order. A variety of artifacts types are available from which database objects can be created by HDI. These artifacts are handled by specialized build plug-ins. For more information about HDI build plug-ins, see *HDI Artifact Types and Build Plug-ins* in *Related Information* below.

The content development API of an HDI container includes stored procedures and views located in the container's `#DI` schema (for example, schema `C#DI` in container “`C`”). Before this API of a container can be called by a content developer, an administrator of the container has to grant the necessary privileges to the developer. If you are the administrator of the container, you can find more information in *Related Information* under *Maintaining SAP HDI Containers*. The APIs provided enable developers, amongst other things, to create objects and their content in the HDI container's work file system and deploy the objects to the deployed file system.

The HDI Container's Work File System

After a container administrator has configured the build and plug-in libraries for an HDI container and granted developers the required access privileges for content-development, developers can start to `WRITE` files and their content to the container's “work” file system. The `DELETE` API enables you to remove files from the HDI container's work file system, and the `LIST`, `READ`, and `STATUS` APIs enable access to the work file-system's content. For more information about the APIs that provide access to an HDI container's "work" file system, see *The HDI Container API* in *Related Information*.

The HDI Container's Deployed File System

To create database objects from the files in the HDI container's work file system, one of the `MAKE` operations is used, which copies the affected files to the HDI container's deployed file system. You can display information about the content of the deployed file-system's content with the `LIST_DEPLOYED` and `READ_DEPLOYED` APIs, too. For more information about the APIs that provide access to an HDI container's "deployed" file system, see *The HDI Container API* in *Related Information*.

Container Access for Content Development

The APIs provided for administrative operations in the context of content development enable you to grant (or revoke) the roles and privileges that provide access the HDI container or the container's schema, for example, `GRANT_CONTAINER_SCHEMA_PRIVILEGES` and `GRANT_CONTAINER_SCHEMA_ROLES` and the listing of the build and plug-in libraries configured for an HDI container (`LIST_CONFIGURED_LIBRARIES`). For more information about the APIs that enable the management of access privileges for an HDI container, see *The HDI Container API* in *Related Information*.

> **i Note**
>
> The HDI also includes views that can be used to display information about the currently running `MAKE` operations, recent messages and objects, or the deployed roles.

Related Information

The HDI Container API [page 141]
HDI Container Views [page 205]
SAP HDI Artifact Types and Build Plug-ins Reference [page 210]
Maintaining SAP HDI Containers [page 90]

5.3.1 Set up an SAP HDI Container with the HDI API

Set up the environment required for the deployment to the SAP HANA Deployment Infrastructure (HDI) of database artifacts.

Prerequisites

To complete this task, bear in mind the following prerequisites:

- SAP HDI is enabled
- The SAP HDI administrator has created an SAP HDI **group** administrator
- The SAP HDI group administrator has created an SAP HDI **container** administrator
- You can connect to the SAP HANA database where you want to maintain HDI containers.

Context

In SAP HDI, the design-time artifact types for XS advanced applications are distinguished by the artifact's file suffix (for example, `.hdbtable`). These artifact types must be associated with a corresponding build plug-in to ensure that the artifacts can be successfully built and deployed. The binding between an artifact type and a corresponding build plug-in is defined in the contents of the `.hdiconfig` file, which must be deployed into the target container, too.

In the context of HDI, the SQL container-administration APIs can be used to create (and drop) the target HDI containers for the deployed applications as well as write and deploy files to the “work” file systems of the newly created HDI containers. It is important to bear in mind that an HDI container is a set of schemata and a set of users that, together, enable an isolated deployment of database artifacts. From an application perspective, however, it is just a technical user and a schema.

For backward compatibility with legacy applications, the deployed objects can also be assigned to a run-time name space, which is defined in the `.hdinamespace` file that also has to be deployed to the target HDI container.

> → Tip
>
> For information about setting up an HDI container for use by XS advanced applications, see *Related Information* below.

Procedure

1. Check that HDI is enabled in your SAP HANA instance and that all necessary administration users are available with the required roles and privileges.

   The `diserver` process must usually be enabled by the database administrator before HDI can be used. If required by the usage scenario, other database process may also need to be started as well. The database administrator SYSTEM is needed for the first-time enabling of HDI in SAP HANA and for creating an HDI administrator; it is the HDI administrator who is then responsible for performing the tasks required to set up and maintain other HDI administrators, for example, container (and container-group) administrators.
2. Create an HDI container group.

   Note

   This step must be performed by a user with the privileges of an HDI administrator.

   To create an HDI container group named "G", run the following SQL code in an SQL console:

   Sample Code

   ```
   CALL _SYS_DI.CREATE_CONTAINER_GROUP('G', _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
   ```
3. Create an HDI container.

   Note

   This step must be performed by a user with the privileges of an HDI container-group administrator.

   To create an HDI container named "C" in container group "G", run the following SQL code in an SQL console:

   Sample Code

   ```
   CALL _SYS_DI#G.CREATE_CONTAINER('C', _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
   ```
4. Configure the build plug-ins required for the artifacts deployed to an 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. The `.hdiconfig` file must be placed in the root folder of the database source-files, for example, `myApp/db/.hdiconfig`. The contents of the `.hdiconfig` file must be specified in the JSON format illustrated in the following example; the JSON content is included as `VALUES` for the `CONTENT` parameter in the `INSERT` command shown later in this tutorial:

   Sample Code

   Example `.hdiconfig` File

   ```
   {
   "plugin_version" : "2.0.30.0",
   "file_suffixes" : {
       "hdbtable" : {
           "plugin_name" : "com.sap.hana.di.table" },
       "hdbview" : {
   ```

```
        "plugin_name" : "com.sap.hana.di.view"},
    "hdbprocedure" : {
        "plugin_name" : "com.sap.hana.di.procedure",
    "<hdb_file_suffix_#>" : {
        "plugin_name"   : "<plugin_NAME>",
        "plugin_version": "<optional_plugin_VERSION>" }
    }
}
```

i Note

The HDI container administrator can display a list of the build and plug-in libraries currently available in an HDI container (and the version) by using the `LIST_CONFIGURED_LIBRARIES` API. The HDI container administrator can also configure a default (or custom) set of build and plug-in libraries for an HDI container. For more information, see *Maintaining SAP HDI Containers* in *Related Information*.

5. Configure the run-time name spaces for the objects added to the HDI 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 folder hierarchy to which the naming rules apply. The content of the `.hdinamespace` file is specified according to the JSON format with the following structure; the JSON content is included as `VALUES` for the `CONTENT` parameter in the `INSERT` command shown later in this tutorial:

   Sample Code

   `src/.hdinamespace`

```
{
    "name"      : "com.sap.hana.example",
    "subfolder" : "<[append | ignore]>"
}
```

   → Tip

   Change the path defined in `name` and, for `subfolder`, choose between `append` (add sub-folder name to object name space) and `ignore` (do **not** add sub-folder name to the object name space).

6. Write an application's design-time files to the work file system of the target HDI container.

   The following example shows how to use the HDI's `WRITE` API to create design-time content in HDI container C's "work" file system; a file is created by specifying the file name (with the `PATH` parameter) and its content (in the `CONTENT` parameter) and providing the corresponding data as `VALUES` in the required format:

   ! Restriction

   This step requires the privileges of an HDI container administrator.

   Sample Code

   Write File Content to the Work File System of Container "C"

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PATHS LIKE
_SYS_DI.TT_FILESFOLDERS_CONTENT;
```

```
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('.hdiconfig',
'{ "plugin_version" : "2.0.30.0", "file_suffixes" : { "hdbtable" :
{ "plugin_name" : "com.sap.hana.di.table" }, "hdbview" : { "plugin_name" :
"com.sap.hana.di.view" }, "hdbprocedure" : { "plugin_name" :
"com.sap.hana.di.procedure" } } }');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('.hdinamespace', '{ "name": "",
"subfolder": "ignore" }');
CALL C#DI.WRITE(#PATHS, _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PATHS;
```

7. Deploy an application's design-time files to the target HDI container.

   The following example shows how to use the HDI's `MAKE` API to deploy design-time content to an HDI container. The deployment operation pushes the specified application's database objects from the “work” file system to the “deployed” file system of the target HDI container (“C”). This operation also uses the build plug-in libraries to create the corresponding database objects in the catalog.

   To deploy the newly created files with the `MAKE` API, you must provide the file paths in the `DEPLOY_PATHS` parameter.

   ! Restriction

   This step requires the privileges of an HDI container administrator. Only files can be deployed; it is not possible to deploy folders.

   Sample Code

   Deploy Files to the "Deployed" File System of Container “C”

```
CREATE LOCAL TEMPORARY COLUMN TABLE #DEPLOY_PATHS LIKE
_SYS_DI.TT_FILESFOLDERS;
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('.hdiconfig');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('.hdinamespace');
CREATE LOCAL TEMPORARY COLUMN TABLE #UNDEPLOY_PATHS LIKE
_SYS_DI.TT_FILESFOLDERS;
CREATE LOCAL TEMPORARY COLUMN TABLE #PATH_PARAMETERS LIKE
_SYS_DI.TT_FILESFOLDERS_PARAMETERS;
CALL C#DI.MAKE(#DEPLOY_PATHS, #UNDEPLOY_PATHS, #PATH_PARAMETERS,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #DEPLOY_PATHS;
DROP TABLE #UNDEPLOY_PATHS;
DROP TABLE #PATH_PARAMETERS;
```

Related Information

Set up an SAP HDI Container for XS Advanced Applications [page 133]
Maintaining SAP HDI Container Groups [page 77]
Maintaining SAP HDI Containers [page 90]
The HDI Container API [page 141]

5.3.2 The HDI Container API

Maintain HDI containers and container content using the HDI container API.

In SAP HANA Deployment Infrastructure (HDI), an HDI container administrator is responsible for configuring and controlling access to an HDI container; the SQL application-programming interfaces (APIs) for content development in an HDI container are in the container's corresponding schema. For example, the SQL APIs for the HDI container `C` are in the schema `C#DI`. The following tables indicate which SQL APIs are available for use during content development in an HDI container:

SQL APIs for Content Development in HDI Containers

SQL API	Description
`WRITE`	Writes or creates the specified files or folders in the HDI container's **work** file system.
`DELETE`	Deletes the given files or folders from the HDI container's **work** file system.
`STATUS`	Shows the status of the specified files or folders in the HDI container's **work** file system.
`MAKE`	Triggers a `make` call with the specified set of files or folders; the `make` call returns after the `make` operation has finished
`MAKE_ASYNC`	Triggers a `make` call with the specified set of files or folders; the call returns with a `REQUEST_ID` while the `make` command continues to run in the background
`LIST`	Lists the specified files or folders from the HDI container's **work** file system
`LIST_DEPLOYED`	Lists the specified files or folders from the HDI container's **deployed** file system
`READ`	Reads the specified files or folders from the HDI container's **work** file system
`READ_DEPLOYED`	Reads the specified files or folders from the HDI container's **deployed** file system
`GET_MAKE_GROUPS`	Calculates groups of objects that are independent of each other and could be passed to a `make` call individually
`GET_DEPENDENCIES`	Retrieves dependency data; the kind of data that is retrieved depends on the optional `<variant>` parameter passed via `PARAMETERS`
`LIST_CONFIGURED_LIBRARIES`	Displays a list of all the build plug-in libraries available for use in the specified HDI container
`GRANT_CONTAINER_SCHEMA_PRIVILEGES`	Grants access privileges to a database object consumer for the entire HDI container schema in which the database objects are located

SQL API	Description
REVOKE_CONTAINER_SCHEMA_PRIVILEGES	Revokes access privileges from a database object consumer for the entire container schema in which the database objects are located
GRANT_CONTAINER_SCHEMA_ROLES	Enable access to objects in the container schema by means of a role from the container schema
REVOKE_CONTAINER_SCHEMA_ROLES	Disable access to objects in the container schema by means of a role from the container schema

Related Information

SAP HDI Container Content Development with the HDI API [page 135]
HDI Container Views [page 205]

5.3.2.1 WRITE

Write or create files or folders in an HDI container.

The SAP HDI Container API includes the `WRITE` command, which enables you to create files or folders in an HDI container's **work** file system.

Signature

Sample Code

```
WRITE
 (
  IN   PATHS          _SYS_DI.TT_FILESFOLDERS_CONTENT,
  IN   PARAMETERS     _SYS_DI.TT_PARAMETERS,
  OUT  RETURN_CODE    INT,
  OUT  REQUEST_ID     BIGINT,
  OUT  MESSAGES       _SYS_DI.TT_MESSAGES
 )
```

Parameters

The following parameters can be used with `IN` and `OUT` parameters in the `WRITE` command:

PATHS [IN]

Specifies the list of files or folders to write, including the content.

_SYS_DI.TT_FILESFOLDERS_CONTENT

Name	Data Type	Description
PATH	NVARCHAR(511)	A single path is either a fully qualified file path (for example, `/path/to/a/file.txt`'), or a fully qualified folder path (for example, `/path/to/`).
CONTENT	BLOB	The content of a file is either binary or UTF8 text. The content of a folder **must** be NULL and is ignored during the write operation.

PARAMETERS [IN]

Additional parameters can be used to control various aspects of the procedure execution. If no parameters are needed, the empty predefined parameters table `_SYS_DI.T_NO_PARAMETERS` can be used.

_SYS_DI.TT_PARAMETERS

Name	Data Type	Description
KEY	NVARCHAR(256)	The key name of the parameter
VALUE	NVARCHAR(256)	The value assigned to the parameter

The following additional parameters are available:

- `container_lock_wait_timeout`
- `trace_context`
- `trace_level.<trace topic>`
- `message_severity`

RETURN_CODE [OUT]

The return code indicates if the procedure executed successfully. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

REQUEST_ID [OUT]

A unique ID is generated for each HDI container API call. For more details about which IDs are generated for API calls, see *The SQL API for SAP HDI* in *Related Information*.

MESSAGES [OUT]

A table is used to display messages that contain information logged during (and about) the execution of the procedure. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

Examples

The following example shows how to create files and folders in the work file system of a container named "C":

- Folder: `src1/`
    - Table file: `src1/t.hdbtable`
    - View file: `src1/v.hdbview`
- Folder: `src2/`
    - Procedure file: `src2/p.hdbprocedure`

i Note

For the sake of completeness, the `.hdiconfig` and `.hdinamespace` files are also included in the example. A folder must be specified, only if it does not yet exist.

Sample Code

Write folders and files to an HDI container's work file system

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PATHS LIKE
_SYS_DI.TT_FILESFOLDERS_CONTENT;
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('.hdiconfig',
'{ "plugin_version" : "2.0.0.0", "file_suffixes" : { "hdbtable" :
{ "plugin_name" : "com.sap.hana.di.table" }, "hdbview" : { "plugin_name" :
"com.sap.hana.di.view" }, "hdbprocedure" : { "plugin_name" :
"com.sap.hana.di.procedure" } } }');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('.hdinamespace', '{ "name": "",
"subfolder": "ignore" }');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/', NULL);
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/t.hdbtable', 'COLUMN TABLE T
( A INTEGER )');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/v.hdbview', 'VIEW V AS
SELECT A FROM T');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src2/', '');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src2/p.hdbprocedure', 'PROCEDURE
P (OUT RESULT INT) LANGUAGE SQLSCRIPT AS BEGIN SELECT COUNT(*) INTO RESULT
FROM V; end');
CALL C#DI.WRITE(#PATHS, _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PATHS;
```

To update (overwrite) existing files, you only need to specify the changed files. The following example shows how to update the procedure `src2/p.hdbprocedure`:

Sample Code

Update files in an HDI container's work file system

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PATHS LIKE
_SYS_DI.TT_FILESFOLDERS_CONTENT;
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src2/p.hdbprocedure', 'PROCEDURE
P (OUT RESULT INT)
LANGUAGE SQLSCRIPT AS BEGIN SELECT COUNT(*)+2 INTO RESULT FROM V; end');
CALL C#DI.WRITE(#PATHS, _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PATHS;
```

Related Information

DELETE [page 145]
STATUS [page 148]

5.3.2.2 DELETE

Delete files or folders from an HDI container.

The SAP HDI Container API includes the `DELETE` command, which enables you to remove files or folders from an HDI container's **work** file system.

Signature

Sample Code

```
DELETE
 (
  IN   PATHS         _SYS_DI.TT_FILESFOLDERS,
  IN   PARAMETERS    _SYS_DI.TT_PARAMETERS,
  OUT  RETURN_CODE   INT,
  OUT  REQUEST_ID    BIGINT,
  OUT  MESSAGES      _SYS_DI.TT_MESSAGES
 )
```

Parameters

The following parameters can be used with `IN` and `OUT` parameters in the `DELETE` command:

PATH [IN]

Specifies the list of files or folders to delete.

_SYS_DI.TT_FILESFOLDERS_CONTENT

Name	Data Type	Description
PATH	NVARCHAR(511)	A single path is either a fully qualified file path (for example, `/path/to/a/file.txt`), or a fully qualified folder path (for example, `/path/to/`).

PARAMETERS [IN]

Additional parameters can be used to control various aspects of the procedure execution. If no parameters are needed, the empty predefined parameters table `_SYS_DI.T_NO_PARAMETERS` can be used.

_SYS_DI.TT_PARAMETERS

Name	Data Type	Description
KEY	NVARCHAR(256)	The key name of the parameter
VALUE	NVARCHAR(256)	The value assigned to the parameter

The following additional parameters are available:

- `container_lock_wait_timeout`
- `ignore_non_existing_paths`
- `recursive`
- `trace_context`
- `trace_level.<trace topic>`
- `message_severity`

RETURN_CODE [OUT]

The return code indicates if the procedure executed successfully. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

REQUEST_ID [OUT]

A unique ID is generated for each HDI container API call. For more details about which IDs are generated for API calls, see *The SQL API for SAP HDI* in *Related Information*.

MESSAGES [OUT]

A table is used to display messages that contain information logged during (and about) the execution of the procedure. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

Examples

The following example shows how to **delete** files and folders from the work (design-time) file system of a container named "C" that contains the following content:

- Folder: `src1/`
  - Table file: `src1/t.hdbtable`
  - View file: `src1/v.hdbview`
- Folder: `src2/`
  - Procedure file: `src2/p.hdbprocedure`

i Note

For the sake of completeness, the `.hdiconfig` and `.hdinamespace` files are also included in the following example.

In the following example, we first create the content, and then show how to delete selected files and folders from the created content.

Sample Code

Write folders and files to an HDI container's work file system

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PATHS LIKE
_SYS_DI.TT_FILESFOLDERS_CONTENT;
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('.hdiconfig',
'{ "plugin_version" : "2.0.0.0", "file_suffixes" : { "hdbtable" :
{ "plugin_name" : "com.sap.hana.di.table" }, "hdbview" : { "plugin_name" :
"com.sap.hana.di.view" }, "hdbprocedure" : { "plugin_name" :
"com.sap.hana.di.procedure" } } }');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('.hdinamespace', '{ "name": "",
"subfolder": "ignore" }');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/', NULL);
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/t.hdbtable', 'COLUMN TABLE T
( A INTEGER )');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/v.hdbview', 'VIEW V AS
SELECT A FROM T');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src2/', '');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src2/p.hdbprocedure', 'PROCEDURE
P (OUT RESULT INT) LANGUAGE SQLSCRIPT AS BEGIN SELECT COUNT(*) INTO RESULT
FROM V; end');
CALL C#DI.WRITE(#PATHS, _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PATHS;
```

The following example shows how to delete the view file `src1/v.hdbview` and the folder `src2/` from the content created in container "C".

Note

It is not possible to delete a folder that contains content. Before deleting a folder, you first have to ensure that it is empty, for example, by deleting any files in the folder intended for deletion.

In this example, to remove the folder `src2/`, the file `src2/p.hdbprocedure` has to be deleted, too.

Sample Code

Delete files and folders from an HDI container's work file system

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PATHS LIKE _SYS_DI.TT_FILESFOLDERS;
INSERT INTO #PATHS (PATH) VALUES ('src1/v.hdbview');
INSERT INTO #PATHS (PATH) VALUES ('src2/p.hdbprocedure');
INSERT INTO #PATHS (PATH) VALUES ('src2/');
CALL C#DI.DELETE(#PATHS, _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PATHS;
```

Related Information

WRITE [page 142]
STATUS [page 148]

5.3.2.3 STATUS

Show the status of files or folders in an HDI container.

The SAP HDI Container API includes the `STATUS` command, which enables you to display the status of specified files or folders in either the **work** or **deployed** file systems of an HDI container.

Signature

Sample Code

```
STATUS
 (
  IN   PATHS          _SYS_DI.TT_FILESFOLDERS,
  IN   PARAMETERS     _SYS_DI.TT_PARAMETERS,
  OUT  RETURN_CODE    INT,
  OUT  REQUEST_ID     BIGINT,
  OUT  MESSAGES       _SYS_DI.TT_MESSAGES,
  OUT  RESULT         _SYS_DI.TT_FILESFOLDERS_STATUS
 )
```

Parameters

The following parameters can be used with `IN` and `OUT` parameters in the `STATUS` command:

PATH [IN]

Specifies the list of files or folders whose status you want to check:

_SYS_DI.TT_FILESFOLDERS

Name	Data Type	Description
PATH	NVARCHAR(511)	A single path is either a fully qualified file path (for example, `/path/to/a/file.txt`'), or a fully qualified folder path (for example, `/path/to/`).

PARAMETERS [IN]

Additional parameters can be used to control various aspects of the procedure execution. If no parameters are needed, the empty predefined parameters table `_SYS_DI.T_NO_PARAMETERS` can be used.

_SYS_DI.TT_PARAMETERS

Name	Data Type	Description
KEY	NVARCHAR(256)	The key name of the parameter

Name	Data Type	Description
VALUE	NVARCHAR(256)	The value assigned to the parameter

The following additional parameters are available:

- `container_lock_wait_timeout`
- `trace_context`
- `trace_level.<trace topic>`
- `message_severity`

RETURN_CODE [OUT]

The return code indicates if the procedure executed successfully. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

REQUEST_ID [OUT]

A unique ID is generated for each HDI container API call. For more details about which IDs are generated for API calls, see *The SQL API for SAP HDI* in *Related Information*.

MESSAGES [OUT]

A table is used to display messages that contain information logged during (and about) the execution of the procedure. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

RESULT [OUT]

Returns the result of the operation, with the details specified in the following table:

_SYS_DI.TT_FILESFOLDERS_STATUS

Name	Data Type	Description
PATH	NVARCHAR(511)	A single path is either a fully qualified file path (for example, `/path/to/a/file.txt`'), or a fully qualified folder path (for example, `/path/to/`).
STATUS	NVARCHAR(1)	The status of the file or folder, for example: • U: Unchanged • A: Added • D: Deleted • M: Modified
SHA256	NVARCHAR(64)	SHA256 hash of the file, if present in the **work** file system.
SHA256_DEPLOYED	NVARCHAR(64)	SHA256 hash of the file, if present in the **deployed** file system.

Examples

The following examples show how to use `C#DI.STATUS` command to compare the work (design-time) and deployed file systems in the HDI container "C" and display any differences. The examples show the following steps:

1. Create content in the work file system of HDI container "C" and display the status.
2. Deploy the newly created files and compare the status of the content in the work and deployed file systems of HDI container "C".
3. Change content in the work file system, for example, by modifying and deleting design-time files and compare the status of the content in the work and deployed file systems of HDI container "C".

To see any meaningful information about the status, the respective file systems require some content. In this example, we create and use the following content:

- Folder: `src1/`
  - Table file: `src1/t.hdbtable`
  - View file: `src1/v.hdbview`
- Folder: `src2/`
  - Procedure file: `src2/p.hdbprocedure`

i Note

For the sake of completeness, the `.hdiconfig` and `.hdinamespace` files are also included in the following examples.

Create Content in an HDI container "C" and Display its Status

In the following example, we first create content in HDI container "C':

Sample Code

Write folders and files to the work file system of container C

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PATHS LIKE
_SYS_DI.TT_FILESFOLDERS_CONTENT;
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('.hdiconfig',
'{ "plugin_version" : "2.0.0.0", "file_suffixes" : { "hdbtable" :
{ "plugin_name" : "com.sap.hana.di.table" }, "hdbview" : { "plugin_name" :
"com.sap.hana.di.view" }, "hdbprocedure" : { "plugin_name" :
"com.sap.hana.di.procedure" } } }');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('.hdinamespace', '{ "name": "",
"subfolder": "ignore" }');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/', NULL);
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/t.hdbtable', 'COLUMN TABLE T
( A INTEGER )');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/v.hdbview', 'VIEW V AS
SELECT A FROM T');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src2/', '');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src2/p.hdbprocedure', 'PROCEDURE
P (OUT RESULT INT) LANGUAGE SQLSCRIPT AS BEGIN SELECT COUNT(*) INTO RESULT
FROM V; end');
CALL C#DI.WRITE(#PATHS, _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PATHS;
```

Assuming HDI container C's **deployed** file system is empty, a call to STATUS (without asking for the status of a specific file or folder) will show that three new files have been added to the work file system, as illustrated in the following example:

Sample Code

Status of Content in the Work File System of Container "C"

```
CALL C#DI.STATUS(_SYS_DI.T_NO_FILESFOLDERS,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?, ?);
PATH                   STATUS   SHA256                    SHA256_DEPLOYED
------------------------------------------------------------------------------
---
src1/t.hdbtable        A        fb86b3a6c5e2ac98361d8...
src1/v.hdbview         A        22273fb8a0684f93faa05...
src2/p.hdbprocedure    A        b1b5c23366f73e8c6edc3...
```

Note

The value "A" in the `STATUS` column indicates that a file has been added to the work file system; the files corresponding hash value is displayed in the `SHA256` column.

Deploy Content to an HDI Container and Display its Status

In this example, we deploy the newly created files to the deployed file system in HDI container "C" (for example, using the `MAKE` API).

Sample Code

Deploy Files to the Deployed File System of Container "C"

```
CREATE LOCAL TEMPORARY COLUMN TABLE #DEPLOY_PATHS LIKE
_SYS_DI.TT_FILESFOLDERS;
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('.hdiconfig');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('.hdinamespace');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('src1/t.hdbtable');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('src1/v.hdbview');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('src2/p.hdbprocedure');
CREATE LOCAL TEMPORARY COLUMN TABLE #UNDEPLOY_PATHS LIKE
_SYS_DI.TT_FILESFOLDERS;
CREATE LOCAL TEMPORARY COLUMN TABLE #PATH_PARAMETERS LIKE
_SYS_DI.TT_FILESFOLDERS_PARAMETERS;
CALL C#DI.MAKE(#DEPLOY_PATHS, #UNDEPLOY_PATHS, #PATH_PARAMETERS,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #DEPLOY_PATHS;
DROP TABLE #UNDEPLOY_PATHS;
DROP TABLE #PATH_PARAMETERS;
```

Comparing the content status at this point does not display any results because the content of the work file system in container "C" is identical to the content of container C's deployed file system.

Display Content Status after Modification or Deletion

To demonstrate other status types, in this example we modify one of the files (`src1/t.hdbtable`) and deletion another file (`src2/p.hdbprocedure`) in the HDI container' s design-time work file system.

Sample Code

Modify `src1/t.hdbtable` in the Work File System of Container "C"

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PATHS LIKE
_SYS_DI.TT_FILESFOLDERS_CONTENT
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/t.hdbtable', 'COLUMN TABLE T
( A INTEGER, B INTEGER )');
CALL C#DI.WRITE(#PATHS, _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PATHS;
```

Sample Code

Delete `src2/p.hdbprocedure` from the Work File System of Container "C"

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PATHS LIKE _SYS_DI.TT_FILESFOLDERS;
INSERT INTO #PATHS (PATH) VALUES ('src2/p.hdbprocedure');
CALL C#DI.DELETE(#PATHS, _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PATHS;
```

The `STATUS` API shows deleted files with the status `D` and modified files with the status of `M`.

→ Tip

If files exist in both the work and deployed file system, then a hash value is shown for both file versions, otherwise only a hash value is displayed from the file system where the file is present.

Sample Code

Status of Files in the Work File System of Container "C"

```
CALL C#DI.STATUS(_SYS_DI.T_NO_FILESFOLDERS,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?, ?);
PATH                      STATUS    SHA256                    SHA256_DEPLOYED
------------------------------------------------------------------------------
---
src2/p.hdbprocedure       D
b1b5c23366f73e8c6edc3514...
src1/t.hdbtable           M         13e1efb1fea4d951a2...
fb86b3a6c5e2ac98361d8c71...
```

Since no `PATH` has been specified to the `STATUS` API, the API returns the status for the entire work and deployed file systems. By passing specific file or folder names, only the status for the given `PATHS` is returned. The status for the folder `src1/` returns only the status of all files below the `src1` folder.

Sample Code

Status of the Contents of Folder `src1/` in Container "C"

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PATHS LIKE _SYS_DI.TT_FILESFOLDERS;
INSERT INTO #PATHS (PATH) VALUES ('src1/');
CALL C#DI.STATUS(#PATHS, _SYS_DI.T_NO_PARAMETERS, ?, ?, ?, ?);
DROP TABLE #PATHS;
PATH                     STATUS    SHA256                     SHA256_DEPLOYED
------------------------------------------------------------------------------
---
src1/t.hdbtable          M         13e1efb1fea4d951a2...
fb86b3a6c5e2ac98361d8c71...
```

You can display the status for specific files in an HDI container, for example, `src1/v.hdbview` and `src2/p.hdbprocedure`). Note that results are displayed only for those files that have been changed.

Sample Code

Status of the Specific Files in Container "C"

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PATHS LIKE _SYS_DI.TT_FILESFOLDERS;
INSERT INTO #PATHS (PATH) VALUES ('src1/v.hdbview');
INSERT INTO #PATHS (PATH) VALUES ('src2/p.hdbprocedure');
CALL C#DI.STATUS(#PATHS, _SYS_DI.T_NO_PARAMETERS, ?, ?, ?, ?);
DROP TABLE #PATHS;
PATH                  STATUS   SHA256                 SHA256_DEPLOYED
-------------------------------------------------------------------------------
---
src2/p.hdbprocedure   D
b1b5c23366f73e8c6edc3514...
```

Related Information

WRITE [page 142]
DELETE [page 145]

5.3.2.4 MAKE

Trigger a `make` operation in an HDI container with a specified set of files or folders.

The SAP HDI Container API includes the `MAKE` command, which enables you to start a `make` operation in an HDI container with a specified set of files or folders. The call returns after the `MAKE` operation has finished.

Signature

Sample Code

```
MAKE
 (
  IN   DEPLOY_PATHS      _SYS_DI.TT_FILESFOLDERS,
  IN   UNDEPLOY_PATHS    _SYS_DI.TT_FILESFOLDERS,
  IN   PATH_PARAMETERS   _SYS_DI.TT_FILESFOLDERS_PARAMETERS,
  IN   PARAMETERS        _SYS_DI.TT_PARAMETERS,
  OUT  RETURN_CODE       INT,
  OUT  REQUEST_ID        BIGINT,
  OUT  MESSAGES          _SYS_DI.TT_MESSAGES
 )
```

Parameters

The following parameters can be used with `IN` and `OUT` parameters in the `MAKE` command:

DEPLOY_PATHS [IN]

Specifies a list of files or folders to be deployed:

_SYS_DI.TT_FILESFOLDERS

Name	Data Type	Description
PATH	NVARCHAR(511)	A single path is either a fully qualified file path (for example, `/path/to/a/file.txt`'), or a fully qualified folder path (for example, `/path/to/`).

UNDEPLOY_PATHS [IN]

Specifies a list of files or folders to be **un**deployed:

_SYS_DI.TT_FILESFOLDERS

Name	Data Type	Description
PATH	NVARCHAR(511)	A single path is either a fully qualified file path (for example, `/path/to/a/file.txt`'), or a fully qualified folder path (for example, `/path/to/`).

PATH_PARAMETERS [IN]

The following table lists path parameters you can use with the `MAKE` command at the file or folder level:

_SYS_DI.TT_FILESFOLDERS_PARAMETERS

Name	Data Type	Description
PATH	NVARCHAR(511)	A single path is either a fully qualified file path (for example, `/path/to/a/file.txt`'), or a fully qualified folder path (for example, `/path/to/`).
KEY	NVARCHAR(256)	The key name of the parameter
VALUE	NVARCHAR(256)	The value assigned to the parameter

PARAMETERS [IN]

Additional parameters can be used to control various aspects of the procedure execution. If no parameters are needed, the empty predefined parameters table `_SYS_DI.T_NO_PARAMETERS` can be used.

_SYS_DI.TT_PARAMETERS

Name	Data Type	Description
KEY	NVARCHAR(256)	The key name of the parameter
VALUE	NVARCHAR(256)	The value assigned to the parameter

The following additional parameters are available:

- `container_lock_wait_timeout`
- `max_parallel_jobs`
- `optimized_redeploy`
- `simulate_make`
- `skip_unchanged_expansions`
- `trace_context`
- `trace_level.<trace topic>`
- `treat_warnings_as_errors`
- `undeploy_dependent_recursively`
- `use_redeploy_for_unchanged`
- `enable_make_enforcer`
- `message_severity`

RETURN_CODE [OUT]

The return code indicates if the procedure executed successfully. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

REQUEST_ID [OUT]

A unique ID is generated for each HDI container API call. For more details about which IDs are generated for API calls, see *The SQL API for SAP HDI* in *Related Information*.

MESSAGES [OUT]

A table is used to display messages that contain information logged during (and about) the execution of the procedure. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

Examples

The following examples show how to use the `MAKE` API to deploy files from the work file system to the deployed file system; this operation also creates the corresponding database objects in the catalog. The following content is written to the file system:

- Folder: `src1/`
  - Table file: `src1/t.hdbtable`
  - View file: `src1/v.hdbview`
- Folder: `src2/`

- Procedure file: `src2/p.hdbprocedure`

Note

For the sake of completeness, the `.hdiconfig` and `.hdinamespace` files are also included in the following example.

The following example shows how to create the files in container C's work file system:

Sample Code

Write Files to the Work File System of Container C

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PATHS LIKE
_SYS_DI.TT_FILESFOLDERS_CONTENT;
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('.hdiconfig',
'{ "plugin_version" : "2.0.0.0", "file_suffixes" : { "hdbtable" :
{ "plugin_name" : "com.sap.hana.di.table" }, "hdbview" : { "plugin_name" :
"com.sap.hana.di.view" }, "hdbprocedure" : { "plugin_name" :
"com.sap.hana.di.procedure" } } }');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('.hdinamespace', '{ "name": "",
"subfolder": "ignore" }');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/', NULL);
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/t.hdbtable', 'COLUMN TABLE T
( A INTEGER )');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/v.hdbview', 'VIEW V AS
SELECT A FROM T');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src2/', '');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src2/p.hdbprocedure', 'PROCEDURE
P (OUT RESULT INT) LANGUAGE SQLSCRIPT AS BEGIN SELECT COUNT(*) INTO RESULT
FROM V; end');
CALL C#DI.WRITE(#PATHS, _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PATHS;
```

To deploy the newly created files with the `MAKE` API, you must provide the file paths in the `DEPLOY_PATHS` parameter.

! Restriction

Only files can be deployed; it is not possible to deploy folders.

Sample Code

Deploy Files to the Work File System of Container C

```
CREATE LOCAL TEMPORARY COLUMN TABLE #DEPLOY_PATHS LIKE
_SYS_DI.TT_FILESFOLDERS;
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('.hdiconfig');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('.hdinamespace');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('src1/t.hdbtable');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('src1/v.hdbview');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('src2/p.hdbprocedure');
CREATE LOCAL TEMPORARY COLUMN TABLE #UNDEPLOY_PATHS LIKE
_SYS_DI.TT_FILESFOLDERS;
CREATE LOCAL TEMPORARY COLUMN TABLE #PATH_PARAMETERS LIKE
_SYS_DI.TT_FILESFOLDERS_PARAMETERS;
CALL C#DI.MAKE(#DEPLOY_PATHS, #UNDEPLOY_PATHS, #PATH_PARAMETERS,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #DEPLOY_PATHS;
DROP TABLE #UNDEPLOY_PATHS;
DROP TABLE #PATH_PARAMETERS;
```

To undeploy files with the MAKE API, the file paths must be provided in the UNDEPLOY_PATHS parameter. Undeploying a file will drop the corresponding database object and remove the file from the deployed file system. A STATUS call would then show the file as existing in the work file system only.

! Restriction

Only files can be undeployed; it is not possible to undeploy folders.

Sample Code

Undeploy the File src2/p.hdbprocedure from the Work File System of Container C

```
CREATE LOCAL TEMPORARY COLUMN TABLE #DEPLOY_PATHS LIKE
_SYS_DI.TT_FILESFOLDERS;
CREATE LOCAL TEMPORARY COLUMN TABLE #UNDEPLOY_PATHS LIKE
_SYS_DI.TT_FILESFOLDERS;
INSERT INTO #UNDEPLOY_PATHS (PATH) VALUES ('src2/p.hdbprocedure');
CREATE LOCAL TEMPORARY COLUMN TABLE #PATH_PARAMETERS LIKE
_SYS_DI.TT_FILESFOLDERS_PARAMETERS;
CALL C#DI.MAKE(#DEPLOY_PATHS, #UNDEPLOY_PATHS, #PATH_PARAMETERS,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #DEPLOY_PATHS;
DROP TABLE #UNDEPLOY_PATHS;
DROP TABLE #PATH_PARAMETERS;
```

→ Tip

It is possible to combine the deployment and undeployment of files in the same MAKE call except where the same files are being deployed and undeployed.

Related Information

WRITE [page 142]
MAKE ASYNC [page 157]

5.3.2.5 MAKE ASYNC

Trigger an asynchronous MAKE operation in an HDI container with a specified set of files or folders.

The SAP HDI Container API includes the MAKE ASYNC command, which enables you to start a MAKE operation in asynchronous mode in an HDI container using a specified set of files or folders. The call returns with a REQUEST_ID while the MAKE ASYNC command continues to run in the background.

Signature

Sample Code

```
MAKE ASYNC
 (
  IN    DEPLOY_PATHS     _SYS_DI.TT_FILESFOLDERS,
  IN    UNDEPLOY_PATHS   _SYS_DI.TT_FILESFOLDERS,
  IN    PATH_PARAMETERS  _SYS_DI.TT_FILESFOLDERS_PARAMETERS,
  IN    PARAMETERS       _SYS_DI.TT_PARAMETERS,
  OUT   RETURN_CODE      INT,
  OUT   REQUEST_ID       BIGINT,
  OUT   MESSAGES         _SYS_DI.TT_MESSAGES
 )
```

Parameters

The following parameters can be used with `IN` and `OUT` parameters in the `MAKE ASYNC` command:

DEPLOY_PATHS [IN]

Specifies a list of files or folders to be deployed:

_SYS_DI.TT_FILESFOLDERS

Name	Data Type	Description
PATH	NVARCHAR(511)	A single path is either a fully qualified file path (for example, `/path/to/a/file.txt`'), or a fully qualified folder path (for example, `/path/to/`).

UNDEPLOY_PATHS [IN]

Specifies a list of files or folders to be **un**deployed:

_SYS_DI.TT_FILESFOLDERS

Name	Data Type	Description
PATH	NVARCHAR(511)	A single path is either a fully qualified file path (for example, `/path/to/a/file.txt`'), or a fully qualified folder path (for example, `/path/to/`).

PATH_PARAMETERS [IN]

The following table lists path parameters you can use with the `MAKE ASYNC` command at the file or folder level:

_SYS_DI.TT_FILESFOLDERS_PARAMETERS

Name	Data Type	Description
PATH	NVARCHAR(511)	A single path is either a fully qualified file path (for example, `/path/to/a/file.txt`'), or a fully qualified folder path (for example, `/path/to/`).
KEY	NVARCHAR(256)	The key name of the parameter
VALUE	NVARCHAR(256)	The value assigned to the parameter

PARAMETERS [IN]

Additional parameters can be used to control various aspects of the procedure execution. If no parameters are needed, the empty predefined parameters table `_SYS_DI.T_NO_PARAMETERS` can be used.

_SYS_DI.TT_PARAMETERS

Name	Data Type	Description
KEY	NVARCHAR(256)	The key name of the parameter
VALUE	NVARCHAR(256)	The value assigned to the parameter

The following additional parameters are available:

- `container_lock_wait_timeout`
- `max_parallel_jobs`
- `optimized_redeploy`
- `simulate_make`
- `skip_unchanged_expansions`
- `trace_context`
- `trace_level.<trace topic>`
- `treat_warnings_as_errors`
- `undeploy_dependent_recursively`
- `use_redeploy_for_unchanged`
- `enable_make_enforcer`
- `message_severity`

RETURN_CODE [OUT]

The return code indicates if the procedure executed successfully. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

REQUEST_ID [OUT]

A unique ID is generated for each HDI container API call. For more details about which IDs are generated for API calls, see *The SQL API for SAP HDI* in *Related Information*.

MESSAGES [OUT]

A table is used to display messages that contain information logged during (and about) the execution of the procedure. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

Examples

The following examples show how to use the MAKE_ASYNC API to deploy files from the work file system to the deployed file system; this operation also creates the corresponding database objects in the catalog. The following content is written to the file system:

- Folder: src1/
  - Table file: src1/t.hdbtable
  - View file: src1/v.hdbview
- Folder: src2/
  - Procedure file: src2/p.hdbprocedure

i Note

For the sake of completeness, the .hdiconfig and .hdinamespace files are also included in the following example.

The following example shows how to create the files in container C's work file system:

Sample Code

Write Files to the Work File System of Container C

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PATHS LIKE
_SYS_DI.TT_FILESFOLDERS_CONTENT;
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('.hdiconfig',
'{ "plugin_version" : "2.0.0.0", "file_suffixes" : { "hdbtable" :
{ "plugin_name" : "com.sap.hana.di.table" }, "hdbview" : { "plugin_name" :
"com.sap.hana.di.view" }, "hdbprocedure" : { "plugin_name" :
"com.sap.hana.di.procedure" } } }');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('.hdinamespace', '{ "name": "",
"subfolder": "ignore" }');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/', NULL);
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/t.hdbtable', 'COLUMN TABLE T
( A INTEGER )');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/v.hdbview', 'VIEW V AS
SELECT A FROM T');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src2/', '');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src2/p.hdbprocedure', 'PROCEDURE
P (OUT RESULT INT) LANGUAGE SQLSCRIPT AS BEGIN SELECT COUNT(*) INTO RESULT
FROM V; end');
CALL C#DI.WRITE(#PATHS, _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PATHS;
```

To deploy the newly created files with the MAKE_ASYNC API, you must provide the file paths in the DEPLOY_PATHS parameter.

! Restriction

Only files can be deployed; it is not possible to deploy folders.

Sample Code

Deploy Files to the Work File System of Container C

```
CREATE LOCAL TEMPORARY COLUMN TABLE #DEPLOY_PATHS LIKE
_SYS_DI.TT_FILESFOLDERS;
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('.hdiconfig');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('.hdinamespace');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('src1/t.hdbtable');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('src1/v.hdbview');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('src2/p.hdbprocedure');
CREATE LOCAL TEMPORARY COLUMN TABLE #UNDEPLOY_PATHS LIKE
_SYS_DI.TT_FILESFOLDERS;
CREATE LOCAL TEMPORARY COLUMN TABLE #PATH_PARAMETERS LIKE
_SYS_DI.TT_FILESFOLDERS_PARAMETERS;
CALL C#DI.MAKE_ASYNC(#DEPLOY_PATHS, #UNDEPLOY_PATHS, #PATH_PARAMETERS,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #DEPLOY_PATHS;
DROP TABLE #UNDEPLOY_PATHS;
DROP TABLE #PATH_PARAMETERS;
```

To undeploy files with the `MAKE_ASYNC` API, the file paths must be provided in the `UNDEPLOY_PATHS` parameter. Undeploying a file will drop the corresponding database object and remove the file from the deployed file system. A `STATUS` call would then show the file as existing in the work file system only.

! Restriction

Only files can be undeployed; it is not possible to undeploy folders.

Sample Code

Undeploy the File `src2/p.hdbprocedure` from the Work File System of Container C

```
CREATE LOCAL TEMPORARY COLUMN TABLE #DEPLOY_PATHS LIKE
_SYS_DI.TT_FILESFOLDERS;
CREATE LOCAL TEMPORARY COLUMN TABLE #UNDEPLOY_PATHS LIKE
_SYS_DI.TT_FILESFOLDERS;
INSERT INTO #UNDEPLOY_PATHS (PATH) VALUES ('src2/p.hdbprocedure');
CREATE LOCAL TEMPORARY COLUMN TABLE #PATH_PARAMETERS LIKE
_SYS_DI.TT_FILESFOLDERS_PARAMETERS;
CALL C#DI.MAKE_ASYNC(#DEPLOY_PATHS, #UNDEPLOY_PATHS, #PATH_PARAMETERS,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #DEPLOY_PATHS;
DROP TABLE #UNDEPLOY_PATHS;
DROP TABLE #PATH_PARAMETERS;
```

→ Tip

It is possible to combine the deployment and undeployment of files in the same `MAKE_ASYNC` call except where the same files are being deployed and undeployed.

Related Information

WRITE [page 142]

MAKE [page 153]

5.3.2.6 LIST

Display specific files or folders in an HDI container.

The SAP HDI Container API includes the `LIST` command, which enables you to display a list of files or folders in an HDI container's **work** file system.

Signature

Sample Code

```
LIST
 (
   IN    PATHS          _SYS_DI.TT_FILESFOLDERS,
   IN    PARAMETERS     _SYS_DI.TT_PARAMETERS,
   OUT   RETURN_CODE    INT,
   OUT   REQUEST_ID     BIGINT,
   OUT   MESSAGES       _SYS_DI.TT_MESSAGES,
   OUT   RESULT         _SYS_DI.TT_FILESFOLDERS_METADATA
 )
```

Parameters

The following parameters can be used with `IN` and `OUT` parameters in the `LIST` command:

PATHS [IN]

Specifies a list of files or folders to list:

_SYS_DI.TT_FILESFOLDERS

Name	Data Type	Description
PATH	NVARCHAR(511)	A single path is either a fully qualified path to a **work** file (for example, `/path/to/a/file.txt`'), or a fully qualified path to a **work** folder (for example, `/path/to/`).

PARAMETERS [IN]

Additional parameters can be used to control various aspects of the procedure execution. If no parameters are needed, the empty predefined parameters table `_SYS_DI.T_NO_PARAMETERS` can be used.

_SYS_DI.TT_PARAMETERS

Name	Data Type	Description
KEY	NVARCHAR(256)	The key name of the parameter

Name	Data Type	Description
VALUE	NVARCHAR(256)	The value assigned to the parameter

The following additional parameters are available:

- `container_lock_wait_timeout`
- `ignore_files` true, false (default)
- `ignore_folders` true, false (default)
- `recursive` true, false (default)
- `trace_context`
- `trace_level.<trace topic>`
- `message_severity`

RETURN_CODE [OUT]

The return code indicates if the procedure executed successfully. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

REQUEST_ID [OUT]

A unique ID is generated for each HDI container API call. For more details about which IDs are generated for API calls, see *The SQL API for SAP HDI* in *Related Information*.

MESSAGES [OUT]

A table is used to display messages that contain information logged during (and about) the execution of the procedure. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

RESULT [OUT]

Returns a list of files or folders including metadata:

_SYS_DI.TT_FILESFOLDERS_METADATA

Name	Data Type	Description
PATH	NVARCHAR(511)	A single path is either a fully qualified path to the **work** file (for example, `/path/to/a/file.txt`'), or a fully qualified path to a **work** folder (for example,. `/path/to/`).
CREATE_USER_NAME	NVARCHAR(256)	User associated with the creation of the **work** file or folder
CREATE_APPUSER_NAME	NVARCHAR(256)	Application user associated with the creation of the **work** file or folder
CREATE_TIMESTAMP_UTC	TIMESTAMP	Time stamp indicating when the **work** file or folder was created
MODIFICATION_USER_NAME	NVARCHAR(256)	User associated with the last modification of the **work** file or folder

Name	Data Type	Description
MODIFICATION_AP-PUSER_NAME	NVARCHAR(256)	Application user associated with the last modification of the **work** file or folder
MODIFICATION_TIME-STAMP_UTC	TIMESTAMP	Time stamp indicating when the **work** file or folder was last modified
SIZE	BIGINT	Size of the listed **work** file in bytes
SHA256	NVARCHAR(64)	SHA256 hash of the file, if present in the **work** file system.

Examples

The following examples show how to use the `LIST` API to obtain and display information about the content of the work file system. The following content is written to the file system:

- Configuration File: `.hdiconfig`
- Configuration File: `.hdinamespace`
- Folder: `src1/`
  - Table file: `src1/t.hdbtable`
  - View file: `src1/v.hdbview`
- Folder: `src2/`
  - Procedure file: `src2/p.hdbprocedure`

The following example shows how to create the files in container C's work file system:

Sample Code

Write Files to the Work File System of Container C

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PATHS LIKE
_SYS_DI.TT_FILESFOLDERS_CONTENT;
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('.hdiconfig',
'{ "plugin_version" : "2.0.0.0", "file_suffixes" : { "hdbtable" :
{ "plugin_name" : "com.sap.hana.di.table" }, "hdbview" : { "plugin_name" :
"com.sap.hana.di.view" }, "hdbprocedure" : { "plugin_name" :
"com.sap.hana.di.procedure" } } }');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('.hdinamespace', '{ "name": "",
"subfolder": "ignore" }');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/', NULL);
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/t.hdbtable', 'COLUMN TABLE T
( A INTEGER )');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/v.hdbview', 'VIEW V AS
SELECT A FROM T');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src2/', '');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src2/p.hdbprocedure', 'PROCEDURE
P (OUT RESULT INT) LANGUAGE SQLSCRIPT AS BEGIN SELECT COUNT(*) INTO RESULT
FROM V; end');
CALL C#DI.WRITE(#PATHS, _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PATHS;
```

List non-recursively all files and folders in the work file system of container C; this displays objects only at the topmost level:

Sample Code

List Files and Folders Non-Recursively

```
CALL C#DI.LIST(_SYS_DI.T_NO_FILESFOLDERS,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?, ?);
```

List recursively all files and folders in the work file system of container C; this displays all objects at all levels:

Sample Code

List Files and Folders Recursively

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PARAMETERS LIKE _SYS_DI.TT_PARAMETERS;
INSERT INTO #PARAMETERS ( KEY, VALUE ) VALUES ('recursive', 'true');
CALL C#DI.LIST(_SYS_DI.T_NO_FILESFOLDERS, #PARAMETERS, ?, ?, ?, ?);
DROP TABLE #PARAMETERS;
```

List recursively all files (but no folders) in the work file system of container C:

Sample Code

List Only Files Recursively

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PARAMETERS LIKE _SYS_DI.TT_PARAMETERS;
INSERT INTO #PARAMETERS ( KEY, VALUE ) VALUES ('ignore_folders', 'true');
INSERT INTO #PARAMETERS ( KEY, VALUE ) VALUES ('recursive', 'true');
CALL C#DI.LIST(_SYS_DI.T_NO_FILESFOLDERS, #PARAMETERS, ?, ?, ?, ?);
DROP TABLE #PARAMETERS;
```

List recursively all folders (but no files) in the work file system of container C:

Sample Code

List Only Folders Recursively

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PARAMETERS LIKE _SYS_DI.TT_PARAMETERS;
INSERT INTO #PARAMETERS ( KEY, VALUE ) VALUES ('ignore_files', 'true');
INSERT INTO #PARAMETERS ( KEY, VALUE ) VALUES ('recursive', 'true');
CALL C#DI.LIST(_SYS_DI.T_NO_FILESFOLDERS, #PARAMETERS, ?, ?, ?, ?);
DROP TABLE #PARAMETERS;
```

List specific files or folders in the work file system of container C:

Sample Code

List Specified Files and Folders

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PATHS LIKE _SYS_DI.TT_FILESFOLDERS;
INSERT INTO #PATHS (PATH) VALUES ('src1/');
INSERT INTO #PATHS (PATH) VALUES ('src2/p.hdbprocedure');
CREATE LOCAL TEMPORARY COLUMN TABLE #PARAMETERS LIKE _SYS_DI.TT_PARAMETERS;
INSERT INTO #PARAMETERS ( KEY, VALUE ) VALUES ('recursive', 'false');
CALL C#DI.LIST(#PATHS, #PARAMETERS, ?, ?, ?, ?);
DROP TABLE #PATHS; DROP TABLE #PARAMETERS;
```

Related Information

WRITE [page 142]
READ [page 170]
LIST_DEPLOYED [page 166]

5.3.2.7 LIST_DEPLOYED

Display specific files or folders in an HDI container.

The SAP HDI Container API includes the `LIST_DEPLOYED` command, which enables you to display a list of files or folders in the **deployed** file system of an HDI container.

Signature

Sample Code

```
LIST_DEPLOYED
 (
  IN    PATHS          _SYS_DI.TT_FILESFOLDERS,
  IN    PARAMETERS     _SYS_DI.TT_PARAMETERS,
  OUT   RETURN_CODE   INT,
  OUT   REQUEST_ID     BIGINT,
  OUT   MESSAGES       _SYS_DI.TT_MESSAGES,
  OUT   RESULT         _SYS_DI.TT_FILESFOLDERS_METADATA
 )
```

Parameters

The following parameters can be used with `IN` and `OUT` parameters in the `LIST_DEPLOYED` command:

PATHS [IN]

Specifies a list of files or folders to list:

_SYS_DI.TT_FILESFOLDERS

Name	Data Type	Description
PATH	NVARCHAR(511)	A single path is either a fully qualified path to a **deployed** file (for example, `/path/to/a/file.txt`'), or a fully qualified path to a **deployed** folder (for example, `/path/to/`).

PARAMETERS [IN]

Additional parameters can be used to control various aspects of the procedure execution. If no parameters are needed, the empty predefined parameters table `_SYS_DI.T_NO_PARAMETERS` can be used.

_SYS_DI.TT_PARAMETERS

Name	Data Type	Description
KEY	NVARCHAR(256)	The key name of the parameter
VALUE	NVARCHAR(256)	The value assigned to the parameter

The following additional parameters are available:

- `container_lock_wait_timeout`
- `ignore_files` true, false (default)
- `ignore_folders` true, false (default)
- `recursive` true, false (default)
- `trace_context`
- `trace_level.<trace topic>`
- `message_severity`

RETURN_CODE [OUT]

The return code indicates if the procedure executed successfully. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

REQUEST_ID [OUT]

A unique ID is generated for each HDI container API call. For more details about which IDs are generated for API calls, see *The SQL API for SAP HDI* in *Related Information*.

MESSAGES [OUT]

A table is used to display messages that contain information logged during (and about) the execution of the procedure. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

RESULT [OUT]

Returns a list of files or folders including metadata:

_SYS_DI.TT_FILESFOLDERS_METADATA

Name	Data Type	Description
PATH	NVARCHAR(511)	A single path is either a fully qualified path to a **deployed** file (for example, `/path/to/a/file.txt`'), or a fully qualified path to a deployed **folder** (for example,. `/path/to/`).
CREATE_USER_NAME	NVARCHAR(256)	User associated with the creation of the **deployed** file or folder

Name	Data Type	Description
CREATE_APPUSER_NAME	NVARCHAR(256)	Application user associated with the creation of the **deployed** file or folder
CREATE_TIMESTAMP_UTC	TIMESTAMP	Time stamp indicating when the **deployed** file or folder was created
MODIFICATION_USER_NAME	NVARCHAR(256)	User associated with the last modification of the **deployed** file or folder
MODIFICATION_APPUSER_NAME	NVARCHAR(256)	Application user associated with the last modification of the **deployed** file or folder
MODIFICATION_TIMESTAMP_UTC	TIMESTAMP	Time stamp indicating when the **deployed** file or folder was last modified
SIZE	BIGINT	Size of the listed **deployed** file in bytes
SHA256	NVARCHAR(64)	SHA256 hash of the **deployed** file, if present in the **work** file system.

Examples

The following examples show how to use the `LIST_DEPLOYED` API to obtain and display information about the content of the deployed file system, after writing the following content to the file system using the `MAKE` API:

- Configuration File: `.hdiconfig`
- Configuration File: `.hdinamespace`
- Folder: `src1/`
  - Table file: `src1/t.hdbtable`
  - View file: `src1/v.hdbview`
- Folder: `src2/`
  - Procedure file: `src2/p.hdbprocedure`

The following example shows how to create the files in container C's work file system:

Sample Code

Write Files to the Container's Work File System

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PATHS LIKE
_SYS_DI.TT_FILESFOLDERS_CONTENT;
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('.hdiconfig',
'{ "plugin_version" : "2.0.0.0", "file_suffixes" : { "hdbtable" :
{ "plugin_name" : "com.sap.hana.di.table" }, "hdbview" : { "plugin_name" :
"com.sap.hana.di.view" }, "hdbprocedure" : { "plugin_name" :
"com.sap.hana.di.procedure" } } }');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('.hdinamespace', '{ "name": "",
"subfolder": "ignore" }');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/', NULL);
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/t.hdbtable', 'COLUMN TABLE T
( A INTEGER )');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/v.hdbview', 'VIEW V AS
SELECT A FROM T');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src2/', '');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src2/p.hdbprocedure', 'PROCEDURE
P (OUT RESULT INT) LANGUAGE SQLSCRIPT AS BEGIN SELECT COUNT(*) INTO RESULT
FROM V; end');
CALL C#DI.WRITE(#PATHS, _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PATHS;
```

Deploy the files to container C's deployed file system:

Sample Code

Deploy Files to the Container's

```
CREATE LOCAL TEMPORARY COLUMN TABLE #DEPLOY_PATHS LIKE
_SYS_DI.TT_FILESFOLDERS;
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('.hdiconfig');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('.hdinamespace');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('src1/t.hdbtable');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('src1/v.hdbview');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('src2/p.hdbprocedure');
CREATE LOCAL TEMPORARY COLUMN TABLE #UNDEPLOY_PATHS LIKE
_SYS_DI.TT_FILESFOLDERS;
CREATE LOCAL TEMPORARY COLUMN TABLE #PATH_PARAMETERS LIKE
_SYS_DI.TT_FILESFOLDERS_PARAMETERS;
CALL C#DI.MAKE(#DEPLOY_PATHS, #UNDEPLOY_PATHS, #PATH_PARAMETERS,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #DEPLOY_PATHS;
DROP TABLE #UNDEPLOY_PATHS;
DROP TABLE #PATH_PARAMETERS;
```

List non-recursively all files and folders in the deployed file system of container C; this displays objects only at the topmost level:

Sample Code

List Files and Folders Non-Recursively

```
CALL C#DI.LIST_DEPLOYED(_SYS_DI.T_NO_FILESFOLDERS,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?, ?);
```

List recursively all files and folders in the deployed file system of container C; this displays all objects at all levels:

Sample Code

List Files and Folders Recursively

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PARAMETERS LIKE _SYS_DI.TT_PARAMETERS;
INSERT INTO #PARAMETERS ( KEY, VALUE ) VALUES ('recursive', 'true');
CALL C#DI.LIST_DEPLOYED(_SYS_DI.T_NO_FILESFOLDERS, #PARAMETERS, ?, ?, ?, ?);
DROP TABLE #PARAMETERS;
```

List recursively all files (but no folders) in the deployed file system of container C:

Sample Code

List Only Files Recursively

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PARAMETERS LIKE _SYS_DI.TT_PARAMETERS;
INSERT INTO #PARAMETERS ( KEY, VALUE ) VALUES ('ignore_folders', 'true');
INSERT INTO #PARAMETERS ( KEY, VALUE ) VALUES ('recursive', 'true');
CALL C#DI.LIST_DEPLOYED(_SYS_DI.T_NO_FILESFOLDERS, #PARAMETERS, ?, ?, ?, ?);
DROP TABLE #PARAMETERS;
```

List recursively all folders (but no files) in the deployed file system of container C:

Sample Code

List Only Files Recursively

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PARAMETERS LIKE _SYS_DI.TT_PARAMETERS;
INSERT INTO #PARAMETERS ( KEY, VALUE ) VALUES ('ignore_files', 'true');
INSERT INTO #PARAMETERS ( KEY, VALUE ) VALUES ('recursive', 'true');
CALL C#DI.LIST_DEPLOYED(_SYS_DI.T_NO_FILESFOLDERS, #PARAMETERS, ?, ?, ?, ?);
DROP TABLE #PARAMETERS;
```

List specific files or folders in the deployed file system of container C:

Sample Code

List Specified Files and Folders

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PATHS LIKE _SYS_DI.TT_FILESFOLDERS;
INSERT INTO #PATHS (PATH) VALUES ('src1/');
INSERT INTO #PATHS (PATH) VALUES ('src2/p.hdbprocedure');
CREATE LOCAL TEMPORARY COLUMN TABLE #PARAMETERS LIKE _SYS_DI.TT_PARAMETERS;
INSERT INTO #PARAMETERS ( KEY, VALUE ) VALUES ('recursive', 'false');
CALL C#DI.LIST_DEPLOYED(#PATHS, #PARAMETERS, ?, ?, ?, ?);
DROP TABLE #PATHS; DROP TABLE #PARAMETERS;
```

Related Information

WRITE [page 142]
MAKE [page 153]
READ_DEPLOYED [page 175]

5.3.2.8 READ

Reads the specified files or folders from an HDI container's **work** file system.

The SAP HDI Container API includes the `READ` command, which enables you to read a list of files or folders from an HDI container's **work** file system.

Signature

Sample Code

```
READ
 (
  IN   PATHS         _SYS_DI.TT_FILESFOLDERS,
  IN   PARAMETERS    _SYS_DI.TT_PARAMETERS,
  OUT  RETURN_CODE   INT,
  OUT  REQUEST_ID    BIGINT,
  OUT  MESSAGES      _SYS_DI.TT_MESSAGES,
  OUT  RESULT        _SYS_DI.TT_FILESFOLDERS_METADATA_CONTENT
 )
```

Parameters

The following parameters can be used with `IN` and `OUT` parameters in the `READ` command:

PATHS [IN]

Specifies a list of **work** files or folders to read:

_SYS_DI.TT_FILESFOLDERS

Name	Data Type	Description
PATH	NVARCHAR(511)	A single path is either a fully qualified path to a **work** file (for example, `/path/to/a/file.txt`'), or a fully qualified path to a **work** folder (for example, `/path/to/`).

PARAMETERS [IN]

Additional parameters can be used to control various aspects of the procedure execution. If no parameters are needed, the empty predefined parameters table `_SYS_DI.T_NO_PARAMETERS` can be used.

_SYS_DI.TT_PARAMETERS

Name	Data Type	Description
KEY	NVARCHAR(256)	The key name of the parameter
VALUE	NVARCHAR(256)	The value assigned to the parameter

The following additional parameters are available:

- `container_lock_wait_timeout`
- `ignore_files` true, false (default)
- `ignore_folders` true, false (default)
- `recursive` true, false (default)
- `trace_context`

- `trace_level.<trace topic>`
- `message_severity`

RETURN_CODE [OUT]

The return code indicates if the procedure executed successfully. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

REQUEST_ID [OUT]

A unique ID is generated for each HDI container API call. For more details about which IDs are generated for API calls, see *The SQL API for SAP HDI* in *Related Information*.

MESSAGES [OUT]

A table is used to display messages that contain information logged during (and about) the execution of the procedure. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

RESULT [OUT]

Returns a list of **work** files or folders including metadata:

_SYS_DI.TT_FILESFOLDERS_METADATA

Name	Data Type	Description
PATH	NVARCHAR(511)	A single path is either a fully qualified path to the **work** file (for example, `/path/to/a/file.txt`'), or a fully qualified path to a **work** folder (for example, `/path/to/`).
CREATE_USER_NAME	NVARCHAR(256)	User associated with the creation of the **work** file or folder
CREATE_APPUSER_NAME	NVARCHAR(256)	Application user associated with the creation of the **work** file or folder
CREATE_TIMESTAMP_UTC	TIMESTAMP	Time stamp indicating when the **work** file or folder was created
MODIFICATION_USER_NAME	NVARCHAR(256)	User associated with the last modification of the **work** file or folder
MODIFICATION_APPUSER_NAME	NVARCHAR(256)	Application user associated with the last modification of the **work** file or folder
MODIFICATION_TIMESTAMP_UTC	TIMESTAMP	Time stamp indicating when the **work** file or folder was last modified
SIZE	BIGINT	Size of the listed **work** file in bytes
SHA256	NVARCHAR(64)	SHA256 hash of the file, if present in the **work** file system.
CONTENT	BLOB	Content of the **work** file

Examples

The following examples show how to use the `READ` API to obtain and display information about the content of the work file system after writing the following content to the work file system:

- Configuration File: `.hdiconfig`
- Configuration File: `.hdinamespace`
- Folder: `src1/`
  - Table file: `src1/t.hdbtable`
  - View file: `src1/v.hdbview`
- Folder: `src2/`
  - Procedure file: `src2/p.hdbprocedure`

The following example shows how to create the files in container C's work file system:

Sample Code

Write Files to the Work File System of Container C

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PATHS LIKE
_SYS_DI.TT_FILESFOLDERS_CONTENT;
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('.hdiconfig',
'{ "plugin_version" : "2.0.0.0", "file_suffixes" : { "hdbtable" :
{ "plugin_name" : "com.sap.hana.di.table" }, "hdbview" : { "plugin_name" :
"com.sap.hana.di.view" }, "hdbprocedure" : { "plugin_name" :
"com.sap.hana.di.procedure" } } }');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('.hdinamespace', '{ "name": "",
"subfolder": "ignore" }');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/', NULL);
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/t.hdbtable', 'COLUMN TABLE T
( A INTEGER )');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/v.hdbview', 'VIEW V AS
SELECT A FROM T');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src2/', '');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src2/p.hdbprocedure', 'PROCEDURE
P (OUT RESULT INT) LANGUAGE SQLSCRIPT AS BEGIN SELECT COUNT(*) INTO RESULT
FROM V; end');
CALL C#DI.WRITE(#PATHS, _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PATHS;
```

Read non-recursively all files and folders in the work file system of container C; this displays objects only at the topmost level:

Sample Code

Read Files and Folders Non-Recursively

```
CALL C#DI.READ(_SYS_DI.T_NO_FILESFOLDERS,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?, ?);
```

Read recursively all files and folders in the work file system of container C; this displays all objects at all levels:

Sample Code

Read Files and Folders Recursively

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PARAMETERS LIKE _SYS_DI.TT_PARAMETERS;
```

```
INSERT INTO #PARAMETERS ( KEY, VALUE ) VALUES ('recursive', 'true');
CALL C#DI.READ(_SYS_DI.T_NO_FILESFOLDERS, #PARAMETERS, ?, ?, ?, ?);
DROP TABLE #PARAMETERS;
```

Read recursively all files (but no folders) in the work file system of container C:

Sample Code

Read Only Files Recursively

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PARAMETERS LIKE _SYS_DI.TT_PARAMETERS;
INSERT INTO #PARAMETERS ( KEY, VALUE ) VALUES ('ignore_folders', 'true');
INSERT INTO #PARAMETERS ( KEY, VALUE ) VALUES ('recursive', 'true');
CALL C#DI.READ(_SYS_DI.T_NO_FILESFOLDERS, #PARAMETERS, ?, ?, ?, ?);
DROP TABLE #PARAMETERS;
```

Read recursively all folders (but no files) in the work file system of container C:

Sample Code

Read Only Folders Recursively

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PARAMETERS LIKE _SYS_DI.TT_PARAMETERS;
INSERT INTO #PARAMETERS ( KEY, VALUE ) VALUES ('ignore_files', 'true');
INSERT INTO #PARAMETERS ( KEY, VALUE ) VALUES ('recursive', 'true');
CALL C#DI.READ(_SYS_DI.T_NO_FILESFOLDERS, #PARAMETERS, ?, ?, ?, ?);
DROP TABLE #PARAMETERS;
```

Read specific files or folders in the work file system of container C:

Sample Code

Read Specified Files and Folders

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PATHS LIKE _SYS_DI.TT_FILESFOLDERS;
INSERT INTO #PATHS (PATH) VALUES ('src1/');
INSERT INTO #PATHS (PATH) VALUES ('src2/p.hdbprocedure');
CREATE LOCAL TEMPORARY COLUMN TABLE #PARAMETERS LIKE _SYS_DI.TT_PARAMETERS;
INSERT INTO #PARAMETERS ( KEY, VALUE ) VALUES ('recursive', 'false');
CALL C#DI.READ(#PATHS, #PARAMETERS, ?, ?, ?, ?);
DROP TABLE #PATHS; DROP TABLE #PARAMETERS;
```

Related Information

WRITE [page 142]
LIST [page 162]

5.3.2.9 READ_DEPLOYED

Read the specified deployed files or folders from an HDI container

The SAP HDI Container API includes the `READ_DEPLOYED` command, which enables you to read specific files or folders from the **deployed** file system of an HDI container.

Signature

Sample Code

```
READ_DEPLOYED
 (
  IN   PATHS         _SYS_DI.TT_FILESFOLDERS,
  IN   PARAMETERS    _SYS_DI.TT_PARAMETERS,
  OUT  RETURN_CODE   INT,
  OUT  REQUEST_ID    BIGINT,
  OUT  MESSAGES      _SYS_DI.TT_MESSAGES,
  OUT  RESULT        _SYS_DI.TT_FILESFOLDERS_METADATA_CONTENT
 )
```

Parameters

The following parameters can be used with `IN` and `OUT` parameters in the `READ_DEPLOYED` command:

PATHS [IN]

Specifies a list of **deployed** files or folders to read:

_SYS_DI.TT_FILESFOLDERS

Name	Data Type	Description
PATH	NVARCHAR(511)	A single path is either a fully qualified path to a **deployed** file (for example, `/path/to/a/file.txt`'), or a fully qualified path to a **deployed** folder (for example, `/path/to/`).

PARAMETERS [IN]

Additional parameters can be used to control various aspects of the procedure execution. If no parameters are needed, the empty predefined parameters table `_SYS_DI.T_NO_PARAMETERS` can be used.

_SYS_DI.TT_PARAMETERS

Name	Data Type	Description
KEY	NVARCHAR(256)	The key name of the parameter

Name	Data Type	Description
VALUE	NVARCHAR(256)	The value assigned to the parameter

The following additional parameters are available:

- `container_lock_wait_timeout`
- `ignore_files` true, false (default)
- `ignore_folders` true, false (default)
- `recursive` true, false (default)
- `trace_context`
- `trace_level.<trace topic>`
- `message_severity`

RETURN_CODE [OUT]

The return code indicates if the procedure executed successfully. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

REQUEST_ID [OUT]

A unique ID is generated for each HDI container API call. For more details about which IDs are generated for API calls, see *The SQL API for SAP HDI* in *Related Information*.

MESSAGES [OUT]

A table is used to display messages that contain information logged during (and about) the execution of the procedure. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

RESULT [OUT]

Returns a list of files or folders including metadata:

_SYS_DI.TT_FILESFOLDERS_METADATA

Name	Data Type	Description
PATH	NVARCHAR(511)	A single path is either a fully qualified path to the **deployed** file (for example, `/path/to/a/file.txt`'), or a fully qualified path to a **deployed** folder (for example, `/path/to/`).
CREATE_USER_NAME	NVARCHAR(256)	User associated with the creation of the **deployed** file or folder
CREATE_APPUSER_NAME	NVARCHAR(256)	Application user associated with the creation of the **deployed** file or folder
CREATE_TIMESTAMP_UTC	TIMESTAMP	Time stamp indicating when the **deployed** file or folder was cre-ated
MODIFICA-TION_USER_NAME	NVARCHAR(256)	User associated with the last modification of the **deployed** file or folder

Name	Data Type	Description
MODIFICATION_AP-PUSER_NAME	NVARCHAR(256)	Application user associated with the last modification of the **de-ployed** file or folder
MODIFICATION_TIME-STAMP_UTC	TIMESTAMP	Time stamp indicating when the **deployed** file or folder was last modified
SIZE	BIGINT	Size of the listed **deployed** file in bytes
SHA256	NVARCHAR(64)	SHA256 hash of the file, if present in the **deployed** file system.
CONTENT	BLOB	Content of the **deployed** file

Examples

The following examples show how to use the `READ_DEPLOYED` API to obtain and display information about the content of the deployed file system, after writing the following content to the file system using the `MAKE` API:

- Configuration File: `.hdiconfig`
- Configuration File: `.hdinamespace`
- Folder: `src1/`
  - Table file: `src1/t.hdbtable`
  - View file: `src1/v.hdbview`
- Folder: `src2/`
  - Procedure file: `src2/p.hdbprocedure`

The following example shows how to create the files in container C's work file system:

Sample Code

Write Files to the Container's Work File System

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PATHS LIKE
_SYS_DI.TT_FILESFOLDERS_CONTENT;
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('.hdiconfig',
'{ "plugin_version" : "2.0.0.0", "file_suffixes" : { "hdbtable" :
{ "plugin_name" : "com.sap.hana.di.table" }, "hdbview" : { "plugin_name" :
"com.sap.hana.di.view" }, "hdbprocedure" : { "plugin_name" :
"com.sap.hana.di.procedure" } } }');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('.hdinamespace', '{ "name": "",
"subfolder": "ignore" }');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/', NULL);
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/t.hdbtable', 'COLUMN TABLE T
( A INTEGER )');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/v.hdbview', 'VIEW V AS
SELECT A FROM T');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src2/', '');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src2/p.hdbprocedure', 'PROCEDURE
P (OUT RESULT INT) LANGUAGE SQLSCRIPT AS BEGIN SELECT COUNT(*) INTO RESULT
FROM V; end');
CALL C#DI.WRITE(#PATHS, _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PATHS;
```

Deploy the files to container C's deployed file system:

Sample Code

Deploy Files to the Container's File System

```
CREATE LOCAL TEMPORARY COLUMN TABLE #DEPLOY_PATHS LIKE
_SYS_DI.TT_FILESFOLDERS;
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('.hdiconfig');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('.hdinamespace');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('src1/t.hdbtable');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('src1/v.hdbview');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('src2/p.hdbprocedure');
CREATE LOCAL TEMPORARY COLUMN TABLE #UNDEPLOY_PATHS LIKE
_SYS_DI.TT_FILESFOLDERS;
CREATE LOCAL TEMPORARY COLUMN TABLE #PATH_PARAMETERS LIKE
_SYS_DI.TT_FILESFOLDERS_PARAMETERS;
CALL C#DI.MAKE(#DEPLOY_PATHS, #UNDEPLOY_PATHS, #PATH_PARAMETERS,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #DEPLOY_PATHS;
DROP TABLE #UNDEPLOY_PATHS;
DROP TABLE #PATH_PARAMETERS;
```

Read non-recursively all files and folders in the deployed file system of container C; this displays objects only at the topmost level:

Sample Code

Read Files and Folders Non-Recursively

```
CALL C#DI.READ_DEPLOYED(_SYS_DI.T_NO_FILESFOLDERS,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?, ?);
```

Read recursively all files and folders in the deployed file system of container C; this displays all objects at all levels:

Sample Code

Read Files and Folders Recursively

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PARAMETERS LIKE _SYS_DI.TT_PARAMETERS;
INSERT INTO #PARAMETERS ( KEY, VALUE ) VALUES ('recursive', 'true');
CALL C#DI.READ_DEPLOYED(_SYS_DI.T_NO_FILESFOLDERS, #PARAMETERS, ?, ?, ?, ?);
DROP TABLE #PARAMETERS;
```

Read recursively all files (but no folders) in the deployed file system of container C:

Sample Code

Read Only Files Recursively

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PARAMETERS LIKE _SYS_DI.TT_PARAMETERS;
INSERT INTO #PARAMETERS ( KEY, VALUE ) VALUES ('ignore_folders', 'true');
INSERT INTO #PARAMETERS ( KEY, VALUE ) VALUES ('recursive', 'true');
CALL C#DI.READ_DEPLOYED(_SYS_DI.T_NO_FILESFOLDERS, #PARAMETERS, ?, ?, ?, ?);
DROP TABLE #PARAMETERS;
```

Read recursively all folders (but no files) in the deployed file system of container C:

Sample Code

Read Only Files Recursively

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PARAMETERS LIKE _SYS_DI.TT_PARAMETERS;
INSERT INTO #PARAMETERS ( KEY, VALUE ) VALUES ('ignore_files', 'true');
INSERT INTO #PARAMETERS ( KEY, VALUE ) VALUES ('recursive', 'true');
CALL C#DI.READ_DEPLOYED(_SYS_DI.T_NO_FILESFOLDERS, #PARAMETERS, ?, ?, ?, ?);
DROP TABLE #PARAMETERS;
```

Read specific files or folders in the deployed file system of container C:

Sample Code

Read Specified Files and Folders

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PATHS LIKE _SYS_DI.TT_FILESFOLDERS;
INSERT INTO #PATHS (PATH) VALUES ('src1/');
INSERT INTO #PATHS (PATH) VALUES ('src2/p.hdbprocedure');
CREATE LOCAL TEMPORARY COLUMN TABLE #PARAMETERS LIKE _SYS_DI.TT_PARAMETERS;
INSERT INTO #PARAMETERS ( KEY, VALUE ) VALUES ('recursive', 'false');
CALL C#DI.READ_DEPLOYED(#PATHS, #PARAMETERS, ?, ?, ?, ?);
DROP TABLE #PATHS; DROP TABLE #PARAMETERS;
```

Related Information

WRITE [page 142]
MAKE [page 153]
LIST_DEPLOYED [page 166]

5.3.2.10 GET_MAKE_GROUPS

Calculate the groups of objects that could be passed to a make call.

The SAP HDI Container API includes the `GET_MAKE_GROUPS` command, which enables you to calculate groups of objects that do not have dependencies with each other and could be passed to a make call individually.

Signature

Sample Code

```
GET_MAKE_GROUPS
 (
  IN    DEPLOY_PATHS     _SYS_DI.TT_FILESFOLDERS,
  IN    UNDEPLOY_PATHS   _SYS_DI.TT_FILESFOLDERS,
  IN    PATH_PARAMETERS  _SYS_DI.TT_FILESFOLDERS_PARAMETERS,
  IN    PARAMETERS       _SYS_DI.TT_PARAMETERS,
  OUT  RETURN_CODE      INT,
  OUT  REQUEST_ID       BIGINT,
  OUT  MESSAGES         _SYS_DI.TT_MESSAGES
 )
```

Parameters

The following parameters can be used with `IN` and `OUT` parameters in the `GET_MAKE_GROUPS` command:

DEPLOY_PATHS [IN]

Specifies a list of files or folders to be deployed:

_SYS_DI.TT_FILESFOLDERS

Name	Data Type	Description
PATH	NVARCHAR(511)	A single path is either a fully qualified path to a file (for example, `/path/to/a/file.txt`'), or a fully qualified path to a folder (for example, `/path/to/`).

UNDEPLOY_PATHS [IN]

Specifies a list of files or folders to be **un**deployed:

_SYS_DI.TT_FILESFOLDERS

Name	Data Type	Description
PATH	NVARCHAR(511)	A single path is either a fully qualified file path (for example, `/path/to/a/file.txt`'), or a fully qualified folder path (for example, `/path/to/`).

PATH_PARAMETERS [IN]

The following table lists path parameters you can use with the `GET_MAKE_GROUPS` command at the file or folder level:

_SYS_DI.TT_FILESFOLDERS_PARAMETERS

Name	Data Type	Description
PATH	NVARCHAR(511)	A single path is either a fully qualified file path (for example, `/path/to/a/file.txt`'), or a fully qualified folder path (for example, `/path/to/`).
KEY	NVARCHAR(256)	The key name of the parameter
VALUE	NVARCHAR(256)	The value assigned to the parameter

PARAMETERS [IN]

Additional parameters can be used to control various aspects of the procedure execution. If no parameters are needed, the empty predefined parameters table `_SYS_DI.T_NO_PARAMETERS` can be used.

_SYS_DI.TT_PARAMETERS

Name	Data Type	Description
KEY	NVARCHAR(256)	The key name of the parameter
VALUE	NVARCHAR(256)	The value assigned to the parameter

The following additional parameters are available:

- `container_lock_wait_timeout`
- `max_parallel_jobs`
- `optimized_redeploy`
- `simulate_make`
- `skip_unchanged_expansions`
- `trace_context`
- `trace_level.<trace topic>`
- `treat_warnings_as_errors`
- `undeploy_dependent_recursively`
- `use_redeploy_for_unchanged`
- `message_severity`

RETURN_CODE [OUT]

The return code indicates if the procedure executed successfully. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

REQUEST_ID [OUT]

A unique ID is generated for each HDI container API call. For more details about which IDs are generated for API calls, see *The SQL API for SAP HDI* in *Related Information*.

MESSAGES [OUT]

A table is used to display messages that contain information logged during (and about) the execution of the procedure. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

RESULT [OUT]

Returns a table listing the independent MAKE groups, as follows:

_SYS_DI.TT_MAKE_GROUPS

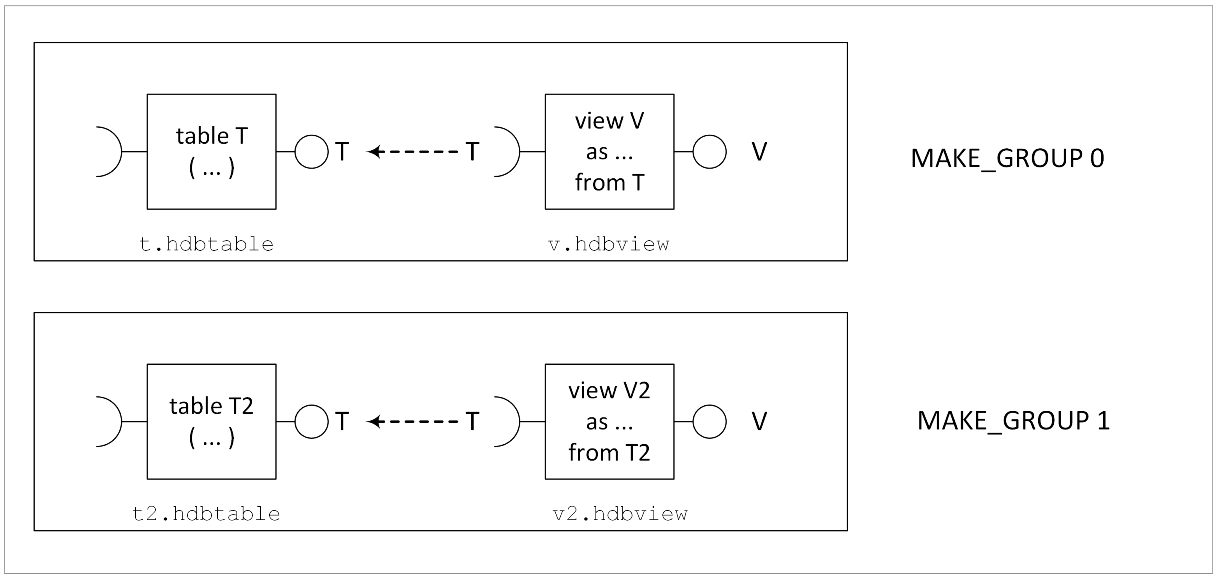
Name	Data Type	Description
MAKE_GROUP	BIGINT	An identifier for a make group which is assigned to every path

Name	Data Type	Description
PATH	NVARCHAR(511)	A single path is either a fully qualified file path (for example, `'a/path/to/a/file.txt`), or a fully qualified folder path (for example, `a/path/to/a/folder`).

Examples

In this example, the following files should be deployed to container C's deployed file system:

- `t.hdbtable`
  Defines a table T
- `v.hdbview`
  Defines a view V that selects from table T, which means that `v.hdbview` depends on `t.hdbtable`
- `t2.hdbtable`
  Defines a table T2
- `v2.hdbview`
  Defines a view V2 that selects from table T2, which means `v2.hdbview` depends on `t2.hdbtable`.



In this example, there are two sets of files (make groups) which could be deployed independently of each other, using the `MAKE` API. If necessary, the files first have to be written to the container's work file system using the `WRITE` API. For the sake of completeness, the HDI configuration files `.hdiconfig` and `.hdinamespace` are also included.

Sample Code

Write Folders and Files to an HDI Container's Work File System

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PATHS LIKE
_SYS_DI.TT_FILESFOLDERS_CONTENT;
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('.hdiconfig',
'{ "plugin_version" : "2.0.0.0", "file_suffixes" : { "hdbtable" :
```

```
{ "plugin_name" : "com.sap.hana.di.table" }, "hdbview" : { "plugin_name" :
"com.sap.hana.di.view" }, "hdbprocedure" : { "plugin_name" :
"com.sap.hana.di.procedure" } } }');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('.hdinamespace', '{ "name": "",
"subfolder": "ignore" }');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/', NULL);
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/t.hdbtable', 'COLUMN TABLE T
( A INTEGER )');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src1/v.hdbview', 'VIEW V AS
SELECT A FROM T');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src2/', '');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('src2/p.hdbprocedure', 'PROCEDURE
P (OUT RESULT INT) LANGUAGE SQLSCRIPT AS BEGIN SELECT COUNT(*) INTO RESULT
FROM V; end');
CALL C#DI.WRITE(#PATHS, _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PATHS;
```

To determine the independent make groups, you can call `GET_MAKE_GROUPS`, which takes the same input parameters as a `MAKE` call. `GET_MAKE_GROUPS` considers `DEPLOY_PATHS` and `UNDEPLOY_PATHS`.

Sample Code

Display the MAKE Groups (`DI#.GET_MAKE_GROUPS`)

```
CREATE LOCAL TEMPORARY COLUMN TABLE #DEPLOY_PATHS LIKE
_SYS_DI.TT_FILESFOLDERS;
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('.hdiconfig');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('.hdinamespace');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('t.hdbtable');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('v.hdbview');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('t2.hdbtable');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('v2.hdbview');
CREATE LOCAL TEMPORARY COLUMN TABLE #UNDEPLOY_PATHS LIKE
_SYS_DI.TT_FILESFOLDERS;
CREATE LOCAL TEMPORARY COLUMN TABLE #PATH_PARAMETERS LIKE
_SYS_DI.TT_FILESFOLDERS_PARAMETERS;
CALL C#DI.GET_MAKE_GROUPS(#DEPLOY_PATHS, #UNDEPLOY_PATHS, #PATH_PARAMETERS,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?, ?);
DROP TABLE #DEPLOY_PATHS;
DROP TABLE #UNDEPLOY_PATHS;
DROP TABLE #PATH_PARAMETERS;
```

The following output example shows what the returned results looks like:

Output Code

`C#DI.GET_MAKE_GROUPS`

```
MAKE_GROUP    PATH
--------------------------
0             t.hdbtable
0             v.hdbview
1             t2.hdbtable
1             v2.hdbview
```

The resulting file paths for each `MAKE` group can be used for separate calls to the `MAKE` API.

→ Remember

Since the `.hdiconfig` and `.hdinamespace` files do not define a database artifact, neither is included in the results returned by the `GET_MAKE_BROUPS` call. For this reason, the `.hdiconfig` and `.hdinamespace` files should be included in the first MAKE call, **before** deploying any database artifacts, as illustrated in the following examples:

Sample Code

MAKE the .hdiconfig and .hdinamespace files, if required

```
CREATE LOCAL TEMPORARY COLUMN TABLE #DEPLOY_PATHS LIKE
_SYS_DI.TT_FILESFOLDERS;
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('.hdiconfig');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('.hdinamespace');
CREATE LOCAL TEMPORARY COLUMN TABLE #UNDEPLOY_PATHS LIKE
_SYS_DI.TT_FILESFOLDERS;
CREATE LOCAL TEMPORARY COLUMN TABLE #PATH_PARAMETERS LIKE
_SYS_DI.TT_FILESFOLDERS_PARAMETERS;
CALL C#DI.MAKE(#DEPLOY_PATHS, #UNDEPLOY_PATHS, #PATH_PARAMETERS,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #DEPLOY_PATHS;
DROP TABLE #UNDEPLOY_PATHS;
DROP TABLE #PATH_PARAMETERS;
```

Sample Code

MAKE Group 0

```
CREATE LOCAL TEMPORARY COLUMN TABLE #DEPLOY_PATHS LIKE
_SYS_DI.TT_FILESFOLDERS;
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('t.hdbtable');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('v.hdbview');
CREATE LOCAL TEMPORARY COLUMN TABLE #UNDEPLOY_PATHS LIKE
_SYS_DI.TT_FILESFOLDERS;
CREATE LOCAL TEMPORARY COLUMN TABLE #PATH_PARAMETERS LIKE
_SYS_DI.TT_FILESFOLDERS_PARAMETERS;
CALL C#DI.MAKE(#DEPLOY_PATHS, #UNDEPLOY_PATHS, #PATH_PARAMETERS,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #DEPLOY_PATHS;
DROP TABLE #UNDEPLOY_PATHS;
DROP TABLE #PATH_PARAMETERS;
```

Sample Code

MAKE Group 1

```
CREATE LOCAL TEMPORARY COLUMN TABLE #DEPLOY_PATHS LIKE
_SYS_DI.TT_FILESFOLDERS;
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('t2.hdbtable');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('v2.hdbview');
CREATE LOCAL TEMPORARY COLUMN TABLE #UNDEPLOY_PATHS LIKE
_SYS_DI.TT_FILESFOLDERS;
CREATE LOCAL TEMPORARY COLUMN TABLE #PATH_PARAMETERS LIKE
_SYS_DI.TT_FILESFOLDERS_PARAMETERS;
CALL C#DI.MAKE(#DEPLOY_PATHS, #UNDEPLOY_PATHS, #PATH_PARAMETERS,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #DEPLOY_PATHS;
DROP TABLE #UNDEPLOY_PATHS;
DROP TABLE #PATH_PARAMETERS;
```

Related Information

WRITE [page 142]
MAKE [page 153]

5.3.2.11 GET_DEPENDENCIES

Retrieve dependency data for objects and groups.

The SAP HDI Container API includes the `GET_MAKE_GROUPS` command, which enables you to Retrieve dependency data; the kind of data that is retrieved depends on the optional "variant" parameter passed in `PARAMETERS`.

Signature

Sample Code

```
GET_DEPENDENCIES
 (
  IN    OBJECTS       _SYS_DI.TT_OBJECTS,
  IN    PARAMETERS    _SYS_DI.TT_PARAMETERS,
  OUT   RETURN_CODE   INT,
  OUT   REQUEST_ID    BIGINT,
  OUT   MESSAGES      _SYS_DI.TT_MESSAGES,
  OUT   RESULT        _SYS_DI.TT_DEPENDENCIES
 )
```

Parameters

The following parameters can be used with `IN` and `OUT` parameters in the `GET_DEPENDENCIES` command:

OBJECTS [IN]

The Objects for which dependency data should be retrieved:

_SYS_DI.TT_OBJECTS

Name	Data Type	Description
DOMAIN	NVARCHAR(256)	Domain of the object to be queried. • ' ': An empty string indicates that the object is a file in the HDI container's deployed file system • 'db': Indicates that the object is a database object in the HDI container
URI	NVARCHAR(510)	Unique resource identifier of the object. • A file path, if the object is the deployed file system in the container • An object name, if the object is a database object in the container

PARAMETERS [IN]

Additional parameters can be used to control various aspects of the procedure execution. If no parameters are needed, the empty predefined parameters table `_SYS_DI.T_NO_PARAMETERS` can be used.

_SYS_DI.TT_PARAMETERS

Name	Data Type	Description
KEY	NVARCHAR(256)	The key name of the parameter
VALUE	NVARCHAR(256)	The value assigned to the parameter

The following additional parameters are available:

- `container_lock_wait_timeout`
- `trace_context`
- `trace_level.<trace topic>`
- `variant`:
  - `providers` (default)
    Returns for the given objects the files providing them
  - `provides_and_requires`
    Returns for the given files the provided and required objects
  - `depends`
    Returns for the given files the files they directly depend on
  - `impacted`
    Returns for the given files the files that directly depend on them

- `message_severity`

RETURN_CODE [OUT]

The return code indicates if the procedure executed successfully. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

REQUEST_ID [OUT]

A unique ID is generated for each HDI container API call. For more details about which IDs are generated for API calls, see *The SQL API for SAP HDI* in *Related Information*.

MESSAGES [OUT]

A table is used to display messages that contain information logged during (and about) the execution of the procedure. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

RESULT [OUT]

Returns the dependency information for the queried objects. Depending on the used variant, some of the fields may not contain data:

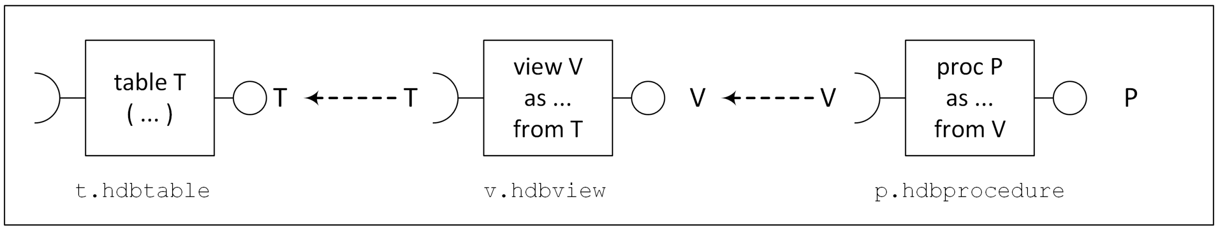
_SYS_DI.TT_DEPENDENCIES

Name	Data Type	Description
OBJECT_DOMAIN	NVARCHAR(256)	The domain where the object is located
OBJECT_NAME	NVARCHAR(510)	Unique resource identifier of the object
DEPENDENCY_DOMAIN	NVARCHAR(256)	The domain where the dependency is located
DEPENDENCY_NAME	NVARCHAR(510)	Unique resource identifier of the dependency
DEPENDENCY_TYPE	NVARCHAR(20)	For the "`providers`" variant, indicates the type of the dependency: • PROVIDE_RT The object provides an object with the name `DEPENDENCY_NAME` • REQUIRE_DEPLOY The object has a dependency on an object with the name `DEPENDENCY_NAME`
DEPENDENCY_LOCATION	NVARCHAR(100)	For the "`providers`" variant, returns the line and column location of the dependency in the object. The type of location returned depends on the object type
DEPENDENCY_LOCATION_PATH	NVARCHAR(5000)	For the "`providers`" variant, returns a path to the location of the dependency in the object. The type of location path returned depends on the object type.

Examples

In this example, the following files should be deployed to container C's deployed file system:

- `t.hdbtable`
  Defines a table T.
- `v.hdbview`
  Defines a view V that selects from table T, which means that `v.hdbview` depends on `t.hdbtable`.
- `p.hdbprocedure`
  Defines a procedure P that depends on `v.hdbview` (which depends on `t.hdbtable`).



If not already present, the dependent files listed above first must be written to the container's work file system using the `WRITE` API. For the sake of completeness, the HDI configuration files `.hdiconfig` and `.hdinamespace` are also included.

Sample Code

Write Folders and Files to an HDI Container's Work File System

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PATHS LIKE
_SYS_DI.TT_FILESFOLDERS_CONTENT;
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('.hdiconfig',
'{ "plugin_version" : "2.0.0.0", "file_suffixes" : { "hdbtable" :
{ "plugin_name" : "com.sap.hana.di.table" }, "hdbview" : { "plugin_name" :
"com.sap.hana.di.view" }, "hdbprocedure" : { "plugin_name" :
"com.sap.hana.di.procedure" } } }');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('.hdinamespace', '{ "name": "",
"subfolder": "ignore" }');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('t.hdbtable', 'COLUMN TABLE T ( A
INTEGER )');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('v.hdbview', 'VIEW V AS SELECT A
FROM T');
INSERT INTO #PATHS (PATH, CONTENT) VALUES ('p.hdbprocedure', 'PROCEDURE P
(OUT RESULT INT) LANGUAGE SQLSCRIPT AS BEGIN SELECT COUNT(*) INTO RESULT FROM
V; end');
CALL C#DI.WRITE(#PATHS, _SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PATHS;
```

The following example shows how to deploy the files to the database:

Sample Code

Deploy Files

```
CREATE LOCAL TEMPORARY COLUMN TABLE #DEPLOY_PATHS LIKE
_SYS_DI.TT_FILESFOLDERS;
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('.hdiconfig');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('.hdinamespace');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('t.hdbtable');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('v.hdbview');
INSERT INTO #DEPLOY_PATHS (PATH) VALUES ('p.hdbprocedure');
```

```
CREATE LOCAL TEMPORARY COLUMN TABLE #UNDEPLOY_PATHS LIKE
_SYS_DI.TT_FILESFOLDERS;
CREATE LOCAL TEMPORARY COLUMN TABLE #PATH_PARAMETERS LIKE
_SYS_DI.TT_FILESFOLDERS_PARAMETERS;
CALL C#DI.MAKE(#DEPLOY_PATHS, #UNDEPLOY_PATHS, #PATH_PARAMETERS,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #DEPLOY_PATHS;
DROP TABLE #UNDEPLOY_PATHS;
DROP TABLE #PATH_PARAMETERS;
```

The dependency relationship at the file (and object) level can be extracted using the `GET_DEPENDENCIES` procedure. The following examples shows what the returned results looks like:

→ Tip

The `variant` parameter determines the type of dependency information being queried, for example, “providers” (default) or “provides and requires”. For more information about the variant parameter, see `_SYS_DI.TT_PARAMETERS` in Parameters [IN] [page 186].

Variant Parameter: Providers

In this variant, the command returns, for the given objects, the files providing them. In this example, the names of the database objects T, V, P (domain is `db`) are passed into the procedure, which then returns the paths of the design-time files that contain the definitions from which the specified database objects were created.

Sample Code

Call `#DI.GET_DEPENDENCIES` (`providers`)

```
CREATE LOCAL TEMPORARY TABLE #OBJECTS LIKE _SYS_DI.TT_OBJECTS;
CREATE LOCAL TEMPORARY TABLE #PARAMETERS LIKE _SYS_DI.TT_PARAMETERS;
INSERT INTO #OBJECTS (DOMAIN, URI) VALUES ('db', 'T');
INSERT INTO #OBJECTS (DOMAIN, URI) VALUES ('db', 'V');
INSERT INTO #OBJECTS (DOMAIN, URI) VALUES ('db', 'P');
INSERT INTO #PARAMETERS (KEY, VALUE) VALUES ('variant', 'providers');
CALL C#DI.GET_DEPENDENCIES(#OBJECTS, #PARAMETERS, ?, ?, ?, ?);
DROP TABLE #OBJECTS;
DROP TABLE #PARAMETERS;
```

Output Code

Call `#DI.GET_DEPENDENCIES` Results (`providers`)

```
OBJECT_  OBJECT_  DEPENDENCY_  DEPENDENCY_     DEPENDENCY_  DEPENDENCY_  [...]
DOMAIN   NAME     DOMAIN       NAME            TYPE         LOCATION
------------------------------------------------------------------------------
db       P                     p.hdbprocedure               1:11-12
db       T                     t.hdbtable                   1:14-15
db       V                     v.hdbview                    1:6-7
```

Variant Parameter: Provides and Requires

In this variant, the command returns, for the given files, the provided and required objects. In this example, the paths of the design-time files are passed into the procedure, which then returns information about the database objects that each of the design-time files creates (provides) or needs (requires) at the time of their deployment. Here, `p.hdbprocedure` provides an object “P” and requires an object “V”; `v.hdbview` provides an object “V” and requires an object “T”; `t.hdbtable` provides an object “T”.

Sample Code

Call `#DI.GET_DEPENDENCIES` (`provides_and_requires`)

```
CREATE LOCAL TEMPORARY TABLE #OBJECTS LIKE _SYS_DI.TT_OBJECTS;
CREATE LOCAL TEMPORARY TABLE #PARAMETERS LIKE _SYS_DI.TT_PARAMETERS;
INSERT INTO #OBJECTS (DOMAIN, URI) VALUES ('', 't.hdbtable');
INSERT INTO #OBJECTS (DOMAIN, URI) VALUES ('', 'v.hdbview');
INSERT INTO #OBJECTS (DOMAIN, URI) VALUES ('', 'p.hdbprocedure');
INSERT INTO #PARAMETERS (KEY, VALUE) VALUES ('variant',
'provides_and_requires');
CALL C#DI.GET_DEPENDENCIES(#OBJECTS, #PARAMETERS, ?, ?, ?, ?);
DROP TABLE #OBJECTS;
DROP TABLE #PARAMETERS;
```

Output Code

`#DI.GET_DEPENDENCIES` Results (`provides_and_requires`)

```
OBJECT_  OBJECT_          DEPENDENCY_  DEPENDENCY_  DEPENDENCY_     DEPENDENCY_
DOMAIN   NAME             DOMAIN       NAME         TYPE            LOCATION
-------------------------------------------------------------------------------
         p.hdbprocedure   db           P            PROVIDE_RT      1:11-12
         p.hdbprocedure   db           V            REQUIRE_DEPLOY  4:17-18
         t.hdbtable       db           T            PROVIDE_RT      1:14-15
         v.hdbview        db           T            REQUIRE_DEPLOY  2:15-16
         v.hdbview        db           V            PROVIDE_RT      1:6-7
```

Variant Parameter: Depends

In this variant, the command returns, for the given objects, the files they directly depend on. The paths of the design-time files are passed into the procedure, which then returns the paths of the design-time files that they depend on. In this example, `p.hdbprocedure` depends on `v.hdbview`, and `v.hdbview` depends on `t.hdbtable`.

Sample Code

Call `#DI.GET_DEPENDENCIES` (`depends`)

```
CREATE LOCAL TEMPORARY TABLE #OBJECTS LIKE _SYS_DI.TT_OBJECTS;
CREATE LOCAL TEMPORARY TABLE #PARAMETERS LIKE _SYS_DI.TT_PARAMETERS;
INSERT INTO #OBJECTS (DOMAIN, URI) VALUES ('', 't.hdbtable');
INSERT INTO #OBJECTS (DOMAIN, URI) VALUES ('', 'v.hdbview');
INSERT INTO #OBJECTS (DOMAIN, URI) VALUES ('', 'p.hdbprocedure');
INSERT INTO #PARAMETERS (KEY, VALUE) VALUES ('variant', 'depends');
CALL C#DI.GET_DEPENDENCIES(#OBJECTS, #PARAMETERS, ?, ?, ?, ?);
DROP TABLE #OBJECTS;
DROP TABLE #PARAMETERS;
```

Output Code

`#DI.GET_DEPENDENCIES` Results (`depends`)

```
OBJECT_  OBJECT_          DEPENDENCY_  DEPENDENCY_  DEPENDENCY_ ...  DEPENDENCY
DOMAIN   NAME             DOMAIN       NAME         TYPE             _LOCATION
                                                                     _PATH
-------------------------------------------------------------------------------
         p.hdbprocedure                v.hdbview               [...]  ?
```

```
          v.hdbview                     t.hdbtable                [...]   ?
```

Variant Parameter: Impacted

In this variant, the command returns, for the given objects, the files they directly depend on. The paths of the design-time files are passed into the procedure, which then returns the paths of the design-time files that have a dependency to them. Here, `t.hdbtable` has a dependent (impacted) object in `v.hdbview`, and `v.hdbview` has a dependent object in `p.hdbprocedure`.

Sample Code

Call `#DI.GET_DEPENDENCIES` (`impacted`)

```
CREATE LOCAL TEMPORARY TABLE #OBJECTS LIKE _SYS_DI.TT_OBJECTS;
CREATE LOCAL TEMPORARY TABLE #PARAMETERS LIKE _SYS_DI.TT_PARAMETERS;
INSERT INTO #OBJECTS (DOMAIN, URI) VALUES ('', 't.hdbtable');
INSERT INTO #OBJECTS (DOMAIN, URI) VALUES ('', 'v.hdbview');
INSERT INTO #OBJECTS (DOMAIN, URI) VALUES ('', 'p.hdbprocedure');
INSERT INTO #PARAMETERS (KEY, VALUE) VALUES ('variant', 'impacted');
CALL C#DI.GET_DEPENDENCIES(#OBJECTS, #PARAMETERS, ?, ?, ?, ?);
DROP TABLE #OBJECTS;
DROP TABLE #PARAMETERS;
```

Output Code

`#DI.GET_DEPENDENCIES` Results (`impacted`)

```
OBJECT_  OBJECT_         DEPENDENCY_  DEPENDENCY_  DEPENDENCY_  [...]  DEPENDENCY
DOMAIN   NAME            DOMAIN       NAME         TYPE                _LOCATION
                                                                       _PATH
----------------------------------------------------------------------------------
         t.hdbtable                   v.hdbview                 [...]  ?
         v.hdbview                    p.hdbprocedure            [...]  ?
```

Related Information

WRITE [page 142]

MAKE [page 153]

5.3.2.12 LIST_CONFIGURED_LIBRARIES

Display which build plug-in libraries are available in an HDI container.

The SAP HDI Container API includes the `LIST_CONFIGURED_LIBRARIES` command, which enables you to display a list the build plug-ins that are available for use in a specified HDI container.

Signature

Sample Code

```
LIST_CONFIGURED_LIBRARIES
 (
  IN  PARAMETERS   _SYS_DI.TT_PARAMETERS,
  OUT RETURN_CODE INT,
  OUT REQUEST_ID  BIGINT,
  OUT MESSAGES    _SYS_DI.TT_MESSAGES,
  OUT RESULT      _SYS_DI.TT_LIBRARY_INFORMATION
 )
```

Parameters

The following parameters can be used with `IN` and `OUT` parameters in the `LIST_CONFIGURED_LIBRARIES` command:

PARAMETERS [IN]

Additional parameters can be used to control various aspects of the procedure execution. If no parameters are needed, the empty predefined parameters table `_SYS_DI.T_NO_PARAMETERS` can be used.

_SYS_DI.TT_PARAMETERS

Name	Data Type	Description
KEY	NVARCHAR(256)	The key name of the parameter
VALUE	NVARCHAR(256)	The value assigned to the parameter

The following additional parameters are available:

- `container_lock_wait_timeout`
- `trace_context`
- `trace_level.<trace topic>`
- `message_severity`

RETURN_CODE [OUT]

The return code indicates if the procedure executed successfully. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

REQUEST_ID [OUT]

A unique ID is generated for each HDI container API call. For more details about which IDs are generated for API calls, see *The SQL API for SAP HDI* in *Related Information*.

MESSAGES [OUT]

A table is used to display messages that contain information logged during (and about) the execution of the procedure. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

RESULT [OUT]

Returns a list of the build plug-in libraries that have been configured for the container by the container's administrator.

_SYS_DI.c

Name	Data Type	Description
LIBRARY_NAME	NVARCHAR(256)	The name of the build plug-in library domain where the build plug-in is located
LIBRARY_VERSION	NVARCHAR(16)	The version number of the build plug-in library
PLUGIN_ID	NVARCHAR(256)	The identifying name of the build plug-in
PLUGIN_VERSION	NVARCHAR(16)	The version number of the build plug-in

Examples

The following example shows an excerpt from the response to the call `LIST_CONFIGURED_LIBRARIES` for the container "C" including information about the configured libraries:

Sample Code

Call `#DI.LIST_CONFIGURED_LIBRARIES`

```
CALL C#DI.LIST_CONFIGURED_LIBRARIES(_SYS_DI.T_NO_PARAMETERS, ?, ?, ?, ?);
```

Note

For reasons of space, the example command output displayed below is occasionally incomplete (`[...]`); the example output is provided here for illustration purposes only.

Output Code

Call `#DI.LIST_CONFIGURED_LIBRARIES`

```
LIBRARY_                        LIBRARY_  PLUGIN_                         PLUGIN_
NAME                            VERSION   ID                              VERSION
-------------------------------------------------------------------------------
com.sap.hana.di.afllangprocedure  0.0     com.[...].afllangprocedure  2.0.30.0
com.sap.hana.di.analyticprivilege 0.0     com.[...].analyticprivilege 2.0.30.0
com.sap.hana.di.calculationview   0.0     com.[...].calculationview   2.0.30.0
com.sap.hana.di.cds               0.0     com.sap.hana.di.cds         2.0.30.0
com.sap.hana.di.collection        0.0     com.sap.hana.di.collection  2.0.30.0
...
```

Related Information

5.3.2.13 GRANT_CONTAINER_SCHEMA_PRIVILEGES

Grant access privileges to a database object consumer for the entire container schema where the database objects are located.

Users that would like to consume objects deployed to an HDI container need to be granted the appropriate privileges. The privileges can be granted to specific objects in the schema (for example, C) by use of a role that has been deployed to the target container (see *GRANT_CONTAINER_SCHEMA_ROLES*), or by granting privileges for the entire schema (*GRANT_CONTAINER_SCHEMA_PRIVILEGES*). Privileges granted by the procedure `GRANT_CONTAINER_SCHEMA_PRIVILEGES` can be revoked with the procedure `REVOKE_CONTAINER_SCHEMA_PRIVILEGES`.

Signature

Sample Code

```
GRANT_CONTAINER_SCHEMA_PRIVILEGES
 (
  IN  PRIVILEGES   _SYS_DI.TT_SCHEMA_PRIVILEGES,
  IN  PARAMETERS   _SYS_DI.TT_PARAMETERS,
  OUT RETURN_CODE INT,
  OUT REQUEST_ID  BIGINT,
  OUT MESSAGES    _SYS_DI.TT_MESSAGES
 )
```

Parameters

The following parameters can be used with `IN` and `OUT` parameters in the `GRANT_CONTAINER_SCHEMA_PRIVILEGES` command:

PRIVILEGES [IN]

The list of privileges for the container schema which you want to grant to a principal.

_SYS_DI.TT_SCHEMA_PRIVILEGES

Name	Data Type	Description
PRIVILEGE_NAME	NVARCHAR(256)	Name of the schema privilege to be granted. The following list shows the permitted privileges for HDI container schemas: • CREATE ANY • CREATE TEMPORARY TABLE • EXECUTE • SELECT CDS METADATA • SELECT METADATA • SELECT • INSERT • UPDATE • DELETE • DEBUG • UNMASKED If set to "`true`" (default), the configuration parameter "`enable_container_schema_privileges_index_trigger_references`" in `diserver.ini`, in the section "`api`" allows the following to be added to the list of valid privileges: • TRIGGER • INDEX • REFERENCES **i Note** The recommended setting is "`false`".
PRINCI-PAL_SCHEMA_NAME	NVARCHAR(256)	The schema name of the principal. This is needed, for example, if the principal is a schema-local role.
PRINCIPAL_NAME	NVARCHAR(256)	The name of the principal

PARAMETERS [IN]

Additional parameters can be used to control various aspects of the procedure execution. If no parameters are needed, the empty predefined parameters table `_SYS_DI.T_NO_PARAMETERS` can be used.

_SYS_DI.TT_PARAMETERS

Name	Data Type	Description
KEY	NVARCHAR(256)	The key name of the parameter
VALUE	NVARCHAR(256)	The value assigned to the parameter

The following additional parameters are available:

- `container_lock_wait_timeout`

- `trace_context`
- `trace_level.<trace topic>`
- `message_severity`

RETURN_CODE [OUT]

The return code indicates if the procedure executed successfully. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

REQUEST_ID [OUT]

A unique ID is generated for each HDI container API call. For more details about which IDs are generated for API calls, see *The SQL API for SAP HDI* in *Related Information*.

MESSAGES [OUT]

A table is used to display messages that contain information logged during (and about) the execution of the procedure. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

Examples

To grant access privileges for the container schema ("C" in this example) to another user, for example, `NEW_CONTAINER_CONSUMER`, use a command like the one shown in the following example:

Sample Code

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PRIVILEGES LIKE
_SYS_DI.TT_SCHEMA_PRIVILEGES;
INSERT INTO #PRIVILEGES(PRIVILEGE_NAME, PRINCIPAL_SCHEMA_NAME,
PRINCIPAL_NAME) VALUES ('SELECT', '', 'NEW_CONTAINER_CONSUMER');
CALL C#DI.GRANT_CONTAINER_SCHEMA_PRIVILEGES(#PRIVILEGES,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PRIVILEGES;
```

To grant privileges on the container schema "C" to a role (for example, `ROLE_IN_CONTAINER_D`) in another container or schema ("D" in this example), use a command like the one shown in the following example:

Sample Code

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PRIVILEGES LIKE
_SYS_DI.TT_SCHEMA_PRIVILEGES;
INSERT INTO #PRIVILEGES(PRIVILEGE_NAME, PRINCIPAL_SCHEMA_NAME,
PRINCIPAL_NAME) VALUES ('INSERT', 'D', 'ROLE_IN_CONTAINER_D');
CALL C#DI.GRANT_CONTAINER_SCHEMA_PRIVILEGES(#PRIVILEGES,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PRIVILEGES;
```

Related Information

REVOKE_CONTAINER_SCHEMA_PRIVILEGES [page 197]
GRANT_CONTAINER_SCHEMA_ROLES [page 200]

5.3.2.14 REVOKE_CONTAINER_SCHEMA_PRIVILEGES

Revoke access privileges from a database object consumer for the entire container schema where the database objects are located.

Users that would like to consume objects deployed to an HDI container need to be granted the appropriate privileges for the consumed objects. The privileges can be granted to specific objects in the schema (for example, C) by use of a role that has been deployed to the target container (see `GRANT_CONTAINER_SCHEMA_ROLES`), or by granting privileges for the entire schema (`GRANT_CONTAINER_SCHEMA_PRIVILEGES`). Privileges granted by the procedure `GRANT_CONTAINER_SCHEMA_PRIVILEGES` can be **revoked** with the procedure `REVOKE_CONTAINER_SCHEMA_PRIVILEGES`.

Signature

Sample Code

```
REVOKE_CONTAINER_SCHEMA_PRIVILEGES
 (
  IN   PRIVILEGES   _SYS_DI.TT_SCHEMA_PRIVILEGES,
  IN   PARAMETERS   _SYS_DI.TT_PARAMETERS,
  OUT  RETURN_CODE  INT,
  OUT  REQUEST_ID   BIGINT,
  OUT  MESSAGES     _SYS_DI.TT_MESSAGES
 )
```

Parameters

The following parameters can be used with `IN` and `OUT` parameters in the `REVOKE_CONTAINER_SCHEMA_PRIVILEGES` command:

PRIVILEGES [IN]

The list of privileges for the container schema which you want to revoke from a principal.

_SYS_DI.TT_SCHEMA_PRIVILEGES

Name	Data Type	Description
PRIVILEGE_NAME	NVARCHAR(256)	Name of the schema privilege to be revoked. The following list shows the permitted privileges for HDI container schemas: • CREATE ANY • CREATE TEMPORARY TABLE • EXECUTE • SELECT CDS METADATA • SELECT METADATA • SELECT • INSERT • UPDATE • DELETE • DEBUG • UNMASKED If set to `"true"` (default), the configuration parameter "`enable_container_schema_privileges_index_trigger_references`" in `diserver.ini`, in the section "`api`" allows the following to be added to the list of valid privileges: • TRIGGER • INDEX • REFERENCES i Note The recommended setting is `"false"`.
PRINCI-PAL_SCHEMA_NAME	NVARCHAR(256)	The schema name of the principal. This is needed, for example, if the principal is a schema-local role.
PRINCIPAL_NAME	NVARCHAR(256)	The name of the principal

PARAMETERS [IN]

Additional parameters can be used to control various aspects of the procedure execution. If no parameters are needed, the empty predefined parameters table `_SYS_DI.T_NO_PARAMETERS` can be used.

_SYS_DI.TT_PARAMETERS

Name	Data Type	Description
KEY	NVARCHAR(256)	The key name of the parameter
VALUE	NVARCHAR(256)	The value assigned to the parameter

The following additional parameters are available:

- `container_lock_wait_timeout`

- `trace_context`
- `trace_level.<trace topic>`
- `message_severity`

RETURN_CODE [OUT]

The return code indicates if the procedure executed successfully. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

REQUEST_ID [OUT]

A unique ID is generated for each HDI container API call. For more details about which IDs are generated for API calls, see *The SQL API for SAP HDI* in *Related Information*.

MESSAGES [OUT]

A table is used to display messages that contain information logged during (and about) the execution of the procedure. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

Examples

To grant (and then revoke) access privileges for the container schema ("C" in this example) to (and from) another user, for example, `NEW_CONTAINER_CONSUMER`, use a command like the ones shown in the following examples:

Sample Code

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PRIVILEGES LIKE
_SYS_DI.TT_SCHEMA_PRIVILEGES;
INSERT INTO #PRIVILEGES(PRIVILEGE_NAME, PRINCIPAL_SCHEMA_NAME,
PRINCIPAL_NAME) VALUES ('SELECT', '', 'NEW_CONTAINER_CONSUMER');
CALL C#DI.GRANT_CONTAINER_SCHEMA_PRIVILEGES(#PRIVILEGES,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PRIVILEGES;
```

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PRIVILEGES LIKE
_SYS_DI.TT_SCHEMA_PRIVILEGES;
INSERT INTO #PRIVILEGES(PRIVILEGE_NAME, PRINCIPAL_SCHEMA_NAME,
PRINCIPAL_NAME) VALUES ('SELECT', '', 'NEW_CONTAINER_CONSUMER');
CALL C#DI.REVOKE_CONTAINER_SCHEMA_PRIVILEGES(#PRIVILEGES,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PRIVILEGES;
```

To grant (and then revoke) privileges on the container schema "C" to (and from) a role (for example, `ROLE_IN_CONTAINER_D`) in another container or schema ("D" in this example), use a command like the ones shown in the following examples:

Sample Code

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PRIVILEGES LIKE
_SYS_DI.TT_SCHEMA_PRIVILEGES;
INSERT INTO #PRIVILEGES(PRIVILEGE_NAME, PRINCIPAL_SCHEMA_NAME,
PRINCIPAL_NAME) VALUES ('INSERT', 'D', 'ROLE_IN_CONTAINER_D');
```

```
CALL C#DI.GRANT_CONTAINER_SCHEMA_PRIVILEGES(#PRIVILEGES,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PRIVILEGES;
```

```
CREATE LOCAL TEMPORARY COLUMN TABLE #PRIVILEGES LIKE
_SYS_DI.TT_SCHEMA_PRIVILEGES;
INSERT INTO #PRIVILEGES(PRIVILEGE_NAME, PRINCIPAL_SCHEMA_NAME,
PRINCIPAL_NAME) VALUES ('INSERT', 'D', 'ROLE_IN_CONTAINER_D');
CALL C#DI.REVOKE_CONTAINER_SCHEMA_PRIVILEGES(#PRIVILEGES,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #PRIVILEGES;
```

Related Information

GRANT_CONTAINER_SCHEMA_PRIVILEGES [page 194]
REVOKE_CONTAINER_SCHEMA_ROLES [page 202]

5.3.2.15 GRANT_CONTAINER_SCHEMA_ROLES

Enable access to objects in a container schema by means of a role already deployed to the container schema.

Users who would like to consume objects deployed to a container need to be granted the appropriate privileges for the consumed objects. The access privileges can be granted to specific objects in the schema (for example, `C#DI`) by using a role that has been deployed to the container (`GRANT_CONTAINER_SCHEMA_ROLES`), or by granting privileges for the entire schema, for example, with the procedure `GRANT_CONTAINER_SCHEMA_PRIVILEGES`.

Signature

Sample Code

```
GRANT_CONTAINER_SCHEMA_ROLES
 (
  IN   PRIVILEGES   _SYS_DI.TT_SCHEMA_ROLES,
  IN   PARAMETERS   _SYS_DI.TT_PARAMETERS,
  OUT  RETURN_CODE  INT,
  OUT  REQUEST_ID   BIGINT,
  OUT  MESSAGES     _SYS_DI.TT_MESSAGES
 )
```

Parameters

The following parameters can be used with `IN` and `OUT` parameters in the `GRANT_CONTAINER_SCHEMA_ROLES` command:

PRIVILEGES [IN]

The list of roles from the container schema which you want to grant to a principal.

_SYS_DI.TT_SCHEMA_ROLES

Name	Data Type	Description
ROLE_NAME	NVARCHAR(256)	Name of the role to be granted.
PRINCI-PAL_SCHEMA_NAME	NVARCHAR(256)	The schema name of the principal. This is needed, for example, if the principal is a schema-local role.
PRINCIPAL_NAME	NVARCHAR(256)	The name of the principal

PARAMETERS [IN]

Additional parameters can be used to control various aspects of the procedure execution. If no parameters are needed, the empty predefined parameters table `_SYS_DI.T_NO_PARAMETERS` can be used.

_SYS_DI.TT_PARAMETERS

Name	Data Type	Description
KEY	NVARCHAR(256)	The key name of the parameter
VALUE	NVARCHAR(256)	The value assigned to the parameter

The following additional parameters are available:

- `container_lock_wait_timeout`
- `trace_context`
- `trace_level.<trace topic>`
- `message_severity`

RETURN_CODE [OUT]

The return code indicates if the procedure executed successfully. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

REQUEST_ID [OUT]

A unique ID is generated for each HDI container API call. For more details about which IDs are generated for API calls, see *The SQL API for SAP HDI* in *Related Information*.

MESSAGES [OUT]

A table is used to display messages that contain information logged during (and about) the execution of the procedure. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

Examples

To grant a role from the container schema ("C" in this example) to another user, for example, NEW_CONTAINER_CONSUMER, use a command like the one shown in the following example:

Sample Code

```
CREATE LOCAL TEMPORARY COLUMN TABLE #ROLES LIKE _SYS_DI.TT_SCHEMA_ROLES;
INSERT INTO #ROLES(ROLE_NAME, PRINCIPAL_SCHEMA_NAME, PRINCIPAL_NAME) VALUES
('myrole', '', 'NEW_CONTAINER_CONSUMER');
CALL C#DI.GRANT_CONTAINER_SCHEMA_ROLES(#ROLES,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #ROLES;
```

To grant a role from the container schema "C" to another role (for example, ROLE_IN_CONTAINER_D) in another container or schema ("D" in this example), use a command like the one shown in the following example:

Sample Code

```
CREATE LOCAL TEMPORARY COLUMN TABLE #ROLES LIKE _SYS_DI.TT_SCHEMA_ROLES;
INSERT INTO #ROLES(ROLE_NAME, PRINCIPAL_SCHEMA_NAME, PRINCIPAL_NAME) VALUES
('myrole', 'D', 'ROLE_IN_CONTAINER_D');
CALL C#DI.GRANT_CONTAINER_SCHEMA_ROLES(#ROLES,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #ROLES;
```

Related Information

GRANT_CONTAINER_SCHEMA_PRIVILEGES [page 194]
REVOKE_CONTAINER_SCHEMA_ROLES [page 202]

5.3.2.16 REVOKE_CONTAINER_SCHEMA_ROLES

Disable access to objects in a container schema by means of a role from the container schema.

Users who would like to consume objects deployed to a container need to be granted the appropriate access privileges for the consumed objects. The access privileges granted by the procedure GRANT_CONTAINER_SCHEMA_ROLES can be revoked with the REVOKE_CONTAINER_SCHEMA_ROLES procedure.

Signature

Sample Code

```
REVOKE_CONTAINER_SCHEMA_ROLES
 (
  IN   PRIVILEGES   _SYS_DI.TT_SCHEMA_ROLES,
  IN   PARAMETERS   _SYS_DI.TT_PARAMETERS,
  OUT  RETURN_CODE  INT,
  OUT  REQUEST_ID   BIGINT,
  OUT  MESSAGES     _SYS_DI.TT_MESSAGES
 )
```

Parameters

The following parameters can be used with `IN` and `OUT` parameters in the `REVOKE_CONTAINER_SCHEMA_ROLES` command:

PRIVILEGES [IN]

The list of roles from the container schema which you want to grant to a principal.

_SYS_DI.TT_SCHEMA_ROLES

Name	Data Type	Description
ROLE_NAME	NVARCHAR(256)	Name of the role to be revoked.
PRINCI-PAL_SCHEMA_NAME	NVARCHAR(256)	The schema name of the principal. This is needed, for example, if the principal is a schema-local role.
PRINCIPAL_NAME	NVARCHAR(256)	The name of the principal

PARAMETERS [IN]

Additional parameters can be used to control various aspects of the procedure execution. If no parameters are needed, the empty predefined parameters table `_SYS_DI.T_NO_PARAMETERS` can be used.

_SYS_DI.TT_PARAMETERS

Name	Data Type	Description
KEY	NVARCHAR(256)	The key name of the parameter
VALUE	NVARCHAR(256)	The value assigned to the parameter

The following additional parameters are available:

- `container_lock_wait_timeout`
- `trace_context`
- `trace_level.<trace topic>`

- `message_severity`

RETURN_CODE [OUT]

The return code indicates if the procedure executed successfully. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

REQUEST_ID [OUT]

A unique ID is generated for each HDI container API call. For more details about which IDs are generated for API calls, see *The SQL API for SAP HDI* in *Related Information*.

MESSAGES [OUT]

A table is used to display messages that contain information logged during (and about) the execution of the procedure. For more details about which codes are returned, see *The SQL API for SAP HDI* in *Related Information*.

Examples

To grant a role from the container schema ("C" in this example) to another user, for example, `NEW_CONTAINER_CONSUMER` and then revoke the granted role again, use a command like the ones shown in the following examples:

Sample Code

```
CREATE LOCAL TEMPORARY COLUMN TABLE #ROLES LIKE _SYS_DI.TT_SCHEMA_ROLES;
INSERT INTO #ROLES(ROLE_NAME, PRINCIPAL_SCHEMA_NAME, PRINCIPAL_NAME) VALUES
('myrole', '', 'NEW_CONTAINER_CONSUMER');
CALL C#DI.GRANT_CONTAINER_SCHEMA_ROLES(#ROLES,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #ROLES;
```

```
CREATE LOCAL TEMPORARY COLUMN TABLE #ROLES LIKE _SYS_DI.TT_SCHEMA_ROLES;
INSERT INTO #ROLES(ROLE_NAME, PRINCIPAL_SCHEMA_NAME, PRINCIPAL_NAME) VALUES
('myrole', '', 'NEW_CONTAINER_CONSUMER');
CALL C#DI.REVOKE_CONTAINER_SCHEMA_ROLES(#ROLES,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #ROLES;
```

To grant a role from the container schema "C" to another role (for example, `ROLE_IN_CONTAINER_D`) in another container or schema ("D" in this example) and then revoke the granted role again, use a command like the ones shown in the following examples:

Sample Code

```
CREATE LOCAL TEMPORARY COLUMN TABLE #ROLES LIKE _SYS_DI.TT_SCHEMA_ROLES;
INSERT INTO #ROLES(ROLE_NAME, PRINCIPAL_SCHEMA_NAME, PRINCIPAL_NAME) VALUES
('myrole', 'D', 'ROLE_IN_CONTAINER_D');
CALL C#DI.GRANT_CONTAINER_SCHEMA_ROLES(#ROLES,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #ROLES;
```

```
CREATE LOCAL TEMPORARY COLUMN TABLE #ROLES LIKE _SYS_DI.TT_SCHEMA_ROLES;
```

```
INSERT INTO #ROLES(ROLE_NAME, PRINCIPAL_SCHEMA_NAME, PRINCIPAL_NAME) VALUES
('myrole', 'D', 'ROLE_IN_CONTAINER_D');
CALL C#DI.REVOKE_CONTAINER_SCHEMA_ROLES(#ROLES,
_SYS_DI.T_NO_PARAMETERS, ?, ?, ?);
DROP TABLE #ROLES;
```

Related Information

REVOKE_CONTAINER_SCHEMA_PRIVILEGES [page 197]
GRANT_CONTAINER_SCHEMA_ROLES [page 200]

5.3.3 HDI Container Views

Display information about calls made with the HDI container API.

In SAP HANA Deployment Infrastructure (HDI), an HDI container administrator is responsible for configuring and controlling access to an HDI container; the SQL application-programming interfaces (APIs) for content development in an HDI container are in the container's corresponding schema. For example, the SQL APIs for the HDI container `C` are in the schema `C#DI`. The following table lists the views that you can use to display information about calls made with the HDI container API:

SQL Views for HDI-Container Content-Development API Calls

HDI Container SQL View	Description
`M_JOBS`	Shows information about the progress of the individual jobs of a MAKE operation
`M_MESSAGES`	Shows all recent messages for API calls to the specified HDI container
`M_OBJECTS`	Shows the database objects in the run-time schema of an HDI container.
`M_ROLES`	Shows all the roles that are deployed in the specified HDI container

Related Information

SAP HDI Container Content Development with the HDI API [page 135]
The HDI Container API [page 141]

5.3.3.1 M_JOBS

View information about the progress of jobs belonging to a `MAKE` operation.

The SAP HDI Container API includes the `M_JOBS` view, which enables you to display information about the progress of the individual jobs that belong to a `MAKE` operation.

M_JOBS

Name	Data Type	Description
REQUEST_ID	BIGINT	The unique ID of the API call that produced this message. This ID is always the same for messages that originate from the same API call.
USER_NAME	NVARCHAR(256)	The user who called the (`ASYNC_`)
APPUSER_NAME	NVARCHAR(256)	The application user who called the (`ASYNC_`)`MAKE API`
CONNECTION_ID	INTEGER	The identifier of the connection that started the (`ASYNC_`)`MAKE`
START_TIME-STAMP_UTC	TIMESTAMP	The UTC time when the (`ASYNC_`)`MAKE` was started
JOB_START_TIME-STAMP_UTC	TIMESTAMP	The UTC time when the current job started
CURRENT_JOB	BIGINT	The number of the currently running job
NUM_JOBS	BIGINT	The total number of jobs that will run
STATUS	NVARCHAR(256)	The current status of the job, for example: Finished, Committed, Simulated, Rolled back, Post commit steps
UPDATE_TIME-STAMP_UTC	TIMESTAMP	UTC time when the job information was last updated
INTERNAL_CONNEC-TION_ID	INTEGER	The identifier of the internal connection used by HDI
INTERNAL_CONNEC-TION_START_TIME	TIMESTAMP	The time when the internal connection started
IS_INTERNAL_CON-NECTION_ACTIVE	VARCHAR(5)	TRUE (or FALSE): indicates that the job is still running (or not)

Related Information

M_ROLES [page 209]
M_MESSAGES [page 207]

5.3.3.2 M_MESSAGES

Shows all recent messages for API calls to an HDI container.

The SAP HDI Container API includes the `M_MESSAGES` view, which enables you to display information about all the recent messages for API calls to the specified HDI container.

M_MESSAGES

Name	Data Type	Description
REQUEST_ID	BIGINT	The unique ID of the API call that produced this message. This ID is always the same for messages that originate from the same API call.
ROW_ID	BIGINT	An increasing number representing the line number of the message log
LEVEL	INTEGER	The indentation level of the message (used for better visual representation)
TYPE	NVARCHAR(32)	The type of message: • SUMMARY: summary of the API call • HDI: message from HDI itself • PLUGIN: message from an HDI plug-in
LIBRARY_ID	NVARCHAR(256)	(**optional**) The ID of the library (for messages from a plug-in)
PLUGIN_ID	NVARCHAR(256)	(**optional**) The ID of the plug-in (for messages from a plug-in)
PATH	NVARCHAR(511)	(**optional**) The path to the artifact that is being processed
SEVERITY	NVARCHAR(16)	The severity of the message (INFO, WARNING, ERROR)
MESSAGE_CODE	BIGINT	A unique code corresponding to the `MESSAGE` field
MESSAGE	NVARCHAR(5000)	The text of the message
LOCATION	NVARCHAR(64)	(**optional**) The position (line:column) within the artifact that the message refers to
LOCATION_PATH	NVARCHAR(256)	(**optional**) XPath expression within the artifact that the message refers to
TIMESTAMP_UTC	TIMESTAMP	The time when the message was created

Related Information

M_JOBS [page 206]
M_ROLES [page 209]

5.3.3.3 M_OBJECTS

Shows the database objects in the run-time schema of an HDI container.

The SAP HDI Container API includes the `M_OBJECTS` view, which shows the database objects that are present in the container's run-time schema. This includes all objects that have been deployed as well as all objects that have been created manually. The view also shows information about the validity of the objects and the object's owners. This view is intended to help troubleshoot issues that may arise from objects becoming invalid, deleted, or created with the wrong user.

M_OBJECTS

Name	Data Type	Description
CONTAINER_NAME	NVARCHAR(64)	The name of the HDI container hosting the objects
RE-MOTE_SOURCE_NAME	NVARCHAR(256)	The object's remote source name, if applicable
DATABASE_NAME	NVARCHAR(256)	The object's database name
SCHEMA_NAME	NVARCHAR(256)	The object's database schema name
OBJECT_NAME	NVARCHAR(256)	The name of the container object
OBJECT_TYPE	VARCHAR(32)	The type of the container object
PATH	NVARCHAR(511)	The object's file path, if it is an object deployed by the container
IS_VALID	NVARCHAR(5)	The object's validity ("TRUE" or "FALSE")
OBJECT_VERSION	BIGINT	The metadata version of the container object
OBJECT_OID	BIGINT	The object's ID
OWNER_NAME	NVARCHAR(256)	The name of the owner of the container object
DEPENDENT_OB-JECT_NAME	NVARCHAR(256)	The name of the object that the current object depends on, for ex-ample, a synonym if the current object is the synonym's target ob-ject

Examples

Select all objects that were manually created in run-time schema of the container "C":

```
SELECT * FROM C#DI.M_OBJECTS WHERE PATH IS NULL
```

Select all objects in container "C" which are no longer owned by the container's object owner:

```
SELECT * FROM C#DI.M_OBJECTS WHERE PATH IS NOT NULL AND OWNER_NAME != 'C#OO'
```

Select all objects that should be deployed in container “C”, but have been manually deleted:

```
SELECT * FROM C#DI.M_OBJECTS WHERE PATH IS NOT NULL AND OBJECT_TYPE IS NULL
```

Select all invalid objects in container “C”:

```
SELECT * FROM C#DI.M_OBJECTS WHERE IS_VALID='FALSE'
```

Related Information

HDI Container Views [page 205]

5.3.3.4 M_ROLES

Show the roles that are deployed in an HDI container.

The SAP HDI Container API includes the `M_ROLES` view, which enables you to display information about the individual roles deployed in an HDI container.

M_ROLES

Name	Data Type	Description
ROLE_NAME	NVARCHAR(256)	The name of the role
ROLE_ID	BIGINT	The role ID
ROLE_MODE	NVARCHAR(5)	The mode of the role: '`LOCAL`'
GLOBAL_IDENTITY	NVARCHAR(256)	The identity specified for role with `ROLE_MODE GLOBAL`
CREATOR	NVARCHAR(256)	The name of the user who created the role
START_TIME-STAMP_UTC	TIMESTAMP	The UTC time when the `(ASYNC_)MAKE` was started
CREATE_TIME	TIMESTAMP	The time when the role was created
CONTEXT	NVARCHAR(2048)	The context for which the role is valid

Related Information

M_JOBS [page 206]
M_MESSAGES [page 207]

6 SAP HDI Artifact Types and Build Plug-ins Reference

The SAP HANA Deployment Infrastructure (HDI) supports a wide variety of database artifact types, for example, tables, types, views.

In SAP HDI, design-time artifacts are distinguished by means of a unique file suffix that must be mapped to a corresponding build plug-in to enable deployment to an HDI container. The following example of an abbreviated HDI configuration file (`.hdiconfig`) illustrates how the design-time artifact types `.hdbcalculationview` and `.hdbcds` are mapped to their corresponding HDI build plug-in:

Code Syntax

The HDI Container Configuration File (`.hdiconfig`)

```
{
  "plugin_version": "2.0.40.0",  // optional, defaults to 0.0.0.0
  "file_suffixes" : {
    "hdbsynonym" : {
      "plugin_name" : "com.sap.hana.di.synonym"},
    "hdbview" : {
      "plugin_name" : "com.sap.hana.di.view"},
    "hdbcalculationview" : {
      "plugin_name" : "com.sap.hana.di.calculationview"},
    "hdbprocedure" : {
      "plugin_name" : "com.sap.hana.di.procedure"},
    "<file suffix 2>" : {
      "plugin_name"   : "com.sap.hana.di.<plugin name>",
      "plugin_version": "<plugin_version>"} // optional; overrides global
version
    }
}
```

! Restriction

All “hdi*” file-suffixes (for example, “`.hdiconfig`” or “`.hdinamespace`”) are reserved by the SAP HANA Deployment Infrastructure (HDI) and cannot be (re)configured.

HDI Plug-in Version

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.

i Note

The three-level format used for the plug-in version in SAP HANA 1.0, for example, `13.1.0`, is still valid; values in the old three-level format are internally converted to the appropriate four-level format used 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 plug-ins 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.

HDI Plug-in Types

The following table lists in alphabetical order the design-time artifacts you can develop and deploy with the SAP HANA Deployment Infrastructure (HDI). For more information about the syntax required for a particular type of design-time artifact and the configuration of the corresponding build plug-in, choose the artifact type in the following table.

> **i Note**
>
> Some plug-in libraries are not available by default, for example, for public and `_SYS_BIC` synonyms. The list of plug-in libraries that are available by default in an HDI container must be configured by the HDI Container administrator, as described in *Maintaining HDI Containers* in the *SAP HANA Administration Guide*. For more information, see *Related Information* below.

Design-Time HDI Artifact Types

A - L	M- T	T - Z
`CSV` *	`Migration Table`	`Table Data Properties`
`.? (txt, copy only)`	`Procedure`	`Table Data Tags`
`AFL Lang Procedure` **	`Projection View`	`Table Type`
`Analytic Privilege`	`Projection View Configuration`	`Text Analysis Configuration`
`App. Time-Period Table`	`Public Synonym`	`Text Analysis Dictionary`
`Calculation View`	`Replication Task`	`Text Extraction Rule`
`Core Data Services`	`Result Cache`	`Text Extraction Include`
`Constraint`	`Role`	`Text Extraction Lexicon`
`DocStore Collection`	`Role Configuration`	`Text Mining Configuration`
`Drop/Create Table`	`Search Rule Set`	`Trigger`

A - L	M- T	T - Z
Flowgraph	Sequence	View
Full Text Index	Synonym	Virtual Function
Function	Synonym Configuration	Virtual Function Configuration
Graph Workspace	Statistics	Virtual Package
Index	Structured Privilege	Virtual Procedure
Map Reduce Job ***	_SYS_BIC Synonym	Virtual Procedure Configuration
Jar	System Versioning Table	Virtual Table
Library	Table	Virtual Table Configuration
Logical Schema Definition	Table Data	-

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 SAP HDI Container Configuration File [page 118]
Run-Time Name Spaces in SAP HDI [page 126]
Maintaining SAP HDI Containers [page 90]

6.1 Application Time-Period Table Plug-in (.hdbapplicationtime)

Transforms a design-time application time-period table into a database table object with application-time period.

The Application-Time Period Table plug-in adds an `APPLICATION_TIME` period over the specified "`validfrom`" and "`validto`" columns of a column-store table, which adds an implicit constraint "`validfrom`

`< validto`". After adding the time period to the table, temporal features such as `UPDATE FOR PORTION OF` are enabled.

→ Tip

For more information about application-time period tables, see the *SAP HANA SQL and System Views Reference*, in *Related Information* below, and in particular the section describing the `<application_time_period_configuration>` in *CREATE TABLE Statement (Data Definition)*.

Example Artifact Code

The following code shows a simple example of an application time-period table definition for HDI:

Code Syntax

`/src/CUSTOMERS.hdbapplicationtime`

```
APPLICATION TIME "CUSTOMERS"(validfrom, validto)
```

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`

```
"hdbapplicationtime" : {
   "plugin_name" : "com.sap.hana.di.applicationtime",
   "plugin_version": "2.0.40.0"
}
```

Related Information

CREATE TABLE Statement (Data Definition)

6.2 AFL Language Procedures (.hdbafllangprocedure)

Transform a design-time Application Function Library (AFL) language procedure resource into a corresponding database procedure object.

The AFLLang Procedure plug-in transforms a design-time Application Function Library (AFL) language-procedure resource (defined in a `.hdbafllangprocedure` artifact) into the corresponding database procedure object that wraps the specified AFL function based on the given table types for the parameters. The file format for `.hdbafllangprocedure` artifacts uses the JSON syntax.

Example Artifact Code

The following code shows a simple example of an AFL language procedure definition for 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`

```
"hdbafllangprocedure" : {
   "plugin_name" : "com.sap.hana.di.afllangprocedure",
   "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]

6.3 Analytic Privileges (.hdbanalyticprivilege)

Transform a design-time XML-based analytic-privileges resource into a analytic-privileges object in the database.

The analytic privileges plug-in transforms a design-time XML-based analytic-privileges resource (defined in a `.hdbanalyticprivilege` artifact) into an analytic-privileges object in the database. The file format required for the design-time `.hdbanalyticprivilege` artifact 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": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]

6.4 Calculation Views (.hdbcalculationview)

Transform a design-time calculation view description into a set of view database objects.

The Calculation View plug-in transforms a design-time calculation view described in a `.hdbcalculationview` artifact into a set of view database objects. The file format used in the `.hdbcalculationview` artifact is based on XML 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.hdbcalculationview`

```
<?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`

```
"hdbcalculationview" : {
   "plugin_name" : "com.sap.hana.di.calculationview",
   "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]

6.5 Constraints (.hdbconstraint)

Transform a design-time constraint into a constraint on database tables.

The foreign-key constraint plug-in transforms a design-time constraint (defined in a `.hdbconstraint` artifact) into a constraint on database tables. The file format required for the design-time `.hdbconstraint` artifact uses a DDL-style syntax that 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 example below.

! Restriction

The constraint plug-in only supports `FOREIGN KEY` constraints. Unique constraints are handled by the `.hdbindex` plugin.

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 `.hdbindex` 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`

```
"hdbconstraint" : {
   "plugin_name" : "com.sap.hana.di.constraint",
   "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]
Creating Database Artifacts with SQL DDL in XS Advanced (Developer Guide)

6.6 Copy Only (.txt)

Transform an arbitrary design-time resource into a deployed object.

The copy-only plug-in transforms an arbitrary design-time resource (defined in a `.txt` artifact) into a deployed object by copying it from the container's work file system to the deployed file system.

> **! Restriction**
>
> This type of design-time resource does not have a representation inside the run-time container. The resource is only accessible by means of SAP HANA DI APIs.

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`

```
"txt" : {
   "plugin_name" : "com.sap.hana.di.copyonly",
   "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]

6.7 Core Data Services (.hdbcds)

Transform a design-time SAP HANA CDS resource into the database objects defined in the CDS document.

The SAP HANA Core Data Services (CDS) plug-in transforms a design-time SAP HANA CDS resource (defined in a `.hdbcds` artifact) into the database objects defined in the CDS document, for example, tables, views, types, etc. It is possible to migrate tables from `.hdbcds` to `.hdb[migration]table`. For more information about the syntax required when creating database artifacts with CDS in XS advanced, see *Related Information* below.

! 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.hdbcds`

```
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 plug-in configuration should look like the following example:

Code Syntax

`.hdiconfig`

```
"hdbcds" : {
    "plugin_name" : "com.sap.hana.di.cds",
    "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]
Creating Database Artifacts with CDS (XS Advanced Developer Guide)

6.8 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`

```
"hdbcollection" : {
   "plugin_name" : "com.sap.hana.di.collection",
   "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]

SAP HANA JSON Document Store Guide

6.9 Flowgraph (.hdbflowgraph)

Transform a design-time, flow-graph description into a corresponding set of database procedure or task objects.

A flow graph performs a configurable series of data transformation operations, such as joining, filtering, cleansing, masking, aggregating, and so on.

→ Tip

For more information about creating `.hdbflowgraph` artifacts, see the *Modeling Guide for SAP HANA Smart Data Integration and SAP HANA Smart Data Quality*.

The file format uses an XML-based syntax.

Example Artifact Code

Code Syntax

`/src/SOURCE_TARGET.hdbflowgraph`

```
<?xml version="1.0" encoding="UTF-8"?>
<flowgraph:ContainerNode xmlns:xmi="http://www.omg.org/XMI" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance" xmlns:flowgraph="http://www.sap.com/ndb/
flowgraph/1.0" xmi:version="2.0" xmi:id="48c98de995349a58f789161dffecddbe"
name="Sample" runtimeBehavior="BATCH_TASK">
    <annotations key="sap.afm.layout"
xmi:id="1e9bdee19337604a2fe702f69b5602b8">
        <annotations xmi:id="f1eade7e1b55c09f00dee393b7a2ea0e" key="x"
value="10">
        </annotations>
        <annotations xmi:id="1641fc95f79b40e0bf4bf5cb21f81bbb" key="y"
value="10">
        </annotations>
        <annotations xmi:id="766ff0b1bdcb118aa9b23388e8cdfdd3" key="width"
value="100">
        </annotations>
        <annotations xmi:id="2882970d6e43a4d3c8091c69a6f85d8f" key="height"
value="84">
        </annotations>
    </annotations>
    <annotations xmi:id="de380e1c77c2bfc8d64806afa2c90dae"
key="sap.afm.palette">
        <annotations xmi:id="64399b3a3b9f04990acc30077471df97"
key="additions" value=""/>
    </annotations>
    <annotations xmi:id="d711eace8b9a9093e2960ff8a5f77907"
key="sap.afm.nodeType" value="ContainerNode">
    </annotations>
    <annotations xmi:id="d1085bb9c5fb4b0ab561e8f3ea12ad34"
key="sap.afm.nodeInputs">
    </annotations>
    <annotations xmi:id="40a784946ecb249542780fa272c213ce"
key="sap.afm.nodeOutputs">
    </annotations>
    <tableMappings xmi:id="0d7122ab56cb9e5b25e8a9705e0a0cca"
source="26ff77a7f671674044822dacec0260c4"
target="1e7e63398016ec4631cdc305771a65c2">
        <attributeMappings xmi:id="6ecf80b841d18839a47c20e24e3c931a"
source="4e3587cdc45c9261038d915014bee06e"
target="30877cc439230ef4ea267db9e79cfe24">
        </attributeMappings>
        <attributeMappings xmi:id="643541c94681ac60cedceff184295921"
source="6529b86aca8a98fcc9f35dbf376cb29e"
target="3043f276bd90ef278e635e92797de2fd">
        </attributeMappings>
        <attributeMappings xmi:id="69c3940901d8a2604db6992ae83f35f4"
source="8e795a3b4ae743a73947278478b782b8"
target="247f3c3cc1f79b1f8fa99fddf4c8a083">
        </attributeMappings>
        <attributeMappings xmi:id="de651bb4a43e3e24f687036775e1b6b5"
source="5876cbdaef85a0ca7d00ee3693bab1d1"
target="072355443d648d4cbe94fa9566a6f513">
        </attributeMappings>
    </tableMappings>
    <nodes xmi:id="5afdacf9175d180411812b177312363e" name="DataSource"
catalogObjectName="avijit.hdbav::qa_dept.Entity1" type="DATABASE_TABLE"
xsi:type="flowgraph:DataNode" dataLayout="COLUMN">
        <annotations key="sap.afm.layout"
xmi:id="b207966200646c7c6d93b2f1be8cbf9a">
            <annotations xmi:id="ea0f7461349e1f6af0e7d4db3394e15c" key="x"
value="12">
            </annotations>
            <annotations xmi:id="e8a54115896fa0cf7460f822fcdcf54c" key="y"
value="12">
            </annotations>
            <annotations xmi:id="db5fda29908c64f41d61e056c1b70fc6"
key="width" value="120">
            </annotations>
            <annotations xmi:id="0e2701c55e90074e952edcdcb46a980a"
key="height" value="115">
            </annotations>
        </annotations>
        <annotations xmi:id="fb0a48789c4bcb827f490dfa2284fba9"
key="sap.afm.description">
        </annotations>
        <annotations xmi:id="99ad396311bf5361f547d17b23de5b00"
key="sap.afm.nodeType" value="DataNode">
        </annotations>
        <annotations xmi:id="8863401ed0df9b898f8c41e2b03ace4c"
key="sap.afm.nodeInputs">
        </annotations>
        <annotations xmi:id="f84979354b412a530c07410516b03183"
key="sap.afm.nodeOutputs">
        </annotations>
        <annotations xmi:id="a2af10ac10cf0fbe01d0a56390868420"
key="sap.afm.displayName" value="Data Source">
        </annotations>
        <annotations xmi:id="5eddebba1b0906bd1db0bd67ffab65d5" key="columns"
value="[{&quot;name&quot;:&quot;DEPTNO&quot;,&quot;type&quot;:&quot;DECIMAL&qu
ot;,&quot;length&quot;:
28,&quot;nullable&quot;:&quot;false&quot;,&quot;primarykey&quot;:true},
{&quot;name&quot;:&quot;DNAME&quot;,&quot;type&quot;:&quot;NVARCHAR&quot;,&quo
t;length&quot;:14,&quot;nullable&quot;:&quot;true&quot;},
{&quot;name&quot;:&quot;LOC&quot;,&quot;type&quot;:&quot;NVARCHAR&quot;,&quot;
length&quot;:14,&quot;nullable&quot;:&quot;true&quot;},
{&quot;name&quot;:&quot;ST&quot;,&quot;type&quot;:&quot;NVARCHAR&quot;,&quot;l
ength&quot;:2,&quot;nullable&quot;:&quot;true&quot;}]">
        </annotations>
        <outputs xmi:id="26ff77a7f671674044822dacec0260c4" name="DataSource">
```

```
            <attributes xmi:id="4e3587cdc45c9261038d915014bee06e"
name="DEPTNO" type="DECIMAL" nullable="false" length="28">
                <annotations xmi:id="dd342f537524896cf2219533b48d3adf"
key="sap.im.primaryKey" value="true"/>
            </attributes>
            <attributes xmi:id="6529b86aca8a98fcc9f35dbf376cb29e"
name="DNAME" type="NVARCHAR" nullable="true" length="14">
            </attributes>
            <attributes xmi:id="8e795a3b4ae743a73947278478b782b8" name="LOC"
type="NVARCHAR" nullable="true" length="14">
            </attributes>
            <attributes xmi:id="5876cbdaef85a0ca7d00ee3693bab1d1" name="ST"
type="NVARCHAR" nullable="true" length="2">
            </attributes>
        </outputs>
    </nodes>
    <nodes xmi:id="fb754fcf4407048bd5f16fb1069fc611" name="DataTarget"
catalogObjectName="DataTarget" type="TEMPLATE_TABLE"
xsi:type="flowgraph:DataNode">
        <annotations key="sap.afm.layout"
xmi:id="0cb7ed884bb690b9be60d8042c8c2b85">
            <annotations xmi:id="ed214d21c1fce72e14aad7d17cf27638" key="x"
value="193">
            </annotations>
            <annotations xmi:id="a0fe6c35b6e73888cd54d8c2b559af99" key="y"
value="12">
            </annotations>
            <annotations xmi:id="04e447789ef9d1e98c177274a411fe7e"
key="width" value="120">
            </annotations>
            <annotations xmi:id="63f5a6ec04b1e54aece877e247259027"
key="height" value="115">
            </annotations>
        </annotations>
        <annotations xmi:id="03ff7aba46adadd41f0d7af8fdd22382"
key="sap.afm.description">
        </annotations>
        <annotations xmi:id="bd2f87dd6344e25ae338b3f46e882508"
key="sap.afm.nodeType" value="DataNode">
        </annotations>
        <annotations xmi:id="58bb3d83aa10a12f684757235bf0fd09"
key="sap.afm.nodeInputs">
        </annotations>
        <annotations xmi:id="1792d885b494117b8b094374ea6dea86"
key="sap.afm.nodeOutputs">
        </annotations>
        <annotations xmi:id="bba0b5c651d69b745a803da7cff4a6e1"
key="sap.afm.displayName" value="Data Target">
        </annotations>
        <inputs xmi:id="1e7e63398016ec4631cdc305771a65c2" name="DataTarget">
            <attributes xmi:id="30877cc439230ef4ea267db9e79cfe24"
name="DEPTNO" type="DECIMAL" nullable="false" length="28">
                <annotations xmi:id="8a05e20a4791e8723346be4819eb2afd"
key="sap.im.primaryKey" value="true"/>
            </attributes>
            <attributes xmi:id="3043f276bd90ef278e635e92797de2fd"
name="DNAME" type="NVARCHAR" nullable="true" length="14">
            </attributes>
            <attributes xmi:id="247f3c3cc1f79b1f8fa99fddf4c8a083" name="LOC"
type="NVARCHAR" nullable="true" length="14">
            </attributes>
            <attributes xmi:id="072355443d648d4cbe94fa9566a6f513" name="ST"
type="NVARCHAR" nullable="true" length="2">
            </attributes>
        </inputs>
    </nodes>
</flowgraph:ContainerNode>
```

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`

```
"hdbflowgraph" : {
   "plugin_name"    : "com.sap.hana.di.flowgraph",
   "plugin_version": "2.0.40.0"
}
```

6.10 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 plug-in transforms a full-text index resource described in a `.hdbfulltextindex` design-time artifact 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

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 and SAP HANA Deployment Infrastructure (HDI):

Code Syntax

`/src/CUSTOMER_NAME_FTI.hdbfulltextindex`

```
FULLTEXT INDEX "com.sap.hana.example::CUSTOMER_NAME_FTI"
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

```
"hdbfulltextindex" : {
   "plugin_name" : "com.sap.hana.di.fulltextindex",
   "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]

6.11 Functions (.hdbfunction)

Transform a design-time function resource into a function database object.

The Function plug-in transforms a design-time function resource described in the `.hdbfunction` artifact into a function database object. The file format required for the `.hdbfunction` artifact uses a DDL-style syntax that is equivalent to the syntax of the corresponding SQL command `CREATE FUNCTION SQL`, without the leading “`CREATE`”.

Note

The Function plug-in handles scalar and table functions, too. Virtual functions, defined in `.hdbvirtualfunction` artifacts, are handled by the Virtual Function plug-in.

Example Artifact Code

The following code shows a simple example of a **scalar** function definition for XS advanced and SAP HANA Deployment Infrastructure (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 and SAP 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 plug-in configuration should look like the following example:

Code Syntax

.hdiconfig

```
"hdbfunction" : {
   "plugin_name" : "com.sap.hana.di.function",
   "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]

6.12 Graph Workspaces (.hdbgraphworkspace)

Transform a design-time graph-workspace resource into a graph-workspace object in the database.

The graph workspace plug-in transforms a design-time graph workspace resource (described in a `.hdbgraphworkspace` artifact) into a graph workspace object in the database. The file format required for the `.hdbgraphworkspace` artifact uses a DDL-style syntax that is equivalent to the 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 design-time graph workspace definition for 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 plug-in configuration should look like the following example:

Code Syntax

.hdiconfig

```
"hdbgraphworkspace" : {
   "plugin_name" : "com.sap.hana.di.graphworkspace",
   "plugin_version": "2.0.0.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]

6.13 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 (for `.hdbvirtualpackage*` artifacts). 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 transforms a design-time Hadoop map-reduce-job resource (described in either a `.hdbmrjob` or a `.jar` artifact) into a virtual-function-package database object with adapter type "HADOOP". The Hadoop MRJ plug-in supports the following 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 plug-in 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": "2.0.40.0"
}
```

```
"jar" : {
   "plugin_name" : "com.sap.hana.di.virtualpackage.hadoop",
   "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]
Virtual Packages (.hdbvirtualpackage) [page 297]

6.14 Indexes (.hdbindex)

Transform a design-time index resource into an index on a database table.

The index plug-in transforms a design-time index resource (`.hdbindex`) into an index on a database table. The file format required for `.hdbindex` artifacts uses a DDL-style syntax that is equivalent to the corresponding syntax in the SQL command `CREATE INDEX`, although without the leading "`CREATE`" command. For more information about the syntax required when creating database artifacts with SQL DDL in XS advanced, see *Related Information* below.

Example Artifact Code

The following code shows a simple example of a design-time index definition for XS advanced and 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 plug-in configuration should look like the following example:

Code Syntax

.hdiconfig

```
"hdbindex" : {
   "plugin_name" : "com.sap.hana.di.index",
   "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]
Creating Database Artifacts with SQL DDL in XS Advanced (Developer Guide)

6.15 Libraries (.hdblibrary)

Transform a design-time library resource into a library database object.

The library plug-in transforms a design-time library resource (hdblibrary) into a library database object. The file format required for .hdblibrary artifacts 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, SQLScript.

Example Artifact Code

The following code shows a simple example of a library for HDI written in SQLScript:

Code Syntax

/src/mylibrary.hdblibrary

```
LIBRARY "com.sap.hana.example::MYLIBRARY"
LANGUAGE SQLSCRIPT
AS
BEGIN
  PUBLIC VARIABLE A CONSTANT INT = 1;
END
```

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

```
"hdblibrary" : {
   "plugin_name" : "com.sap.hana.di.library",
   "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]

6.16 Logical Schema Definition (.hdblogicalschema)

Transforms a design-time logical-schema definition into database objects that can be consumed by synonyms and so on.

A logical schema defines a logical schema name and a mapping to the real schema name and database. The logical schema can then be used by synonyms etc.

i Note

The target schema etc. specified in the `.hdblogicalschema` definition must already exist.

Synonym and projection-view configurations (as described in the corresponding sections for `.hdbsynonymconfig` and `.hdbprojectionviewconfig` artifacts) 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.

Example Artifact Code

Logical schema definitions have the following JSON format.

i Note

A logical schema definition file can contain multiple definitions.

Code Syntax

Logical Schema Definition (`.hdblogicalschema`)

```
{
 "<logical schema 1>" : {
   "target": {
     "schema" : "<target schema>"
   }
 },
 "<logical schema 2>" : {
   "target": {
     "remote"   : "<remote source name>",
     "database" : "<database name>",
     "schema"   : "<target schema>"
    }
 },
 <...>
}
```

The `“schema”` property of a target description is mandatory; it cannot be empty and the declared value must be different from the container schema. In addition, if you also use the properties `"remote"` or `"database"`, bear in mind that they might be not applicable for every plug-in artifact. For example, the Projection View plug-in (`.hdbprojectionview`) does not support the `"remote"` property; the Synonym plug-in, however, accepts either the "remote" or "database" property, but not both properties at the same time. For this reason, it is recommended to specify different logical schema definitions for use with different plug-ins.

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

```
"hdblogicalschema" : {
   "plugin_name" : "com.sap.hana.di.logicalschema",
   "plugin_version": "2.0.40.0"
}
```

Related Information

Projection Views (.hdbprojectionview) [page 237]
Synonyms (.hdbsynonym and .hdbsynonymconfig) [page 259]
Virtual Tables (.hdbvirtualtable) [page 294]

6.17 Migration Tables (.hdbmigrationtable)

The migration-table plugin transforms a design-time table resource into a table database object.

In contrast to the table plug-in (.hdbtable), the migration-table plug-in (.hdbmigrationtable) uses explicit versioning and migration tasks, which means that the modifications of the database table are explicitly specified in the design-time file and carried out on the database table exactly as specified, without incurring the cost of an internal table-copy operation such as the one performed by the .hdbtable plug-in. This behavior makes the .hdbmigrationtable plug-in especially useful for tables that contain a lot of data. When a new version of an already existing table is deployed, the plug-in performs the migration statements that are missing for the new version.

Note

It is possible to migrate from .hdbmigrationtable to .hdbtable.

Migrations

The migration-table file format uses a syntax that is similar in style to the Data Definition Language (DDL); the migration combines the most recent definition of the table and the migration statements required per table version to transform the table from the previous version to the new (migrated) version. Versions with the same definition as the previous version can be omitted. Only `ALTER TABLE`, `RENAME COLUMN` and `COMMENT ON [TABLE|COLUMN]` statements are supported for the migrations. The migration-table plug-in does not support statements written in the Data Manipulation Language (DML), nor any of the following statements:

- `ALTER TABLE ... PARTITION`
- `ALTER TABLE ... [PARTITION ] ... LOADABLE`
- `ALTER TABLE ... OWNER TO`
- `ALTER TABLE ... MOVE TO`
- `ALTER TABLE ... ENABLE|DISABLE TRIGGER`

After finishing the migrations, the migration-table plug-in checks whether the migrated table matches the new definition.

Development Versions and Development Mode

For development purposes, it is possible to specify a development version with a separate definition, which is used as the base for the current migration development. During deployment in a development system, the migration-table plug-in (running in development mode) drops the existing table and creates the development version. Then the plug-in runs the migration starting at the development version instead. The development version is only considered if the build parameter `development_mode` is set to “true”

> i Note
>
> In development mode all data stored in the table is lost.

General Limitations

It is important to bear in mind the following requirements and limitations:

- Table types are limited to `ROW` and `COLUMN`, and specifying the type is mandatory. The following code shows a simple example of a migration table definition for HDI:
- A table must be self-contained and cannot reference another table. For this reason, it is not possible to define a table “like”The following code shows a simple example of a migration table definition for another table (`TABLE A LIKE B`), or a table based on a query (`TABLE A AS SELECT … FROM B`).
- Text-analysis properties `CONFIGURATION` and `MINING CONFIGURATION` for a table column are not supported. To use text-analysis configurations for text columns, you need to define a separate full-text index.
- Flexible tables (using the `WITH SCHEMA FLEXIBILITY` clause) and `HISTORY` tables are not supported.
- Named constraints are not supported.

- Client-side encryption (`CLIENTSIDE ENCRYPTION`) is not supported.
- Virtual tables are handled by the virtual-table build plug-in (`hdbvirtualtable`).
- Foreign-key constraints are handled by the constraint build plug-in (`hdbconstraint`).

Table and column comments can be defined in-place by using the `COMMENT` keyword. Tables also support forward declarations for database-level associations using the `WITH ASSOCIATIONS` clause. In this scenario, the table plug-ins creates an additional intermediate "validate" artifact as part of the deployment, which positions 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 performs a validation of the forward declared associations and issues a warning if an association is not valid.

i Note

It is not possible to use the migration-table plug-in to set up a system-versioned table or an application-time-period table; separate plug-ins are available for these purposes. For more details, see *Related Information* below.

Although it is possible to specify an initial partitioning for the first version of the table in the design-time table resource, it is recommended to only perform partitioning on the run-time table object. For changes to the partitioning of the run-time table object, a user with `ALTER` privileges for the object is required. Partitioning in a migration is not permitted.

Example Artifact Code

Code Syntax

`/src/CUSTOMERS.hdbmigrationtable`

```
== version=28
COLUMN TABLE "com.sap.hana.example::CUSTOMERS" (
  "ID" INTEGER,
  "NAME" NVARCHAR(256),
  "ACTIVE" TINYINT,
  "COUNTRY" NVARCHAR(256),
  PRIMARY KEY ("ID")
)
== dev-version=7
COLUMN TABLE "com.sap.hana.example::CUSTOMERS" (
  "ID" INTEGER,
  "FULLNAME" NVARCHAR(256),
  PRIMARY KEY ("ID")
)
== migration=10
ALTER TABLE "com.sap.hana.example::CUSTOMERS" ADD ("ACTIVE" INT);
RENAME COLUMN "com.sap.hana.example::CUSTOMERS"."FULLNAME" TO "NAME";
== migration=19
ALTER TABLE "com.sap.hana.example::CUSTOMERS" ADD ("COUNTRY" NVARCHAR(256));
ALTER TABLE "com.sap.hana.example::CUSTOMERS" ALTER ("ACTIVE" TINYINT);
```

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`

```
"hdbmigrationtable" : {
   "plugin_name" : "com.sap.hana.di.table.migration",
   "plugin_version": "2.0.40.0"
}
```

Related Information

Application Time-Period Table Plug-in (.hdbapplicationtime) [page 212]
System Versioning Table (.hdbsystemversioning) [page 264]

6.18 Procedures (.hdbprocedure)

Transform a design-time procedure resource into a procedure database object.

The procedure plug-in transforms a design-time procedure resource (described in a `.hdbprocedure` artifact) into a procedure database object. The file format required for the `hdbprocedure` artifact uses a DDL-style syntax that is equivalent to the syntax of the corresponding SQL command `CREATE PROCEDURE`, but without the leading “`CREATE`” command. The procedure plug-in supports the following languages for the `hdbprocedure` artifact: SQLScript, L, and R. AFLLang procedures are handled by the “AFLLang Procedure” plug-in.

Note

If the procedure is implemented in R, then the container’s object owner (`<container>#OO`) needs the privilege `CREATE R SCRIPT`.

Example Artifact Code

The following code shows a simple example of a procedure definition for XS advanced and HDI:

Code Syntax

/src/SELECT_CUTOMER.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 plug-in configuration should look like the following example:

Code Syntax

.hdiconfig

```
"hdbprocedure" : {
   "plugin_name" : "com.sap.hana.di.procedure",
   "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]

6.19 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 (described in a `.hdbprojectionview` artifact) 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.

i 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. For more information about the syntax used in an SQL projection view, see *Related Information* below.

The following code shows a simple example of a projection-view definition for XS advanced and 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"   : "APPLICATON_B", // optional
      "object"   : "CUSTOMERS"
    }
  }
}
```

i 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_APPLICATON", // not in conjunction with
                                               // database or schema
      "object"    : "CUSTOMERS"
    }
  }
}
```

If the `"logical_schema"` field of a target description is defined, it is not possible to define either the `"database"` or the `"schema"` field. The `"logical_schema"` specifies a deployment dependency to a logical schema definition (described in the `.hdblogicalschema` artifact). The logical schema definition contains the schema name that is actually used, which must be different from the name of the container schema.

Note

It is also possible to include an (optional) `"database"` name. However, a `remote` source name is not allowed.

If the target description does not contain the `logical_schema` field and the `schema` field is omitted, too, then the projection view 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 plug-in configuration should look like the following example:

Code Syntax

`.hdiconfig`

```
"hdbprojectionview" : {
   "plugin_name" : "com.sap.hana.di.projectionview",
   "plugin_version": "2.0.40.0"
},
"hdbprojectionviewconfig" : {
   "plugin_name" : "com.sap.hana.di.projectionview.config",
   "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]
Logical Schema Definition (.hdblogicalschema) [page 232]
CREATE PROJECTION VIEW Statement (SQL and System Views Reference)

6.20 Public Synonym (.hdbpublicsynonym)

Create public synonyms that refer to database objects located in the target schema of the current container.

The Public Synonym plug-in 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 plug-in is intended for use in migration scenarios where an existing application uses public synonyms to accesses database objects. As a result, the corresponding plug-in 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, as described in the *Maintaining HDI Containers* section of the *SAP HANA Administration Guide*.

Example Artifact Code

The following code shows a simple example of a public-synonym definition for XS advanced and HDI:

Code Syntax

`src/a_synonym.hdbpublicsynonym`

```
{
  "<synonym 1>" : {
    "target": {
      "object" : "<the target object 1>"
    }
  },
  "<synonym 2>" : {
    "target": {
      "object" : "<the target object 2>"
    }
  },
  <...>
}
```

Like the normal synonym artifact, the design-time definition file for a public-synonym (`.hdbpublicsynonym`) can contain multiple definitions. The elements `<synonym_1>`, `<synonym_2>`, and so on, define the names of the public synonyms 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 name-space rules.

Sample Code

```
{
  "com.sap.hana.example::A_Public_Synonym" : {
    "target": {
      "object" : "com.sap.hana.example::A_TABLE"
    }
  }
}
```

i 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 plug-in configuration should look like the following example:

Code Syntax

.hdiconfig

```
"hdbpublicsynonym": {
    "plugin_name" : "com.sap.hana.di.publicsynonym",
    "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]
Maintaining HDI Containers (SAP HANA Admin Guide) [page 90]

6.21 Replication Task (.hdbreptask)

Transform a design-time replication task description into a corresponding set of database procedure or task operations.

A replication task retrieves data from one or more objects in a single remote source and populates one or more tables in SAP HANA.

→ Tip

For more information about creating `.hdbreptask` artifacts, see the *Modeling Guide for SAP HANA Smart Data Integration and SAP HANA Smart Data Quality*.

The file format uses an XML-based syntax.

Example Artifact Code

The following code shows a simple example of a design-time definition of a replication-task for XS advanced and HDI:

Code Syntax

`/src/A.hdbreptask`

```
<RepTask Name="file:/VTT_Demo/A.hdbreptask:workspace" Description=""
Type="REALTIME" RepVersion="2.1">
  <SourceObjects SourceType="REMOTE_OBJ" TTFullyQualifiedName="false"
VTFullyQualifiedName="false" RemoteSourceName="AHanaSource"
VirtualTableSchema="SYSTEM">
    <SourceObject
RemoteObjectUniqueName="&quot;IM_SERVICES&quot;.&quot;INPUT_PRIM_KEY&quot;"
SourceDisplayName="INPUT_PRIM_KEY">
    </SourceObject>
  </SourceObjects>
  <TargetObjects SchemaName="SYSTEM">
    <TargetObject Type="TABLE" ObjectName="IM_SERVICES_INPUT_PRIM_KEY"
DropTargetTableIfExists="TRUE">
      <TargetColumns>
        <TargetColumn Name="PRIM_KEY" Datatype="VARCHAR" Length="6"
Precision="6" Scale="0" Nullable="FALSE" PartOfPrimaryKey="TRUE">
        </TargetColumn>
        <TargetColumn Name="ADDRESS" Datatype="VARCHAR" Length="100"
Precision="100" Scale="0" Nullable="TRUE" PartOfPrimaryKey="FALSE">
        </TargetColumn>
        <TargetColumn Name="PERSON" Datatype="VARCHAR" Length="100"
Precision="100" Scale="0" Nullable="TRUE" PartOfPrimaryKey="FALSE">
        </TargetColumn>
      </TargetColumns>
    </TargetObject>
  </TargetObjects>
  <Mappings>
    <Mapping ObjectName="IM_SERVICES_INPUT_PRIM_KEY"
RemoteObjectUniqueName="&quot;IM_SERVICES&quot;.&quot;INPUT_PRIM_KEY&quot;"
FilterExpression="" ReplicationBehavior="InitLoadOnly"
VTObjectName="IM_SERVICES_INPUT_PRIM_KEY">
      <TargetColumns>
```

```
        <TargetColumn Name="PRIM_KEY"
ProjectionExpression="&quot;PRIM_KEY&quot;">
        </TargetColumn>
        <TargetColumn Name="ADDRESS"
ProjectionExpression="&quot;ADDRESS&quot;">
        </TargetColumn>
        <TargetColumn Name="PERSON" ProjectionExpression="&quot;PERSON&quot;">
        </TargetColumn>
      </TargetColumns>
    </Mapping>
  </Mappings>
</RepTask>
```

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`

```
"hdbreptask" : {
   "plugin_name"    : "com.sap.hana.di.reptask",
   "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]

6.22 Result Cache (.hdbresultcache)

Transform a DDL-based definition of a result cache into the corresponding catalog object.

The Result Cache plug-in transforms a design-time DDL-based definition of a result cache (`.hdbresultcache`) into a catalog-level result-cache definition. The file format required for the (`.hdbresultcache`) artifact uses a DDL-style syntax that is similar to the corresponding syntax for the `[ALTER VIEW | FUNCTION] <object> ADD CACHE` statement. The `.hdbresultcache` file must use an artificial cache name that includes the prefix `_SYS_CACHE#` followed by the name of the referenced view or function, as illustrated in the following example.

Example Artifact Code

The following code shows a simple example of a design-time definition of a result cache for XS advanced and HDI:

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 plug-in configuration should look like the following example:

Code Syntax

`.hdiconfig`

```
"hdbresultcache": {
    "plugin_name" : "com.sap.hana.di.resultcache",
    "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]

6.23 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 object. The format of the `.hdbrole` 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. For more information about the syntax required when creating database roles in XS advanced, see *Related Information* below.

Example Artifact Code

The following code shows a simple example of a role definition for XS advanced HDI.

→ Tip

The following code sample is interactive; click the property you are interested in learning about to display more detailed information about how to configure and use it.

Code Syntax

/src/myDesignTimeRole.hdbrole

```
{
  "role":{
    "pattern_escape_character": "\",
    "name":"RoleX",
    "global_roles":[
      "GLOBAL_ROLE_1",
      "GLOBAL_ROLE_2"
    ],
    "schema_roles":[
      {
        "schema_reference":"RoleSchemaRef1",
        "names": ["RoleA", "RoleB"]
      },
      {
        "schema_reference":"RoleSchemaRef2",
        "names": ["RoleB", "RoleC"]
      },
      {
        "schema_reference":"LogicalSchemaRef1",
        "names": ["RoleD", "RoleE"]
      }
    ],
    "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" ]
      },
      {
        "name":"<wildcard_object_identifier>",
        "type":"TABLE",
        "privileges":[ "SELECT" ],
        "pattern_mode": "include"
      }
    ],
    "schema_analytic_privileges":[
      {
        "schema_reference":"APSchemaRef1",
        "privileges": ["AP1", "AP2"],
        "privileges_with_grant_option": ["AP4"]
      },
      {
        "schema_reference":"APSchemaRef2",
        "privileges": ["AP1", "AP3"]
      },,
      {
        "schema_reference":"LogicalSchemaRef1",
        "names": ["RoleAP1", "RoleAP3"]
      }
    ]
  }
}
```

Mandatory Role Properties

The following table lists the properties that must be defined in a design-time role definition.

Mandatory 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.

role

The role object is mandatory; it is the root of the JSON data structure and an object with a single key role.

```
{
  "role":{
  ...
  }
}
```

pattern_escape_character

the "`pattern_escape_character`" property is optional; it is only required if you want to change the default escape character (%) in search patterns for wild-card object privileges. The following example shows how to replace the default (%) character with the backslash character (\) as the escape character in search patterns:

```
"pattern_escape_character": "\",
```

The wildcard characters percentage (%) and underscore (_) can be used in the `<wilcard_pattern_expression>`. The percentage sign (%) wildcard matches zero or more characters. The underscore (_) wildcard matches exactly one character.

To match a percentage sign (%) or underscore (_) with the `LIKE` predicate, an escape character must be placed in front of the wildcard character. Use `ESCAPE <escape_expression>` to specify the escape character you are using, as illustrated in the following examples:

`LIKE 'data_%' ESCAPE '_'` matches the string `'data%'`

`LIKE 'data__%' ESCAPE '_'` (two underscores, followed by a percent sign) matches a string that starts with `'data_'`

name

Each role definition must have a valid name, which is a non-empty string that contains only permitted characters. The `"name"` is used as the name of the run-time role created in the database.

```
"name":"RoleX",
```

The following types of role name are allowed:

- `"MyRole#"`
  Role names that end with the hash character (#), for example, "`MyRole#`" can include "`with-grant-option`" privileges in the role along with references to other roles whose names also end with #.

  > i Note
  >
  > A role name ending with # can only be granted to the object owner of an HDI container ("`<container>#OO`"), for example, using the HDI APIs.

- `MyRole`
  Role names that **do not** end with #, for example, `MyRole`, can only include privileges **without** grant options and references to other roles whose names do not end with #.

global_roles

A role can include references to other global roles, which are roles that are created **without** a schema. A valid entry for a global role is any non-empty string, for example: "`GLOBAL_ROLE_1`" or `GLOBAL_ROLE_2`. Multiple roles are separated by a comma.

Sample Code

```
"global_roles":[
    "GLOBAL_ROLE_1",
    "GLOBAL_ROLE_2"
],
```

schema_roles

A role can include references to other schema-local roles, which are roles created **with** a schema. A valid entry for schema-local roles consists of the following elements:

- `"schema_reference"`
  A valid entry for a `"schema_reference"`, which is a unique identifier within the `.hdbrole` file and can be resolved to the name of a real schema or a logical schema by means of the role configuration file (`.hdbroleconfig`).
- `"names"`
  A non-empty, comma-separated list with the 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 role-configuration file (`.hdbroleconfig`).

```
"schema_roles":[
  {
    "schema_reference":"RoleSchemaRef1",
    "names": ["RoleA", "RoleB"]
  }
]
```

system_privileges

A role can include **system** privileges. The relevant system privileges are specified using a list. If the list is specified, it must not be empty and must contain valid system privileges.

```
"system_privileges":[
    "BACKUP ADMIN",
    "USER ADMIN"
  ],
```

schema_privileges

A role can include **schema** 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.

```
"schema_privileges":[
    {
      "reference":"Ref1",
      "privileges":[ "SELECT" ],
      "privileges_with_grant_option":[ "UPDATE" ]
    }
  ],
```

A valid entry for schema privileges consists of:

- `"reference"`
  A unique identifier (within the specified `.hdbrole` file) that can be resolved to a real schema name in the corresponding role-configuration file (`.hdbroleconfig`)

- `"privileges"`
  A non-empty, comma-separated list of schema privileges that are provided by the role **without** grant option
- `"privileges with grant options"`
  A non-empty, comma-separated list of schema privileges that are provided by the role **with** grant option.

> **i Note**
>
> 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.

For any schema managed by HDI, only the following privileges can be specified: `ALTER`, `CREATE ANY`, `CREATE TEMPORARY TABLE`, `EXECUTE`, `SELECT`, `SELECT CDS METADATA`, `SELECT METADATA`, `ATTACH DEBUGGER`, `INSERT`, `UPDATE`, `DELETE`, `DEBUG`, `UNMASKED`.

object_privileges

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.

```
"object_privileges":[
   {
     "name":"Table1",
     "type":"TABLE",
     "privileges":[ "SELECT" ],
     "privileges_with_grant_option":[ "UPDATE" ]
    }
 ],
```

A valid entry for object privileges consists of the following elements:

- `"name"`
  A valid string for the object name, or a synonym of the real object
- `"type"`
  A valid string describing the object type, or the type of the real object in case of a synonym
- `"privileges"`
  A non-empty, comma-separated list of all relevant and applicable privileges on the specified object, which are to be provided by the role **without** grant option
- `"privileges_with_grant_option"`
  A a non-empty, comma-separated list of relevant and applicable privileges, which are to be provided by the role **with** grant option.

The following object types are supported in the specification of object privileges:

- Basic SQL types
  `INDEX`, `FUNCTION`, `PROCEDURE`, `SYNONYM`, `TABLE`, `TRIGGER`, `VIEW`

With SPS04 you can specify objects with wildcard object identifiers to “include” or “exclude” matching objects for a list of privileges. The wildcard object identifier is used in the same way as the SQL `LIKE` regular expression.

```
"object_privileges":[
   {
     "name":"<wildcard_object_identifier>",
     "type":"TABLE",
     "privileges":[ "SELECT" ],
     "pattern_mode": "include"
   },
   {
     "name":"<wildcard_object_identifier>",
     "type":"TABLE",
     "privileges":[ "SELECT" ],
     "pattern_mode": "exclude"
    }
],
```

- `<wildcard_object_identifier>` is specified as a search pattern for object names in `"name"`, for example, `"TABLE%"` or `"TABLE%_HR"` and requires a `"pattern_mode"` section. The pattern_mode must be set to one of the following values:
  - `"include"`
    The search pattern is used to filter all matching objects at the point of deployment; the privileges defined in the `"privileges"` list are granted to the matched objects.
  - `"exclude"`
    The search pattern is used to filter all matching objects at the point of deployment; the privileges defined in the `"privileges"` list are revoked from the matched objects.
- The `"include"` and `"exclude"` pattern mode can be combined in one role several times with different lists of privileges and object types.
- `<wildcard_object_identifier>` can be used for objects of type `FUNCTION`, `PROCEDURE`, `SEQUENCE`, `TABLE`, and `VIEW`.
- The wild-card characters percentage (%) and underscore (_) can be used in the `<wilcard_pattern_expression>`. The percentage sign (%) wildcard matches zero or more characters. The underscore (_) wildcard matches exactly one character.

  > → Tip
  >
  > The `"%"` character is also the character used to escape wildcard characters.

For any schema managed by HDI, only the following privileges can be specified: `ALTER, CREATE ANY, CREATE TEMPORARY TABLE, EXECUTE, SELECT, SELECT CDS METADATA, SELECT METADATA, ATTACH DEBUGGER, INSERT, UPDATE, DELETE, DEBUG, UNMASKED`.

schema_analytic_privileges

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.

```
"schema_analytic_privileges":[
   {
     "schema_reference":"APSchemaRef1",
     "privileges": ["AP1", "AP2"],
     "privileges_with_grant_option": ["AP4", "AP6"]
   }
 ],
```

A valid entry for schema-local analytic privileges consists of: the following elements:

- `"schema_reference"`
  A string that uniquely identifies the schema within the `.hdbrole` file and can be resolved to a real schema name in the corresponding role-configuration file (`.hdbroleconfig`)
- `"privileges"`
  A non-empty, comma-separated list of relevant analytic privileges created in the referenced schema. These privileges are provided by the role **without** grant option.
- `"privileges_with_grant_option"`
  A non-empty, comma-separated list of relevant analytic privileges, which are to be provided by the role **with** grant option.

i Note

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 role-configuration file (`.hdbroleconfig`).

Role Configuration File

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"
    },
    "LogicalSchemaRef1":{
      "logical_schema":"LogicalSchema1"
    }
  }
}
```

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"`. It is possible to specify references of type `"schema"` and `"logical_schema"`.

i Note

The `"logical_schema"` field has a deployment dependency to a logical schema definition, which contains the schema to be used. For more information about defining a logical schema in a `.hdblogicalschema` file, see *Related Information*.

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": "2.0.40.0"
},
"hdbroleconfig" : {
   "plugin_name" : "com.sap.hana.di.role.config",
   "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]
Logical Schema Definition (.hdblogicalschema) [page 232]
Creating Database Roles in XS Advanced (Developer Guide)

6.24 Search Rule Set (.hdbsearchruleset)

Creates search configurations that can be consumed with a built-in database procedure.

The Search Rule Set plug-in creates search configurations defined in design-time `.hdbsearchruleset` artifacts which can be consumed with the built-in database procedure `SYS.EXECUTE_SEARCH_RULE_SET`. The format of the `.hdbsearchruleset` 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.

ℹ 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 design-time definition of a search rule set for XS advanced and SAP HDI:

Code Syntax

`/SRS.hdbsearchruleset`

```
<?xml version="1.0" encoding="UTF-8"?>
<SearchRuleSet:ruleSet xmlns:SearchRuleSet="http://www.sap.com/ndb/
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 plug-in configuration should look like the following example:

Code Syntax

`.hdiconfig`

```
"hdbsearchruleset" : {
   "plugin_name" : "com.sap.hana.di.searchruleset",
   "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]

6.25 Sequence (.hdbsequence)

Transforms a design-time sequence resource into a sequence database object.

The sequence plug-in transforms a design-time sequence resource (defined in a `.hdbsequence` artifact) into a sequence database object. The file format required for the `.hdbsequence` artifact uses a DDL-style syntax that 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 design-time definition of a sequence for SAP 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 plug-in configuration should look like the following example:

Code Syntax

`.hdiconfig`

```
"hdbsequence" : {
   "plugin_name" : "com.sap.hana.di.sequence",
   "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]

6.26 SQL Views (.hdbview)

Transforms a design-time view resource into an SQL view database object.

The view plug-in transforms a design-time view resource (defined in a `.hdbview` artifact) into an SQL view database object. The file format required for the `.hdbview` artifact uses a DDL-style syntax that is equivalent to the syntax of the corresponding SQLScript command `CREATE VIEW`, without the leading "`CREATE`". For more information about the syntax required when creating database artifacts with SQL DDL in XS advanced, see *Related Information* below.

Example Artifact Code

The following code shows a simple example of a design-time definition of an SQL view:

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 plug-in configuration should look like the following example:

Code Syntax

`.hdiconfig`

```
"hdbview" : {
   "plugin_name" : "com.sap.hana.di.view",
   "plugin_version": "2.0.40.0"
}
```

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>#OO`”) 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

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]
Creating Database Artifacts with SQL DDL in XS Advanced (Developer Guide)

6.27 Statistics (.hdbstatistics)

Transforms a design-time statistics resource into a statistics object on a database table.

The statistics plug-in transforms a design-time statistics resource (defined in a `.hdbstatistics` artifact) into a statistics object on a database table. The file format required for the `hdbstatistics` artifact uses a DDL-style syntax that is equivalent to the syntax of the corresponding SQL command `CREATE STATISTICS`, but without the leading "`CREATE`".

! Restriction

The statistics plug-in only supports named statistics.

Example Artifact Code

The following code shows a simple example of a design-time definition of a statistics file for 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 plug-in configuration should look like the following example:

Code Syntax

`.hdiconfig`

```
"hdbstatistics" : {
   "plugin_name" : "com.sap.hana.di.statistics",
   "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]

6.28 Structured Privilege (.hdbstructuredprivilege)

Transforms a design-time DDL-based structured privilege resource into a structured privilege object.

The structured-privilege plug-in transforms a design-time, DDL-based, structured-privilege resource (defined in a `.hdbstructuredprivilege` artifact) into a structured-privilege object in the database. The file format required for the `.hdbstructuredprivilege` artifact uses a DDL-style syntax that 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. For more information about the syntax required when creating database artifacts with SQL DDL in XS advanced, see *Related Information* below.

Example Artifact Code

The following code shows a simple example of a design-time, structured-privilege definition for 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 plug-in configuration should look like the following example:

Code Syntax

`.hdiconfig`

```
"hdbstructuredprivilege" : {
   "plugin_name" : "com.sap.hana.di.structuredprivilege",
   "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]
Creating Database Artifacts with SQL DDL in XS Advanced (Developer Guide)

6.29 Synonyms (.hdbsynonym and .hdbsynonymconfig)

Transforms a design-time synonym definition into a database synonym object.

Although it is true that design-time resources must be free of explicit schema references in order to enable deployment into multiple containers, the rules requiring the use only schema-local objects is occasionally too restrictive. For example, often it is necessary to be able to access database objects that are not brought into the system via HANA DI; such objects include database objects from an ERP system or replicated objects. These objects can be easily accessed by defining synonyms.

Since, in most cases, the fully qualified names of the referenced database objects are not known at design time (because they depend on deployment decisions), the complete definition of a synonym is split into two design-time files: a declaration (with an optional default configuration), and an explicit configuration of the synonym's target. The explicit configuration can be provided at latest at deployment time, and the explicit configuration overrides the optional default configuration. In this way, an administrator can map object references according to the deployment context.

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`).

i Note

The explicit configuration can be provided at the latest at deployment time and it overrides the optional default configuration. In 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",
"revalidate"
        "object" : "<the target object>"
     }
  },
  "com.sap.hana.example::synonym3" : {
     "target" : {
        "remote"   : "<the remote source>", // not in conjunction with
"database"
        "schema"   : "<my Schema>",
        "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.

If the `"remote"` field of a target description is defined, then a `"database"` cannot be specified. Note that the referenced target `"object"` is **not** regarded as an HDI-managed run-time object, is therefore not considered during dependency management.

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 plug-in configuration should look like the following example:

Code Syntax

`.hdiconfig`

```
"hdbsynonym" : {
   "plugin_name" : "com.sap.hana.di.synonym",
   "plugin_version": "2.0.40.0"
},
"hdbsynonymconfig" : {
   "plugin_name" : "com.sap.hana.di.synonym.config",
   "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]
Logical Schema Definition (.hdblogicalschema) [page 232]

6.30 _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.

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. For this reason, the corresponding plug-in library (`com.sap.hana.di.sysbicsynonym`) is not part of the default libraries configures for a new HDI container. Rather, `com.sap.hana.di.sysbicsynonym` must be explicitly configured by an administrator, as described in the *Maintaining HDI Containers* section of the *SAP HANA Deployment Infrastructure (HDI) Reference* listed in *Related Information* below.

Example Artifact Code

The following code shows a simple example of a design-time _SYS_BIC synonym definition for SAP HDI:

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": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]

6.31 System Versioning Table (.hdbsystemversioning)

Transforms a design-time, system-versioned table that refers to a current and history table into a system-versioned table database object.

The System Versioning Table plug-in converts the columns `validfrom` and `validto` of the current table `GENERATED ALWAYS ROW START` and `GENERATED ALWAYS AS ROW END` columns, respectively. In addition, the plug-in adds a `SYSTEM_TIME` period over the `validfrom` and `validto` columns in order to enforce a `validfrom` <= `validto` constraint. Last of all, the plug-in establishes the system-version link between the current and the history table.

→ Tip

For more information about system-versioned tables, see the *SAP HANA SQL and System Views Reference*, for example, the section describing *system_versioning_configuration* in the chapter *CREATE TABLE Statement (Data Definition)*.

Example Artifact Code

The following code shows a simple example of a system-versioned table for SAP HDI:

Code Syntax

/src/CUSTOMERS.hdbsystemversioning

```
SYSTEM VERSIONING "CUSTOMERS"(validfrom, validto) HISTORY TABLE
"CUSTOMERS_HISTORY" NOT VALIDATED
```

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

```
"hdbsystemversioning" : {
   "plugin_name" : "com.sap.hana.di.systemversioning",
   "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HANA SQL and System Views Reference

6.32 Tables (.hdbtable and .hdbdropcreatetable)

Transforms a design-time table resource into a table database object.

The table plug-ins transform a design-time table resource (defined in `.hdbtable` or `.hdbdropcreatetable` artifacts) into a table database object.

The Table plug-in uses a data migration component which transforms an already deployed version of the table into the new structure of the table. The necessary migration steps for the table structure are determined automatically based on changes to the design-time definition of the table. This transformation always uses a temporary shadow table into which the existing data is copied to match the new structure. This can be very costly for tables that contain a lot of data, where the use of the migration-table (`.hdbmigrationtable`) plug-in should be considered. For smaller tables, for example, configuration data, this cost is usually negligible.

i Note

It is possible to migrate from `.hdbtable` to `.hdbmigrationtable`.

The drop-create-table plug-in 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 required for the `.hdbtable` and `.hdbdropcreatetable` artifacts uses a DDL-style syntax that is equivalent to the SQL syntax in the corresponding SQL command `CREATE TABLE`, without the leading "`CREATE`", as illustrated in the following example. For more information about the syntax required when creating database artifacts with SQL DDL in XS advanced, see *Related Information* below.

i Note

Virtual tables defined in the `.hdbvirtualtable` design-time artifact, which is handled by the virtual table build plug-in. Constraints on foreign keys are defined in the `.hdbconstraint` design-time artifacts, which are handled by the constraint build plug-in. In addition, these plug-ins cannot be used to set up a "system-versioned" table or application-time table. A separate plug-in is available for this purpose.

From SAP HANA 2.0 SPS04, changes to the partitioning of the design-time object are ignored so that administrators can change the partitioning on the run-time object directly without fear of a new deployment resetting the partitioning.

→ Tip

With previous versions of SAP HANA, partitioning of the table should only be specified or changed in the design-time table resource, since changes applied to the run-time table object would be lost each time the table is redeployed.

Example Artifact Code

The following code shows a simple example of a table definition for SAP 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

Table and column comments can be defined in-place using the COMMENT keyword, as illustrated in the following code example:

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

Tables support forward declarations for database-level associations using the `WITH ASSOCIATIONS` clause.

The table plug-ins create an additional intermediate “validate” artifact as part of the deployment. This “validate” artifact inserts itself between the table artifact, the artifacts which are referenced in the `WITH ASSOCIATIONS` clause, and the artifacts that consume the table artifact. The “validate” artifact validates the forward-declared associations and issues a warning in the event that 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 plug-in configuration should look like the following example:

Code Syntax

`.hdiconfig`

```
"hdbtable" : {
    "plugin_name" : "com.sap.hana.di.table",
    "plugin_version": "2.0.40.0"
},
"hdbdropcreatetable" : {
    "plugin_name" : "com.sap.hana.di.dropcreatetable",
    "plugin_version": "2.0.40.0"
}
```

General Limitations

The following restrictions apply to the `.hdbtable` or `.hdbdropcreatetable` plug-ins:

- You must specify the table **type**; permitted table types are `ROW` and `COLUMN`. The drop-create-table plug-ins also support global temporary tables with the types `GLOBAL TEMPORARY ROW` and `GLOBAL TEMPORARY COLUMN`.
- A table cannot reference another table:
  - It is not possible to define a table by using another table (for example, `TABLE A LIKE B`)
  - It is not possible to define a table based on a query (for example, `TABLE A AS SELECT […] FROM B`)
- Text-analysis features such as `CONFIGURATION` and `MINING CONFIGURATION` properties for a table column are not supported. To use text-analysis configurations for text columns, define a separate full-text index
- Flexible tables (for example, using the `WITH SCHEMA FLEXIBILITY` clause) and `HISTORY` tables are not supported.
- Virtual tables are handled by the virtual-table build plug-in (`.hdbvirtualtable`).
- Named constraints are not supported.
  Foreign-key constraints are handled by the constraint build plug-in (`.hdbconstraint`).
- Client-side encryption (`CLIENTSIDE ENCRYPTION`) is not supported.
- Table and column comments can be defined in-place using the `COMMENT` keyword.
- Tables also support forward declarations for database-level associations via the `WITH ASSOCIATIONS` clause. The table plug-ins 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 attempts to validate the forward declared associations and will issue a warning if the association is not valid.

  Note

  The `.hdbtable` and `.hdbdropcreatetable` plug-ins cannot be used to set up either a system-versioned table or an application-time table. A separate plug-in is available for these purposes.

Related Information

Table Data (.hdbtabledata) [page 269]
Table Type (.hdbtabletype) [page 284]
SAP HDI Artifact Types and Build Plug-ins Reference [page 210]
Creating Database Artifacts with SQL DDL in XS Advanced (Developer Guide)

6.33 Table Data (.hdbtabledata)

Insert data from other files (for example, CSV, `.properties`, or `.tags` files) into database tables.

The table data plug-in can be used to insert data defined in other design-time artifacts (for example, `.hdbtabledata` and `.csv` files) into database tables which are managed by SAP HANA DI and are not system-versioned, temporary, or virtual tables. It is possible to import data from collection tables but only with restrictions. It is not recommended to use `.hdbtabledata` artifacts to import test content or demo data if you do not want to use those artifacts in the productive scenario.

⚠ Caution

Since the table data plug-in has ownership of the data it manages, any run-time modification is reverted on the next deployment of the corresponding table data artifacts.

This topic also includes information about how to import “simplified” key-value table data, for example, data for translation that is stored in `.properties` or `.tags` files. For more information, see Simplified Key-Value Table Data (.properties and .tags) [page 278]

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 271]

  > **! 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 272]
  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 275]
  Specifies the target table columns and filters for the data that is inserted into the target table
- `column_mappings` [page 277]
  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.

Source Data Attributes

Attribute	Description	Mandatory
`data_type`	The type of the data file. Supported values are: “CSV” and “PROPERTIES” Note The data type “PROPERTIES” specifies data stored in a `.properties` file format used, which is used in translation scenarios in HDI. Unlike the CSV data type, no configuration options are required for the “PROPERTIES” data type.	Yes
`file_name`	The name of a deployed table data source file, for example `data.csv`; the file has to be deployed via the table data source plug-in and follows the normal run-time naming scheme	Yes
`has_header`	Boolean flag to indicate whether the data file contains a header (default = “`false`”)	No

Attribute	Description	Mandatory
`no_data_import`	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`"). ⚠ Caution 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 *Key-Reservation Import Scenario* below.	No
`delete_existing_foreign_data`	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 `include_filters`); that is, the keys are used to delete foreign entries and no check is performed for existing matching foreign data. ⚠ Caution 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 *Key-Reservation Import Scenario* below.	No
`type_config`	Object to further configure the data parser to match the data file format	No
`dialect`	Specifies the type of data to import. Possible dialects are: `HANA` and `HANA-JSON`. Note that the default configuration can be overwritten by `type_config`. ! Restriction `HANA-JSON` is only for use when importing JSON data into a JSON collection in the SAP HANA Document Store (DocStore).	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:

Configuration Type Attributes for the Data Type CSV

CSV Attribute	Description	Default Value
`line_terminator`	Character to use as line terminator, for example, "\n" or "\r"	auto detect: \n, \r, \r \n

CSV Attribute	Description	Default Value
`delimiter`	Character to use as record delimiter, for example, "," (comma) or ";" (semi-colon) ⚠ Caution `<delimiter><delimiter>` (,,) or (;;) is treated as NULL. `<delimiter><quote_character><quote_character><delimiter>` (,"",) or (;"";) is treated as an empty value. `<delimiter><space><delimiter>` (, ,) or (; ;) is treated as a space	,
`do_quote`	Flag to enable quoting	true
`quote_character`	Character to use as quoting character for records, for example, " (double quote) for records like "value,1"	"
`do_escape`	Flag to enable escaping	false
`escape_character`	Character to use as escaping character, for example, \ (backslash) for escaped records such as "a\nb" (to escape the "n" character)	\
`do_weak_escape`	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
`use_escape_sequences`	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 [page 274]
- Column-Import Scenario [page 275]

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 Column 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.

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" }
    },
    "import_settings" : {
       "import_columns": ["A", "B", "IMPORTED"]
    }
  },
```

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"
    }
  ]
}
```

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",`
  "tableCol12" 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`"

[in|ex]clude_filter

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 export 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.

⚠ Caution

To prevent the accidental or unwanted deletion of data, it is strongly recommended to use the `include_filter` to control the import of data into the target table, for example, by setting filters that specify what information is imported into which column. Without include filters the table-data import operation will always delete existing content and redeploy content from the source CSV file into the target table.

column_mappings

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:

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"

Mapping Type	Description	Example
`<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:

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.
extractLanguageCodeFromFileName	`file_name`	Extracts the language code from the named file using the parameter `file_name` (the original file name without suffix and language code). The following filename formats are supported: • `<file_name>_<languageCode>` • `<file_name>` No specified language code (corresponds to the original language)

Simplified Key-Value Table Data (`.properties` and `.tags`)

The Table Data Properties plug-in can be used to insert data from files in the "`.properties`" file format into database tables which are managed by SAP HANA DI, 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 by means of a special `!tabledata` header inside the file; this ensures that the file is self-contained and simplifies the translation process for language-based `.properties` files.

Sample Code

OBJECT_en_US.properties

```
#!tabledata
#{
# "target_table"    : "TEXT_TABLE",
# "column_mappings" : {
#   "OBJECT"    : { "type" : "constant",
#                   "value" : "name.space::VIEW" },
#   "LANGUAGE" : { "type" : "function",
#                   "name" : "extractLanguageCodeFromFileName",
#                   "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
```

You can use the function `extractLanguageCodeFromFileName` to automate the extraction of the language code information from the name of the `.properties` file. For example, the language code “en_US” would be extracted from the file name `OBJECT_en_US.properties`.

The file encoding for `.properties` files is UTF-8, and a `.properties` file does not support the use of the `\u` escape combination. The translation process for HDI and XS advanced artifacts handles `.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.

The special “`target_table`” value “`#BIMC_DESCRIPTIONS`” must be used when translated texts are deployed for calculation views. This is because the translation text is stored inside the database-internal table `_SYS_BI.BIMC_DESCRIPTIONS`.

To enable support for `multi_key` properties, the JSON tag `"multi_key_separator"` is required in the `!tabledata` section, as illustrated in the following example:

```
"multi_key_separator": "<key_separator>"
```

The `"<key_separator>"` property is used to specify a single character (the so-called key separator) which is used to separate the key into a fixed number of subkeys, according to the columns specified in the `"column_mappings"` section. The `"<key_separator>"` **cannot** be one of the following characters: ‘=’ (equals), ‘:’ (colon), ‘#’ (hash), ‘!’ (exclamation mark), ‘ ‘ (space). Note that you cannot escape the `"<key_separator>"` in a subkey definition, and a subkey that has no value assigned is treated as empty and not as NULL.

→ Tip

The Table Data Properties plug-in can also be used to model `.tags` files.

The following example of a `.properties` file shows the use of multiple keys with the `"multi_key_separator"` property:

Sample Code

`mkp_en_US.properties`

```
#!tabledata
#{
# "target_table"        : "sap::myPropertyTable",
# "multi_key_separator" : ".",
# "column_mappings"     : {
#   "OBJECT"   : { "type" : "constant", "value" : "SKPR07 object" },
#   "LANGUAGE" : { "type" : "function",
#                  "name" : "extractLanguageCodeFromFileName",
#                  "parameters": { "file_name": "mkp" }
#                },
#   "KEY1" : 1,
#   "KEY2" : 2,
#   "KEY3" : 3,
#   "VALUE" : 4
#}
#}
#!tabledata
#
#<metadata>
# <key>.<key2>.<key3>=<value>
# XCOL, 120
.key2_2.key3_3=key1 is missing
# XCOL, 120
key_1.key2_2.key3_3=OK
# XCOL, 120
key_1..key3_3=key2 is missing
# XCOL, 120
key_1.key2_2.=key3 is missing
```

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`

```
"hdbtabledata" : {
  "plugin_name" : "com.sap.hana.di.tabledata",
  "plugin_version": "2.0.40.0"
},
"csv" : {
  "plugin_name" : "com.sap.hana.di.tabledata.source",
  "plugin_version": "2.0.40.0"
}
```

To enable the import of data from `.properties` files, the plug-in configuration should look like the following example:

Code Syntax

`.hdiconfig` for `tabledata.properties`

```
"hdbtabledata" : {
  "plugin_name" : "com.sap.hana.di.tabledata.properties",
  "plugin_version": "2.0.40.0"
},
"csv" : {
  "plugin_name" : "com.sap.hana.di.tabledata.properties",
  "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]
SAP HANA JSON Document Store Guide

6.34 Table Data Properties (.properties)

Insert data from files referenced in the "`.properties`" file format into database tables which are managed by SAP HDI.

The Table Data Properties plug-in can be used to insert data from files listed in the "`.properties`" file format into database tables which are managed by SAP 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 by means of 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 `extractLanguageCodeFromFileName` 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 the `\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`.

To support so-called "multi-key" properties in the `.properties` file, it is possible to use the `"multi_key_separator"` tag in the `!tabledata` section. For example, in `"multi_key_separator": "<key_separator>"`, the value specified in `<key_separator>` defines a single character that is used to separate the key into a fixed number of subkeys according to the columns specified in the `"column_mappings"` section of the `.properties` file. When using the `"multi_key_separator"` property, bear in mind the following restrictions:

- `<key_separator>` cannot be any of the following characters: '=', ':', '#', '!', or ' ' (space)
- It is not permitted to "escape" the `key_separator` in subkeys.
- A subkey that has no value is treated as empty and not as NULL.

i Note

The Table Data Properties plug-in can also be used to model `.tags` files.

Example Artifact Code

The following code shows a simple example of a design-time properties definition for SAP HDI. In this example, `TEXT_TABLE` is a table that was created by an HDI table definition which has at least the columns `"OBJECT"`, `"LANGUAGE"`, `"KEY"`, and `"VALUE"`, and where the primary key consists of at least `"OBJECT"`, `"LANGUAGE"`, and `"KEY"`.

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" : "extractLanguageCodeFromFileName",
#                   "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
```

In the following example, the texts for calculation views are imported into `_SYS_BI.BIMC_DESCRIPTIONS`.

Code Syntax

`/src/BASE_CV_en_US.properties`

```
#!tabledata
#{
#   "target_table"      : "#BIMC_DESCRIPTIONS",
#   "column_mappings"   : {
#     "QUALIFIED_NAME" : { "type"   : "constant", "value" : "BASE_CV" },
#     "LANG"           : { "type"   : "function", "name"   :
"extractLanguageCodeFromFileName", "parameters": { "file_name": "BASE_CV" } },
#     "ID"             : 1,
#     "DESCRIPTION"    : 2
#}
#}
#!tabledata
# #<metadata>
# <ID>=<DESCRIPTION>
# XCOL, 120
K1=K1 Desc
# XCOL, 120
K2=K2 Desc
# XCOL, 120
&&VIEW_NAME&&=BASE_CV Description
# XCOL, 120
K3=K3 Desc
# XCOL, 120
K4=K4 Desc
```

The following example shows how to use the `"multi_key_separator"` tag to define "." (dot) as the character to use to separate the subkeys. In this example, `"sap::myPropertyTable"` is a table created by an SAP HDI table definition that has at least the columns `"OBJECT"`, `"LANGUAGE"`, `"KEY1"`, `"KEY2"`, `"KEY3"`, and `"VALUE"`, where the primary key consists of at least `"OBJECT"`, `"LANGUAGE"`, and `"KEY1"`, `"KEY2"`, `"KEY3"`.

Code Syntax

`/src/mkp_en_US.properties`

```
#!tabledata
#{
#   "target_table"      : "sap::myPropertyTable",
#   "multi_key_separator" : ".",
#   "column_mappings"   : {
#     "OBJECT" :     { "type"   : "constant", "value" : "SKPR07 object" },
#     "LANGUAGE"   : { "type"   : "function",
#                   "name"   : "extractLanguageCodeFromFileName",
#                        "parameters": { "file_name": "mkp" }
#                     },
#     "KEY1"      : 1,
#     "KEY2"      : 2,
#     "KEY3"      : 3,
#     "VALUE"     : 4
#}
#}
#!tabledata
# #<metadata>
# <key>.<key2>.<key3>=<value>
# XCOL, 120
.key2_2.key3_3=key1 is missing
# XCOL, 120
key_1.key2_2.key3_3=OK
# XCOL, 120 "
```

```
key_1..key3_3=key2 is missing
# XCOL, 120
key_1.key2_2.=key3 is missing
```

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

```
"properties" : {
   "plugin_name" : "com.sap.hana.di.tabledata.properties",
   "plugin_version": "2.0.40.0"
},
"tags" : {
   "plugin_name" : "com.sap.hana.di.tabledata.properties",
   "plugin_version": "2.0.40.0"
}
```

Related Information

Table Data (.hdbtabledata) [page 269]

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]

6.35 Table Type (.hdbtabletype)

Transforms a design-time table type resource into a table type database object.

The table-type plug-in transforms a design-time table-type resource (defined in a `.hdbtabletype` artifact) into a table-type database object, for example, for use by signatures of SQL procedures. The file format required for the `.hdbtabletype` artifact uses a DDL-style syntax that 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 design-time, **table type** definition for SAP HDI:

Code Syntax

```
/src/TT_PUBLISHERS.hdbtabletype
```

```
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 plug-in configuration should look like the following example:

Code Syntax

```
.hdiconfig
```

```
"hdbtabletype" : {
   "plugin_name" : "com.sap.hana.di.tabletype",
   "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]

6.36 Text Analysis Configuration (.hdbtextconfig)

Customize the features and options to be used for text analysis.

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 plug-in configuration should look like the following example:

Code Syntax

`.hdiconfig`

```
"hdbtextconfig" : {
   "plugin_name" : "com.sap.hana.di.textconfig",
   "plugin_version": "2.0.40.0"
}
```

Related Information

Text Analysis Dictionaries (.hdbtextdict) [page 286]
Text Analysis Extraction Rules (.hdbtextrule) [page 287]
SAP HDI Artifact Types and Build Plug-ins Reference [page 210]
SAP HANA Text Analysis Developer Guide

6.37 Text Analysis Dictionaries (.hdbtextdict)

Define custom entity types and entity names for use with text analysis.

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 plug-in configuration should look like the following example:

Code Syntax

`.hdiconfig`

```
"hdbtextdict" : {
    "plugin_name" : "com.sap.hana.di.textdictionary",
    "plugin_version": "2.0.40.0"
}
```

Related Information

Text Analysis Extraction Rules (.hdbtextrule) [page 287]

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]

SAP HANA Text Analysis Developer Guide

6.38 Text Analysis Extraction Rules (.hdbtextrule)

Specify rules used to extract complex entities and relationships.

The SAP HANA Text Analysis Extraction Rules artifact (`.hdbtextrule`) is used to define rules and patterns for extracting complex entities and relationships with 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 plug-in configuration should look like the following example:

Code Syntax

`.hdiconfig`

```
"hdbtextrule" : {
    "plugin_name" : "com.sap.hana.di.textrule",
    "plugin_version": "2.0.40.0"
}
```

Related Information

Text Analysis Extraction Rules Includes (.hdbtextinclude) [page 288]

Text Analysis Extraction Rules Lexicon (.hdbtextlexicon) [page 289]

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]
SAP HANA Text Analysis Developer Guide

6.39 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 that be included 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 plug-in configuration should look like the following example:

Code Syntax

`.hdiconfig`

```
"hdbtextinclude" : {
   "plugin_name" : "com.sap.hana.di.textrule.include",
   "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]
SAP HANA Text Analysis Developer Guide

6.40 Text Analysis Extraction Rules Lexicon (.hdbtextlexicon)

Define lists of words for use in sets of text-analysis rules.

The SAP HANA Text Analysis Extraction Rules **Lexicon** artifact (`.hdbtextlexicon`) is used to define lists of words that can be used in one or more top-level sets of text-analysis rules, for example, by using the "`#lexicon`" directive.

→ Tip

For examples showing the content and format of the `.hdbtextlexicon` 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 plug-in configuration should look like the following example:

Code Syntax

`.hdiconfig`

```
"hdbtextlexicon" : {
   "plugin_name" : "com.sap.hana.di.textrule.lexicon",
   "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]
Text Mining Configurations (.hdbtextminingconfig) [page 289]
SAP HANA Text Analysis Developer Guide

6.41 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 plug-in configuration should look like the following example:

Code Syntax

`.hdiconfig`

```
"hdbtextminingconfig" : {
   "plugin_name" : "com.sap.hana.di.textminingconfig",
   "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]
SAP HANA Text Mining Developer Guide

6.42 Triggers (.hdbtrigger)

Transforms a design-time trigger resource into a trigger on a database table.

The trigger plug-in transforms a design-time trigger resource (defined in a `.hdbtrigger` artifact) into a trigger on a database table. The file format required for the `.hdbtrigger` artifact uses a DDL-style syntax that 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 SAP 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 plug-in configuration should look like the following example:

Code Syntax

.hdiconfig

```
"hdbtrigger" : {
   "plugin_name" : "com.sap.hana.di.trigger",
   "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]

6.43 Virtual Functions (.hdbvirtualfunction)

Transform a design-time virtual function resource into a virtual function database object.

The virtual-function plug-in transforms a design-time virtual function resource (defined in `hdbvirtualfunction` and `hdbvirtualfunctionconfig`artifacts) into a virtual-function database object. The target database to which the virtual function points must be available by means of a database remote source.

The file format required for the design-time artifacts uses a DDL-style syntax that 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. Ini 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 SAP 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 plug-in configuration should look like the following example:

Code Syntax

`.hdiconfig`

```
"hdbvirtualfunction" : {
   "plugin_name" : "com.sap.hana.di.virtualfunction",
    "plugin_version": "2.0.40.0"
}
"hdbvirtualfunctionconfig" : {
    "plugin_name" : "com.sap.hana.di.virtualfunction.config",
    "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]

6.44 Virtual Procedures (.hdbvirtualprocedure)

Transform a design-time virtual-procedure resource into a virtual-procedure database object.

The Virtual Procedure plug-in transforms a design-time virtual procedure resource (defined in a `.hdbvirtualprocedure` artifact) into a virtual-procedure database object. The target database to which the virtual procedure points must 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. Iin this way, an administrator can map object references according to the deployment context.

Example Artifact Code

The file format required for the `.hdbvirtualprocedure` artifact uses a DDL-style syntax that 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
```

i Note

The container's object owner (`<container>#OO`) 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 plug-in configuration for the virtual procedure should look like the following example:

Code Syntax

`.hdiconfig`

```
"hdbvirtualprocedure" : {
   "plugin_name" : "com.sap.hana.di.virtualprocedure",
   "plugin_version": "2.0.40.0"
},
"hdbprojectionviewconfig" : {
   "plugin_name" : "com.sap.hana.di.virtualprocedure.config",
   "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]

6.45 Virtual Tables (.hdbvirtualtable)

Transform a design-time virtual table resource into a virtual table database object.

The virtual-table plug-in transforms a design-time virtual-table resource (defined in `.hdbvirtualtable` and `.hdbvirtualtableconfig` artifacts) 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 required for the design-time artifacts uses a DDL-style syntax that 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.

The `AT` part of the `VIRTUAL TABLE` definition defines the default configuration; virtual tables `WITH REMOTE` are not supported.

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`”.

If the remote table is changed, this is not normally detected during deployment. To enable the checking and redeployment of changed remote tables, the optional `MAKE` parameter "`validate_virtual_tables`", or the optional container-configuration parameter "`make.validate_virtual_tables`" can be set to "`true`". The check detects changes to the remote table and redeploys the virtual table. The check also detects deleted virtual tables and raises a warning, but does not redeploy a deleted virtual table. An explicit redeployment of such a virtual table may need to be performed, if needed.

Example Artifact Code

The following code shows a simple example of a virtual-table definition for SAP HDI:

Code Syntax

`/src/remote_customers.hdbvirtualtable`

```
VIRTUAL TABLE "com.sap.hana.example::REMOTE_CUSTOMERS"
AT REMOTE.CUSTOMERS
```

The name of the remote source `REMOTE.CUSTOMERS` specified in the virtual-table definition (`hdbvirtualtable`) is overwritten by the name of the remote source that is defined with the `remote` property in the corresponding virtual-table configuration file (`hdbvirtualtableconfig`), i.e., `REMOTE_SYSTEM_A` in the example illustrated below. For syntax reasons, it is also necessary to provide a remote source with the name of the table object in the definition of the virtual table, even though it will be overwritten. Note that the `database` property in the virtual-table configuration below refers to the name of the remote database (`DATABASE_B`).

Code Syntax

`/cfg/remote_customers.hdbvirtualtableconfig`

```
{
  "com.sap.hana.example::REMOTE_CUSTOMERS" : {
      "target" : {
         "remote"    : "REMOTE_SYSTEM_A",
         "database" : "DATABASE_B",
         "schema"    : "APPLICATON_C",
         "object"    : "CUSTOMERS"
      }
    },
   "com.sap.hana.example::REMOTE_CUSTOMERS_2" : {
      "target" : {
         "logical_schema" : "<The logical schema>",
         "object"          : "CUSTOMERS"
      }
    }
}
```

If the "`logical_schema`" field is defined in a target description, it is not possible to use any of the fields "`database`", "`schema`", or "`remote`". Specifying a "`logical_schema`" creates a deployment dependency to a logical schema defined in the `.hdblogicalschema` artifact.

→ Tip

The logical schema defined in a `.hdblogicalschema` artifact contains the name of the actual target `schema` and the name of the `remote` source. Optionally, a "`database`" can be specified in a logical schema defined in the `.hdblogicalschema` artifact.

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`

```
"hdbvirtualtable" : {
   "plugin_name" : "com.sap.hana.di.virtualtable",
   "plugin_version": "2.0.40.0"
},
"hdbvirtualtableconfig" : {
   "plugin_name" : "com.sap.hana.di.virtualtable.config",
   "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]
Logical Schema Definition (.hdblogicalschema) [page 232]

6.46 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 SAP HDI container (`.hdiconfig`), the plug-in 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": "2.0.40.0"
},
"hdbvirtualpackagesparksql" : {
   "plugin_name" : "com.sap.hana.di.virtualpackage.sparksql",
   "plugin_version": "2.0.40.0"
}
```

Related Information

SAP HDI Artifact Types and Build Plug-ins Reference [page 210]

Important Disclaimer for Features in SAP HANA

For information about the capabilities available for your license and installation scenario, refer to the Feature Scope Description for SAP HANA.

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.
About the icons:

- Links with the icon : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
  - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
  - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon : You are leaving the documentation for that particular SAP product or service and are entering a SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.
The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Gender-Related Language

We try not to use gender-specific word forms and formulations. As appropriate for context and readability, SAP may use masculine word forms to refer to all genders.

www.sap.com/contactsap

THE BEST RUN SAP