Platform, Services, and Utilities

Generated on: 2024-11-08 17:40:10 GMT+0000

SAP Commerce | 2205

PUBLIC

Original content: https://help.sap.com/docs/SAP_COMMERCE/d0224eca81e249cb821f2cdf45a82ace?locale=en-US&state=PRODUCTION&version=2205

Warning

Platform Module

The Platform module in the modules directory consists of optional extensions with additional functionality that is required for certain scenarios.

Features

Media Conversion

Patches

Synchronization Between SAP Commerce Installations

Architecture

Platform Module Architecture

Implementation

AdminAPI

Delta Detection

Handle Media Using the MediaContainer

Using Amazon S3 Media Storage Strategy

Using MongoDB GridFS Media Storage Strategy

Windows Azure Blob Media Storage Strategy

Converting Media with ImageMagick

Patch Concepts and Usage

About the Idap Extension

Single Sign-On for Back-End Applications

Platform Module Features

The Platform module in the modules directory provides a range of features related to carrying out specific scenarios. Use it to expose admin-related features through a REST API, convert media, manage media storage strategies and project data, provide libraries that are required by other features, and authenticate users through SAML Single Sign-On.

Media Conversion

Use the media conversion service to manage bitmap images in your projects.

Patches

Patches enable you to effectively manage data in a project lifecycle. Patches improve the approach to system maintenance and solve the problem of managing project data across different environments.

Synchronization Between SAP Commerce Installations

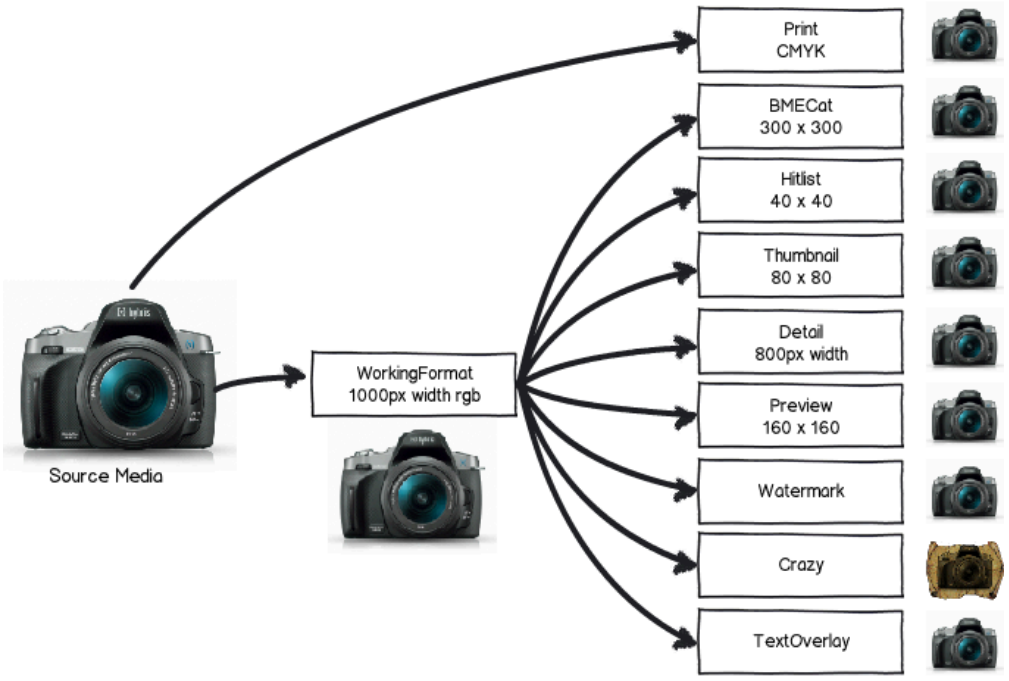
The SAP Commerce-to-SAP Commerce synchronization consists in transferring data from a source system to a target system, for example from one SAP Commerce installation to another, using the Data Hub in between. SAP Commerce-to-SAP Commerce synchronization is possible thanks to the `y2ysync` framework.

Media Conversion

Use the media conversion service to manage bitmap images in your projects.

SAP Commerce allows you to use ImageMagick, an open source software suite, to manage and convert your media.

It's possible to create, edit, compose, or convert almost all available bitmap image formats. It's also possible to extract metadata, such as IPTC and Exif, of a bitmap image.



When applied to a project, media containers can be used as a business representation of one digital asset that contains all the required media derivatives and one source media. It allows SAP Commerce to become a multichannel content distribution platform delivering digital assets to any B2B or B2C touch point.

The main features of the SAP Commerce media conversion are:

- Conversion formats that define conversion settings
- Conversion groups that can contain different conversion format definitions, for example for different asset types
- Conversion chaining possible (input conversion format)
- Manual and scheduled conversion possible
- Media container aggregating source media and all media derivatives

Patches

Patches enable you to effectively manage data in a project lifecycle. Patches improve the approach to system maintenance and solve the problem of managing project data across different environments.

For more information, see Patch Concepts and Usage in Platform Module Implementation.

The Patches feature consists of three separate extensions:

- the `patches` extension includes the main mechanism
- the `patchesdemo` extension demonstrates how to implement and use Patches

- the `patchesbackoffice` extension contains a Backoffice configuration for Patches

Thanks to Patches, it is now possible to reuse the same data files in different contexts. You can, for example:

- create one ImpEx file and use it to create a configuration for 50 shops
- create one file and import it to both Staged and Online catalogs
- create one language-specific ImpEx and import it for each language for a given context
- accelerate country/store roll-outs by simply running a previously created data within the context of the new country or store

Patches support ImpExes that must be run in a single thread or with legacy mode.

Synchronization Between SAP Commerce Installations

The SAP Commerce-to-SAP Commerce synchronization consists in transferring data from a source system to a target system, for example from one SAP Commerce installation to another, using the Data Hub in between. SAP Commerce-to-SAP Commerce synchronization is possible thanks to the `y2ysync` framework.

⚠ Caution

This page refers to deprecated software. For further details, see Deprecation Status.

The `y2ysync` framework ensures high performance of SAP Commerce while you synchronize your data. Synchronization has been divided into stages that are independent of each other. They include transferring data from the source system to the Data Hub, and from the Data Hub to the target system. The `y2ysync` framework guarantees that the process of transferring data from the source system to the Data Hub doesn't affect the target system's performance. Similarly, transferring data from the Data Hub to the target system doesn't affect the source system's or other target systems' performance.

The `y2ysync` data flow architecture ensures high performance and supports scalability of SAP Commerce.

y2ysync Framework

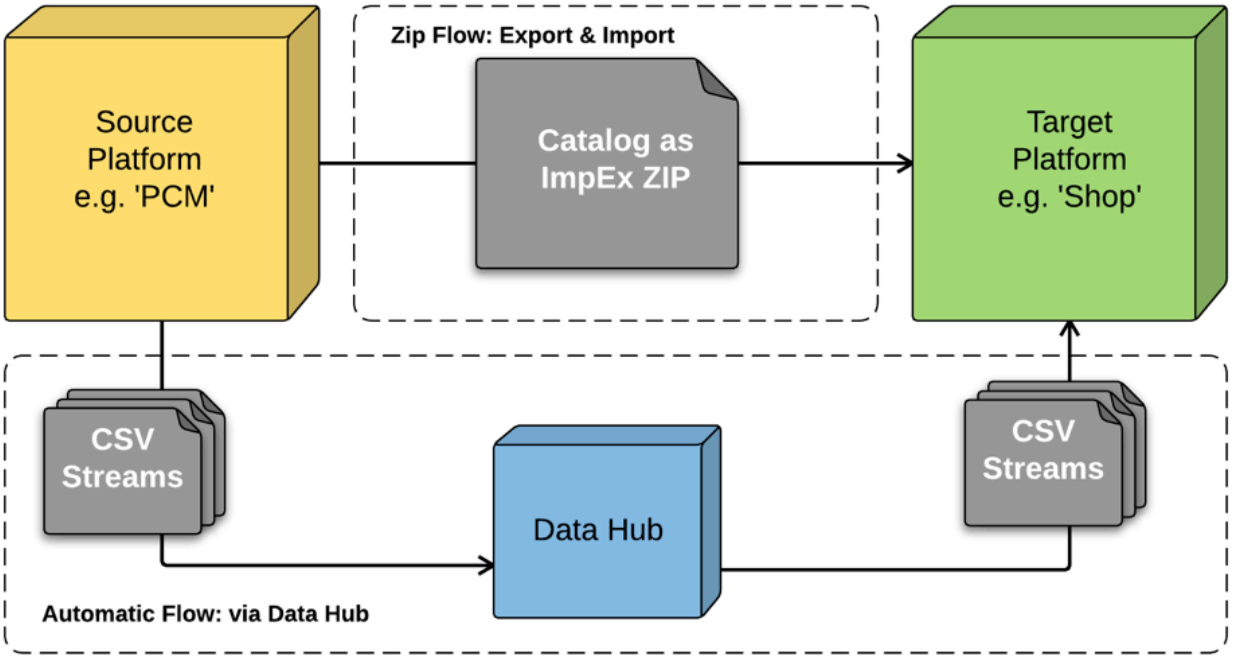
The `y2ysync` framework shipped with Platform allows you to develop your own data synchronization solutions.

⚠ Caution

This page refers to deprecated software. For further details, see Deprecation Status.

With the `y2ysync` framework, you can create sophisticated system architectures consisting of multiple SAP Commerce clusters. Imagine, for example, one SAP Commerce cluster serving as a Product Information Management server, and multiple region-specific storefront clusters. The PIM server would only manage and process the product and catalog master data. It would push out required data to a SAP Commerce installation that serves the storefront and collects orders.

The diagram shows a general architecture of `y2ysync` synchronization:



You can extend the `y2ysync` framework to create solutions that integrate data between SAP Commerce and other systems.

Configuration and Synchronization

`y2ysync` synchronization heavily relies on the `y2ysync` and the Data Hub extensions. For that reason, it requires proper configuration, both on the Platform side and the Data Hub side.

Configuration

We've provided extensive configuration options for you. They allow you to specify the range and type of data you want to synchronize. To configure `y2ysync` on the Platform side, use the following items:

Item	Description
`Y2YColumnDefinition`	The most basic configuration item. It represents a single attribute (a column) that you can synchronize. The `product`'s `code` is an example of such an attribute.
`Y2YStreamConfiguration`	Represents a single type that you can synchronize. It contains multiple `Y2YColumnDefinition` items that define attributes you want to synchronize. The `product`, as defined in Platform (contains multiple attributes), is an example of such a type.
`Y2YStreamConfigurationContainer`	Represents a single **unit** of synchronization that contains multiple `Y2YStreamConfiguration` items. When you initiate synchronization, you search for changes in all the items that belong to the types specified in `Y2YStreamConfigurationContainer`. These changes are then saved as **medias** for export.

In addition to preparing configuration on the source Platform side, you have to configure Data Hub with installed `y2ysync-datahub-ext` extension. Such a configuration describes:

- The data model that you feed into Data Hub from Platform

- Transformations the data model should undergo during the composition phase
- The target systems to which you want to push the data.

You can generate your Data Hub configuration using the **Data Hub Configuration Generator**. It generates an **xml** based on `Y2YStreamConfigurationContainer`. Use the same **Data Hub Configuration Generator** to upload your configuration to the running Data Hub.

For more information, see Generating and Uploading Data Hub Configuration XML.

Synchronization

The synchronization process usually involves three parties:

- The **source system** detects items that you created, changed, or deleted since the last synchronization. It creates files that contain information about the changes - medias. It finally sends a request to Data Hub to start executing its part of the synchronization process.
- Data Hub acts as a middleman between the source and the target systems. Data Hub imports data from the source system, composes it and publishes to the target system. Depending on configuration, you can trigger these steps automatically or manually. For more information, see Target System Definition and Auto Passthrough.
- The **target system** receives the new or updated data via impex

You can initiate synchronization programmatically using the `startSync` method from `SyncExecutionService`, or by clicking the **Perform Y2YSyncJob Action** button available in Backoffice. To initiate synchronization, an instance of `Y2YSyncJob`with a unique code is required. It must be also assigned to a specific `Y2YStreamConfigurationContainer`. As a result of initiation, `Y2YSyncCronJob` is created. Each `Y2YSyncCronJob` represents a single execution of a synchronization action at a given time. It also holds `SyncImpExMedia` objects that contain the data that have changed since the last synchronization.

You can make Data Hub import the `y2ysync` media files again if the previous attempt failed. To do it, click the **Resend to datahub** action button available in Backoffice.

The relation between `Y2YSyncJob` and `Y2YStreamConfigurationContainer` is somehow similar to `Factory` and objects that it produces. The second execution of synchronization would create another `Y2YSyncCronJob` object. The result doesn't overwrite the previous `Y2YSyncCronJob`. Thanks to such a design approach, you can keep the history of synchronization attempts.

Data Hub Processing Details

When all `y2ysync` medias have been created, the source Platform calls the Data Hub endpoint. It is exposed by the `y2ysync-datahub-ext` extension at the address `/y2ysync/v60`. The request from Platform to Data Hub to start executing its part of the synchronization process contains the following information:

- the synchronization execution id
- the source Platform url
- links to all medias created in the change detection process
- a pool and a feed
- the auto-publish target systems.

`y2ysync` uses a feed and a pool just as they are specified in `Y2YStreamConfigurationContainer`. If they don't exist in Data Hub, they get created.

The next step of the synchronization process depends on whether you have specified the target system earlier or not. If not, then `y2ysync` assumes the manual mode for composition and publication. In this case, you must trigger composition and publication manually in Data Hub. However, if you have specified your target system in `Y2YStreamConfigurationContainer` in the source Platform, the target system implicitly enables the **automatic passthrough** mode. In the automatic passthrough mode, composition and publication are triggered automatically. For more information, see Target System Definition and Auto Passthrough.

> ⚠ **Caution**
>
> When using the automatic passthrough mode, before the first synchronization make sure that you have set your target system on the `Y2YStreamConfigurationContainer` object. The first synchronization request creates the necessary feed and pool, and they must have a specific publication strategy set with the target system names. Therefore, if the target system was invalid or empty, automatic passthrough won't take place.

Change Consumption Modes

The process of marking items as synchronized (consumed) is called **change consumption**. There are two change consumption modes that you can use to synchronize data, the synchronous mode and the asynchronous mode. The asynchronous mode may contribute to improving the overall performance of the synchronization process.

Data Hub uses REST to notify the source Platform that it has completed downloading all medias with changed items created in the change detection process. The endpoint responsible for receiving this notification call is exposed by `de.hybris.y2ysync.controller.ConsumeChangesController` at url `/changes/`.

In the **synchronous mode** Data Hub doesn't proceed with data composition or publication until all items are consumed. The notification call to the change consumption endpoint is blocking, and the response is rendered only after all items are consumed. Such behavior may not be desired as it could slow down the whole synchronization process.

The **asynchronous mode** uses the task engine and returns the acknowledgment immediately. It allows Data Hub to resume processing data in no time. As a result the composition and the publication in Data Hub take place simultaneously with change consumption in the source Platform. The asynchronous processing logic is defined by the `consumeY2YChangesTaskRunner` Spring bean.

Set the `y2ysync.controller.consumption.response.async` property to **true** to enable the asynchronous mode, or to **false** to process items synchronously:

```
# Asynchronous change consumption
y2ysync.controller.consumption.response.async=true

# Synchronous change consumption
y2ysync.controller.consumption.response.async=false
```

The change consumption process works in batches and by default uses multiple threads. You can configure it with the following properties:

- `deltadetection.changeconsumption.batch` - number of ItemVersionMarkers processed in a single batch
- `deltadetection.changeconsumption.threads` - number of threads that will be used to consume changes

```
# number of item version markers that will be processed in one go of the change consumption proces
deltadetection.changeconsumption.batch=200
```

```
# number of threads that will be used to consume delta detection changes
deltadetection.changeconsumption.threads=8
```

Generating and Uploading a Data Hub Configuration

You can generate a configuration required for a Data Hub instance that participates in a `y2ysync` synchronization scenario.

Data Hub Configuration

A Data Hub configuration maps `Y2YStreamConfigurationContainer` and its children objects to a Data Hub model, that is to **raw**, **canonical**, and **target** items. Each object type value is taken from `MediaRaw`. Raw, canonical, and target item types are derived from the `dataHubType` property of the `Y2YStreamConfiguration` object, using the following convention:

- `dataHubType` for raw items
- `dataHubType` with the `Canonical` suffix for canonical items
- `dataHubType` with the `Target` suffix for target items

Data Hub Configuration Generator

With the **Data Hub Configuration Generator** you can:

1. Generate an xml configuration based on the `Y2YStreamConfigurationContainer` settings.
2. Save the configuration in the editor area as a `dataHubExtension` property of `Y2YStreamConfigurationContainer`.
3. Save and upload the configuration (extension) to the Data Hub instance, defined with the address specified in the `y2ysync.datahub.url` property.
4. Use the generation options that allow you to specify which parts of the configuration xml to generate.

You must provide a user id, password, and the URL for the Data Hub Adapter on the target Platform.

To access the **Data Hub Configuration Generator**:

1. In the Backoffice explorer tree, click **Y2YSync** > **Export Definitions** .
2. Click `y2ySyncDemoElectronicsToDataHub` to open its editor.
3. Click the **Configure DataHub Action** button. The **Generator** opens.

> ⚠ **Caution**
> In more complicated scenarios, generating a Data Hub configuration xml may require some adjustments. Therefore, you should treat such a generated xml as a base rather than a definite and final configuration. Note, for example, that when you generate the **target** part of Data Hub, `exportURL` and `userName`, password tags are generated with placeholders. Change them before you upload your Data Hub extension.

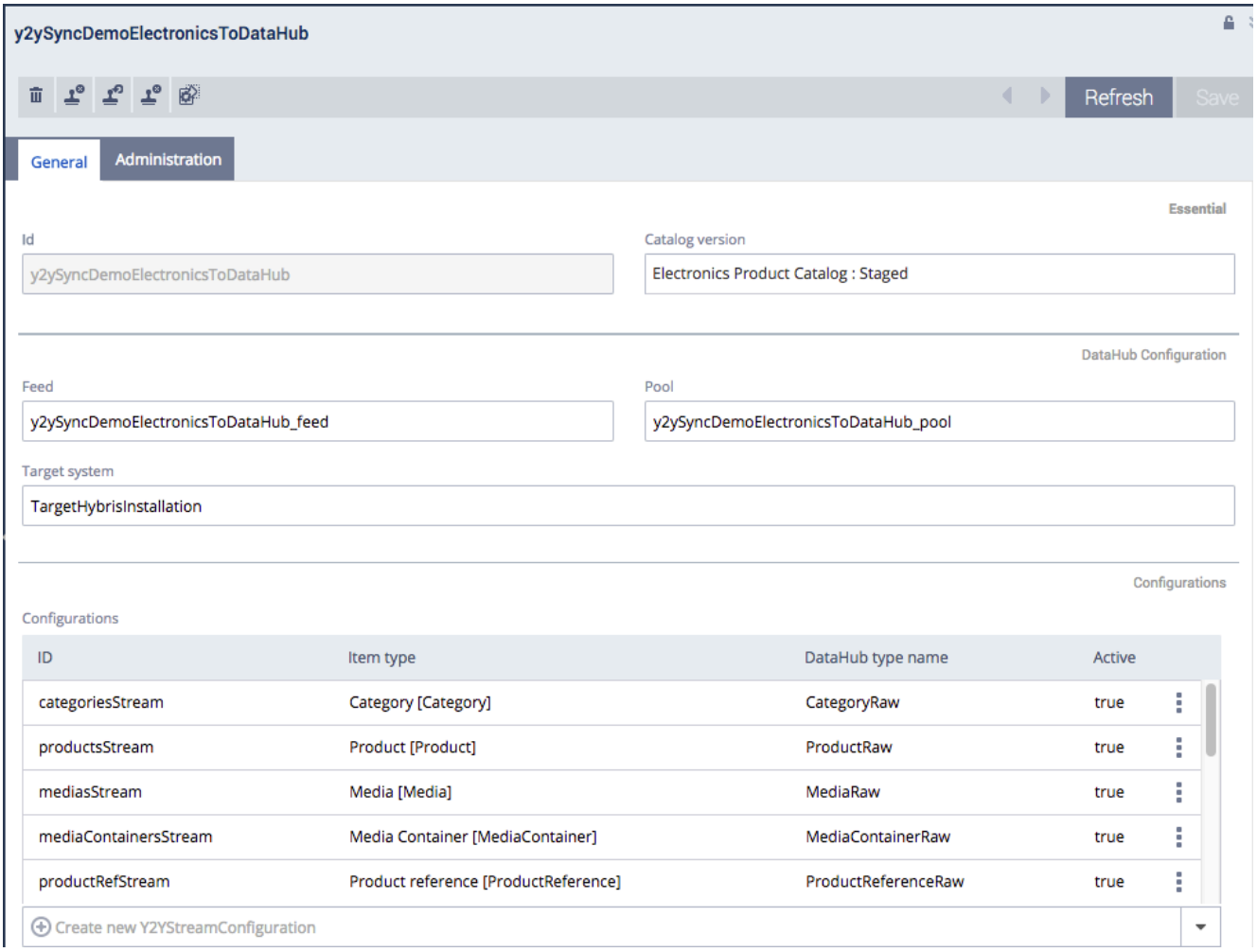
Pool and Feed Definitions

It is possible to name Data Hub feed and pool that are used when synchronizing `Y2YStreamConfigurationContainer`. This feature emulates the **namespace** concept and assures that data in Data Hub from different configuration containers doesn't mix.

Feeds and pools are created by the `y2ysync-datahub-ext` endpoint on the first request. This gives more flexibility. Earlier, the `y2ysync-datahub-ext` extension used single, predefined, and hardcoded feed and pool (`Y2YSYNC_FEED` and `Y2YSYNC_POOL` respectively) for all data integration scenarios.

If the `Feed` and `Pool` names are not specified when creating `Y2YStreamConfigurationContainer`, they are created automatically by appending the `_feed` and `_pool` suffixes to a container `id`.

Here's an example of feed, pool, and target system configuration:



Other y2ysync Features

The `y2ysync` feature has some useful features that can help you manage your synchronizations more conveniently.

Target System Definition and Auto Passthrough

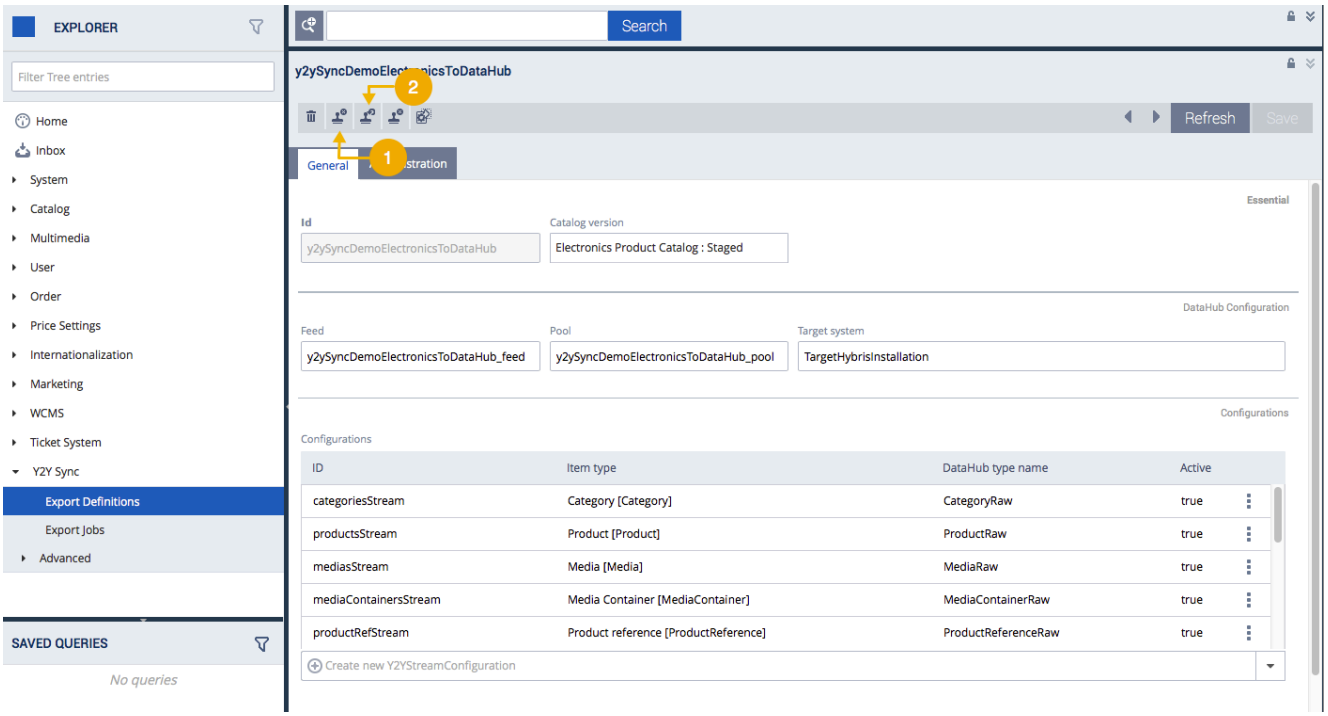
The **auto passthrough** mode triggers, in an automated manner, composition and publication after your data has been imported into Data Hub. To enable auto passthrough, set the `target system` property in a configuration container. Set the same target in the Data Hub configuration xml - the xml must have the same id. If the request Platform sends to initiate synchronization includes information about the target system (id), Data Hub acts in an automated mode and triggers composition automatically. Then, it publishes data to the target system specified with a provided id.

To publish data automatically, set a publication strategy for composition to `y2y:[target system names]` to trigger publication. Therefore, set the target system before the first synchronization - so that a pool with the correct publication strategy name is created. If you don't specify the target system in a container, you'll have to start the composition and publication process manually, like in the previous versions of `y2ysync`.

Resetting and Deleting Item Version Markers

The y2ysync framework uses the deltadetection extension to find out whether you have changed your items, or created new items. Once you have synchronized your items, they are marked as **processed**. In addition, for all of them, **item version markers** are updated, or created. Item version markers provide information about changes. If you don't change anything in your items after the first synchronization, the second invocation of synchronization doesn't do anything - there are no changes to detect and send. However, you can reset, or delete your configuration stream. This enables you to **resend** to the target system exactly the same data you synchronized in the last synchronization.

Delete Stream Action and Reset Stream Action buttons are available in your stream configuration container:



1. Delete Stream Action deletes item version markers
2. Reset Stream Action resets item version markers

Deletion physically removes from the database all item version markers that are related to items sent through a given configuration stream. Reset, on the other hand, reactivates those item version markers that have the **deleted** status. It also sets version timestamps to the beginning of the Epoch time. The result of both those approaches is that data is considered as not synchronized, and you are able to synchronize it again. The only difference is that deletion doesn't trigger the removal action for deleted items, while reset does.

For more actions and wizards, see y2ysync - Available Actions Gallery and y2ysync - Available Wizards Gallery.

y2ysync - Available Actions Gallery

Actions are special **ZK** components that you can add to a widget. They enable you to perform certain y2ysync tasks with only a few clicks. We use actions to provide functionalities for the y2ysync wizards.

These are the standard actions shipped with the y2ysync framework:

Action Name	Description
Configure SAP Commerce Data Hub Action	Generates and then shows a preview of an xml configuration that the user can use to configure the SAP Commerce Data Hub.

Action Name	Description
Resend to SAP Commerce Data Hub Action	Resends into the SAP Commerce Data Hub impex files generated after executing **y2aSyncToDataHub** jobs.
Clone Stream Container Action	Clones a given stream container with the stream configuration assigned to it.
Reset Stream Action	Reactivates **ItemVersionMarkers** that have the **deleted** status.
Delete Stream Action	Deletes all `ItemVersionMarkers` for a selected configuration stream.
Perform Y2YSyncJob Action	Performs `y2ysync` jobs.
Find y2ysync Media Action	Creates a list of all **SyncImpExMedia** media files for a given `y2ysync` cron job.

Configure Data Hub Action

The **Configure Data Hub Action** generates an **xml** configuration for the Data Hub.

Action Definition

Name	Configure Data Hub Action
ID	com.hybris.platform.y2ysync.datahub.config
Description	This action generates and then shows a preview of an xml configuration that a user can use to configure the Data Hub.
Action Class Name	de.hybris.platform.y2ysync.backoffice.actions.CreateDataHubConfigAction
Custom Renderer Class Name	Not applicable.
Inputs	Not applicable.
Outputs	Not applicable.

Action Button Preview

Action Button Location

The **Configure Data Hub Action** button is available in the editor of `y2ysync` stream configuration containers:

1. Log into Backoffice.
2. In the **Explorer tree**, go to:
    - **Y2Y Sync** tab
        - **Export Definitions** tab

          You are in the container **Collection Browser**.

3. In the Collection Browser, click a given configuration container to switch to its editor.

   The button is available in the editor's area.

Configuration

Not applicable.

Dependency

Not applicable.

Resend to Data Hub Action

The **Data Hub Action** resends into the Data Hub impex files generated after executing **y2ySyncToDataHub** jobs.

Action Definition

Name	Resend to Data Hub
ID	com.hybris.platform.y2ysync.datahub.resend
Description	This action enables you to resend into the Data Hub impex files generated after executing **y2ySyncToDataHub** jobs.
Action Class Name	de.hybris.platform.y2ysync.backoffice.actions.ResendToDataHubAction
Custom Renderer Class Name	Not applicable.
Inputs	Not applicable.
Outputs	Not applicable.

Action Button Preview

Action Button Location

The Resend to Data Hub Action button is available in the editor of any **y2ySyncToDataHubJob**-definition cron job:

1. Log into Backoffice.
2. In the Explorer tree, go to:
   - System tab
     - Background Processes tab
       - CronJobs tab

       You are in the cron job Collection Browser.
3. In the Collection Browser, click a given cron job to switch to its editor.

   The button is available in the editor's area.

Configuration

Not applicable.

Dependency

Not applicable.

Clone Stream Container Action

The **Clone Stream Container Action** enables you to clone a given configuration container with the stream configurations assigned to it.

The **Clone Stream Container Action** provides the basic functionality for the `y2yCloneStreamContainer` wizard. For more information about that wizard, see the y2yCloneStreamContainer Wizard topic.

Action Definition

Name	Clone Stream Container Action
ID	com.hybris.cockpitng.action.clonecontainer
Description	The **Clone Stream Container Action** enables you to clone a given stream container with the stream configuration assigned to it.
Action Class Name	de.hybris.platform.y2ysync.backoffice.actions.clone.CloneStreamContainerAction
Custom Renderer Class Name	Not applicable.
Inputs	Not applicable.
Outputs	Not applicable.

Action Button Preview

Action Button Location

The Clone Stream Container Action button is available in the editor of `y2ysync` stream configuration containers:

1. Log into Backoffice.
2. In the Explorer tree, go to:
    - Y2Y Sync tab
        - Export Definitions tab

        You are in the container Collection Browser.
3. In the Collection Browser, click a given container to switch to its editor.

    The button is available in the editor's area.

Configuration

Not applicable.

Dependency

Not applicable.

Reset Stream Action

The **Reset Stream Action** enables you to reactivate those item version markers from a selected configuration stream that have the **deleted** status. It also sets version timestamps to the beginning of the Epoch time.

After you delete an item from a source platform and perform synchronization, the item's `ItemVersionMarker` has the **deleted** status. Performing the Reset Stream Action changes the status into **active**.

Action Definition

Name	Reset Stream Action
ID	com.hybris.platform.y2ysync.resetstream
Description	The **Reset Stream Action** enables you to reset all `ItemVersionMarkers` for a selected configuration stream.
Action Class Name	de.hybris.platform.y2ysync.backoffice.actions.ResetStreamAction
Custom Renderer Class Name	Not applicable.
Inputs	Not applicable.
Outputs	Not applicable.

Action Button Preview

Action Button Location

The Reset Stream Action button is available in the editor of `y2ysync` stream configuration containers:

1. Log into Backoffice.
2. In the Explorer tree, go to:
    - Y2Y Sync tab
        - Export Definitions tab

        You are in the container Collection Browser.
3. In the Collection Browser, click a given container to switch to its editor.

    The button is available in the editor's area.

Configuration

Not applicable.

Dependency

Not applicable.

Delete Stream Action

The **Delete Stream Action** enables you to delete from the Database all `ItemVersionMarkers` for a selected configuration stream (including those that mark items as deleted).

After you perform synchronization on a given item, that item is in the **up-to-date** state. It means that there exists an `ItemVersionMarker` that holds reference to that item, and it has the same `versionTS` as the last **item modified time**. After you delete the `ItemVersionMarker`, during next synchronization the item is considered as one that hasn't been synchronized before and is then synchronized. A new `ItemVersionMarker` is generated for that item.

Action Definition

Name	Delete Stream Action
ID	com.hybris.platform.y2ysync.deletestream
Description	The **Delete Stream Action** enables you to delete all `ItemVersionMarkers` for a selected configuration stream.
Action Class Name	de.hybris.platform.y2ysync.backoffice.actions.DeleteStreamAction
Custom Renderer Class Name	Not applicable.
Inputs	Not applicable.
Outputs	Not applicable.

Action Button Preview

Action Button Location

The Delete Stream Action button is available in the editor of `y2ysync` stream configuration containers:

1. Log into Backoffice.
2. In the Explorer tree, go to:
    - Y2Y Sync tab
        - Export Definitions tab

        You are in the container Collection Browser.
3. In the Collection Browser, click a given container to switch to its editor.

The button is available in the editor's area.

Configuration

Not applicable.

Dependency

Not applicable.

Perform Y2YSyncJob Action

The `Perform Y2YSyncJob Action` provides the basic functionality for the `PerformY2YSyncJob` wizard, which enables you to perform **y2ysync** jobs.

For more information on the `PerformY2YSyncJob` wizard, see the PerformY2YSyncJob Wizard topic.

Action Definition

Name	Perform Y2YSyncJob Action
ID	com.hybris.platform.y2ysync.syncjob.perform
Description	The `Perform Y2YSyncJob Action` enables you to perform **Y2YSync** jobs.
Action Class Name	de.hybris.platform.y2ysync.backoffice.actions.PerformY2YSyncJob
Custom Renderer Class Name	Not applicable.
Inputs	Not applicable.
Outputs	Not applicable.

Action Button Preview

Action Button Location

The **Perform Y2YSyncJob Action** button is available in the **Editor Area** of a **y2ysync** job

1. Log into Backoffice
2. In the **Explorer tree**, click:
    - **Y2Y Sync** tab
        - **Export Jobs** tab

          You are in the Y2YSyncJob **Collection Browser** now.
3. Click a code of a **Y2YSync** job to switch to this job's **Editor Area**.

The Editor Area includes the Perform Y2YSyncJob Action button.

Configuration

Not applicable.

Dependency

Not applicable.

Find y2ysync Media Action

The **Find y2ysync Media Action** enables you to create a list of all **SyncImpExMedia** files generated as a result of executing a y2ysync cron job.

Action Definition

Name	Find y2ysync Media Action
ID	com.hybris.platform.y2ysync.findmedias
Description	The **Find y2ysync Media Action** creates a list of all **SyncImpExMedia** media files for a given y2ysync cron job.
Action Class Name	de.hybris.platform.y2ysync.backoffice.actions.FindY2YSyncMediaAction
Custom Renderer Class Name	Not applicable.
Inputs	Not applicable.
Outputs	Not applicable.

Action Button Preview

Action Button Location

ℹ Note

The **Find y2ysync Media Action** can only be performed on an existing y2ysync cron job. For more information about performing y2ysync jobs, see PerformY2YSyncJob Wizard.

The Find y2ysync Media Action button is available in the editor of any y2ysync cron jobs:

1. Log into Backoffice.
2. In the Explorer tree, go to:
    - System tab
        - Background Processes tab

- CronJobs tab

  You are in the cron job Collection Browser.

3. In the Collection Browser, click a given cron job to switch to its editor.

   The button is available in the editor's area.

Configuration

Not applicable.

Dependency

Not applicable.

y2ysync - Available Wizards Gallery

Wizards enable you to perform certain `y2ysync` tasks with a few clicks, for example create or clone stream configuration containers.

The `y2ysync` framework is shipped with the following wizards:

Wizard Name	Description
PerformY2YSyncJob Wizard	Performs `y2ysync` jobs.
y2yCloneStreamContainer Wizard	Clones `y2ysync` stream configuration containers.
y2yStreamConfiguration Wizard	Creates a `y2ysync` stream configuration for an item type.

y2yStreamConfiguration Wizard

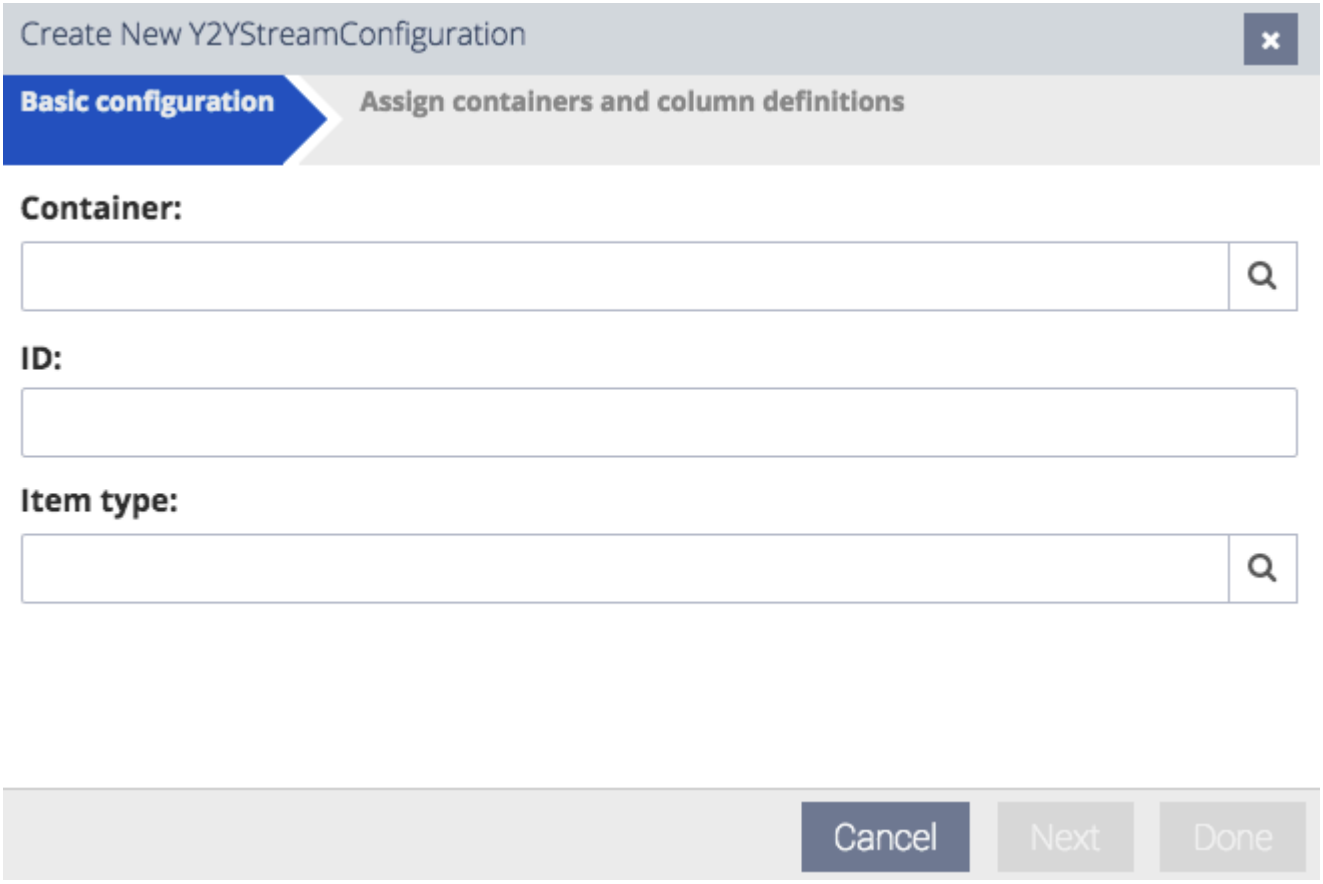
The `Y2YStreamConfiguration` wizard enables you to create a configuration of attributes of a given item type and use this configuration for `y2ysync` synchronization.

Overview

When you create a new stream configuration, the wizard creates column definitions for all attributes defined for a given item type. For example, the wizard creates a column definition for the `Price` attribute for the `Product` item type. You can, however, delete any column definitions that you don't want in your configuration. As a result, you synchronize only chosen column definitions. For example, for the `Product` item, you may decide to synchronize only the `Price` and the `Code` attributes instead of dozens of other column definitions that are by default assigned to this item type.

Wizard Preview

The `Y2YStreamConfiguration` wizard opens in a window.

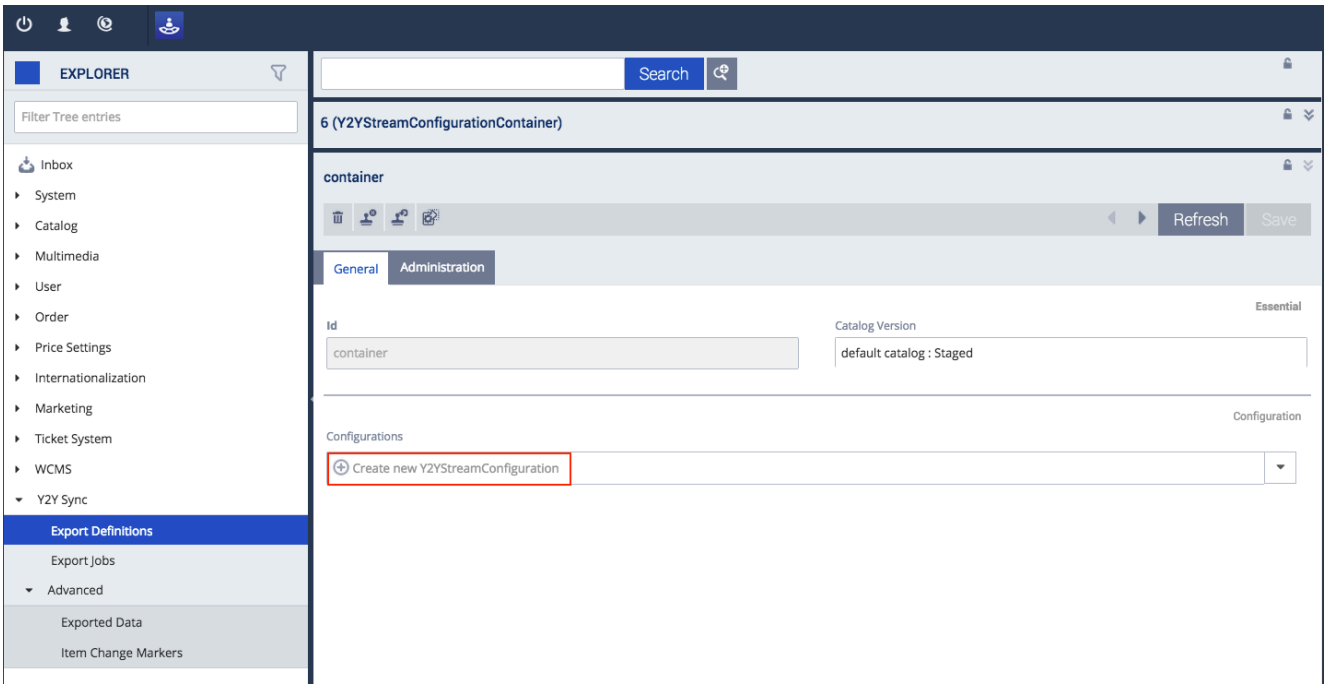


Wizard Location

You can open the `Y2YStreamConfiguration` wizard window from any container's **Editor Area**:

1. Log into Backoffice.
2. In the **Explorer tree**, click:
    - **Y2Y Sync** tab
        - **Export Definitions** tab

        You are in the Y2YStreamConfigurationContainer **Collection Browser**.
3. In the **Collection Browser**, choose any container by clicking its ID to switch to this configuration container's **Editor Area**.
4. In the **Editor Area**, click **Create new Y2YStreamConfiguration**.



The Y2YStreamConfiguration wizard window opens.

Create a Stream Configuration

To create a stream configuration:

1. Open the `Y2YStreamConfiguration` wizard window.
2. Complete the **Container**, **ID**, and **Item type** fields.

   The **Container** field is filled out but you can you can choose other containers.
3. You can complete creating the configuration in two ways:
   - Click **Done** to create a configuration that includes all column definitions defined for the item type you chose, or
   - Click **Next** to be able to delete column definitions that you don't want included in your configuration, and then click **Done**.

You can see your new configuration on the list in the Y2YStreamConfiguration **Collection Browser** after refreshing the view.

Related Information

y2ysync

y2yCloneStreamContainer Wizard

The `y2yCloneStreamContainer` wizard enables you to clone a container and stream configurations assigned to it.

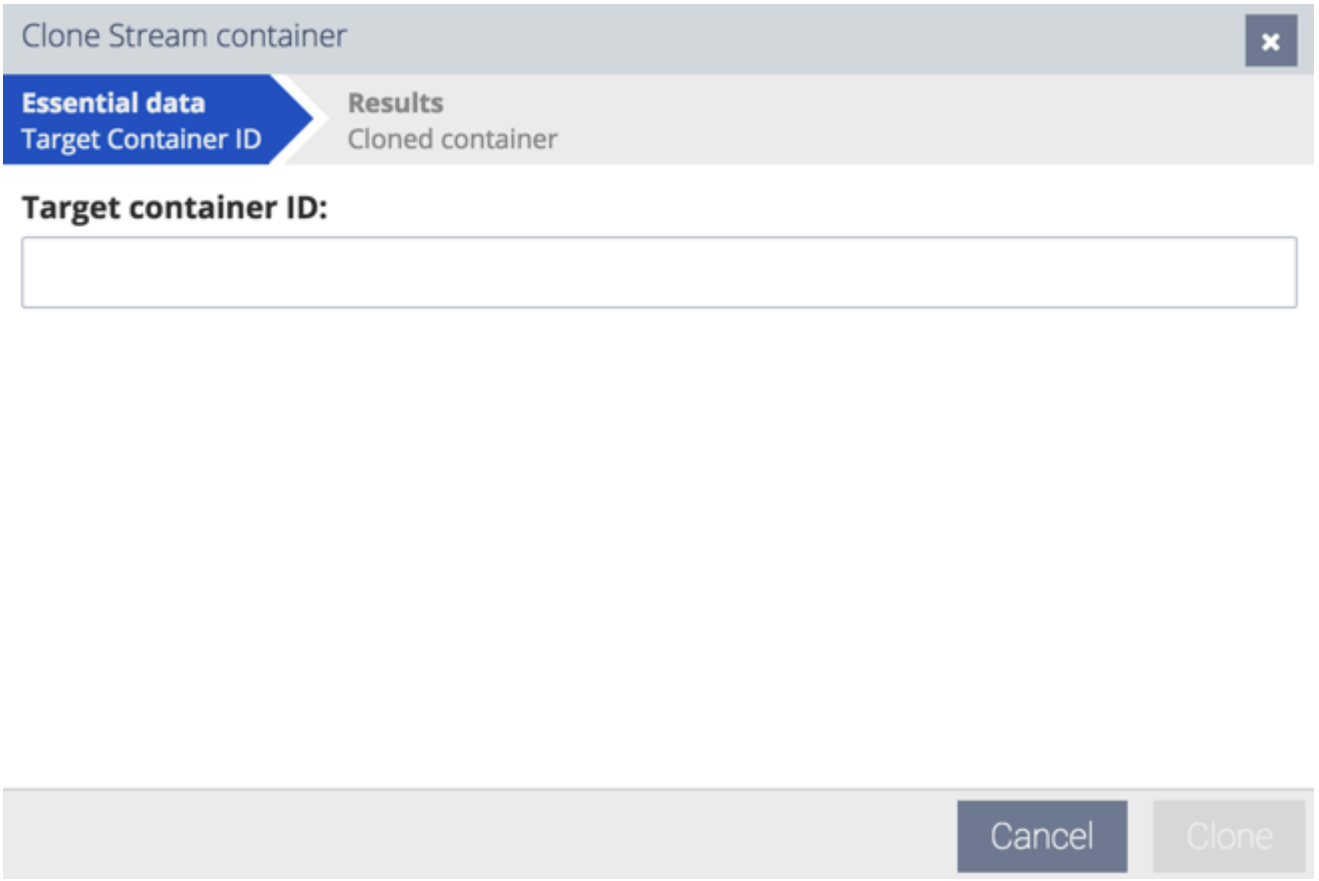
Overview

After you clone a specific container, you can edit the content of the cloned copy according to your needs. Editing a cloned container doesn't affect the original container.

The wizard uses the **Clone Stream Container Action** that provides the cloning functionality. For more information, see the Clone Stream Container Action topic.

Wizard Preview

The `y2yCloneStreamContainer` wizard opens in a window.



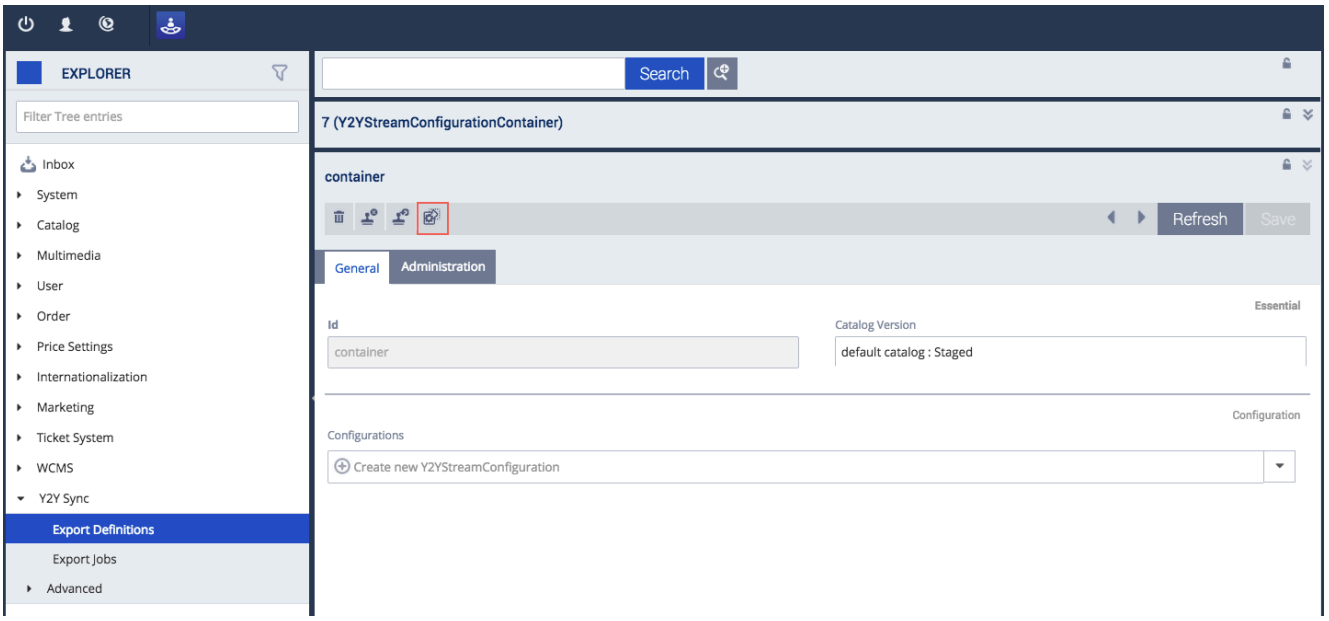
Wizard Location

You can open the `y2yCloneStreamContainer` wizard window from a stream configuration container's **Editor Area**:

1. Log into Backoffice.
2. In the **Explorer tree**, click:
    - **Y2Y Sync** tab
        - **Export Definitions** tab

          You are in the Y2YStreamConfigurationContainer **Collection Browser** now.
3. Click a chosen container ID to switch to this container's **Editor Area**.

   The **Editor Area** includes the **Clone Stream Container** button.



4. Click the **Clone Stream Container** button and confirm you want to clone this container. The `y2yCloneStreamContainer` wizard window opens.

Clone a Container

To clone a container:

1. Go to the Y2YStreamConfigurationContainer **Collection Browser**.
2. Click an ID of a container you want to clone. The window view switches to this container's **Editor Area**,
3. Click the **Clone Stream Container** button.
4. Confirm you want to clone the chosen container.

   The **y2yCloneStreamContainer** wizard window opens.
5. Define an ID for the new container and click **Clone**.

   The window view switches to **Results**.
6. Click **Save** to complete cloning.

   The cloned container's ID should be visible in the Y2YStreamConfigurationContainer **Collection Browser** after refreshing the view.

Related Information

y2ysync

PerformY2YSyncJob Wizard

The `PerformY2YSyncJob` wizard enables you to perform **Y2YSync** jobs.

Overview

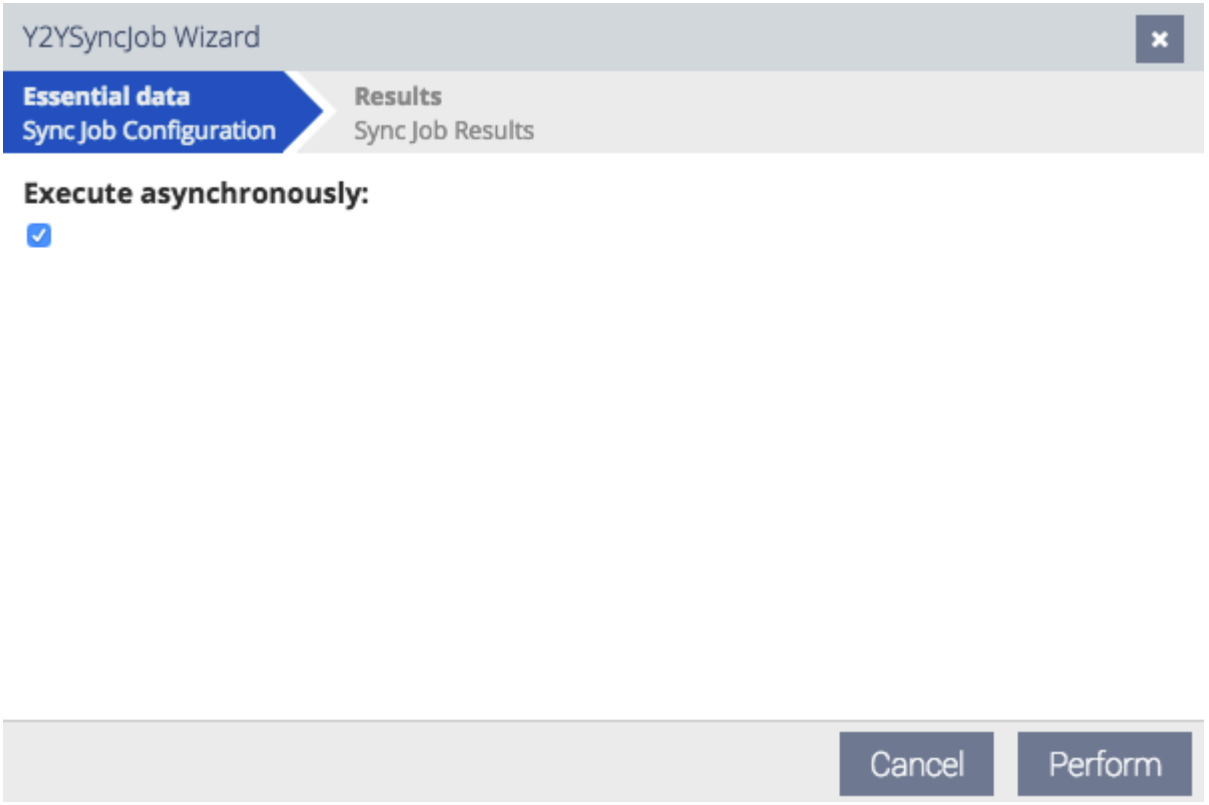
Performing a `y2ysync` job results in creation and execution of a cron job, which in turn results in finding changes in items - based on their version markers - and generating **SyncImpExMedia** files that include data about those changes. When configuring a `y2ysync` job, you can choose the type of `y2ysync` synchronization. It enables you to decide whether to zip the **SyncImpExMedia**

files or leave them in their original form. In the second case, you inform the Data Hub to upload and use these files for `y2ysync` synchronization.

The `PerformY2YSyncJob` wizard uses the `Perform Y2YSyncJob Action` functionality. For more information, see Perform Y2YSyncJob Action.

Wizard Preview

The **PerformY2YSyncJob** wizard opens in a window.



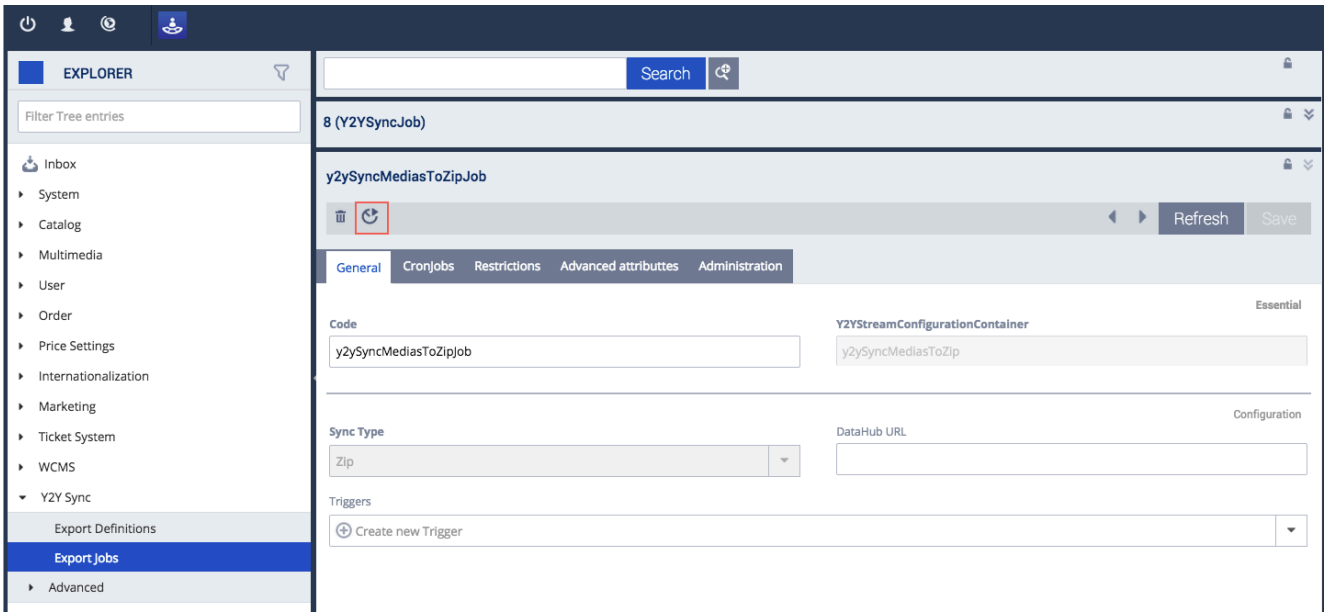
Wizard Location

First, you must choose which `y2ysync` job you want to perform. To open the **PerformY2YSyncJob** wizard for a chosen `y2ysync` job:

1. Log into Backoffice.
2. In the **Explorer tree**, click:
    - **Y2Y Sync** tab
        - **Export Jobs** tab

        You are in the Y2YSyncJob **Collection Browser** now.
3. Click a code of a `y2ysync` job you want to perform to switch to this job's **Editor Area**.

    The **Editor Area** includes the **Perform Y2YSyncJob Action** button.



4. Click the **Perform Y2YSyncJob Action** button.

   The **PerformY2YSyncJob** wizard window opens.

Perform a Job

The **PerformY2YSyncJob** wizard enables you to perform jobs that have been already defined.

To perform a job:

1. Go to a y2ysync **Collection Browser**.
2. Click a code of a y2ysync job you want to perform to switch to this job's **Editor Area**.
3. Click the **Perform Y2YSyncJob Action** button to open the **PerformY2YSyncJob** wizard window.
4. Leave the **Execute asynchronously**: box checked, or uncheck it.
5. Click **Perform**.

   A cron job is generated and executed. The wizard switches to the **Results** view.

6. Click **Done** to close the wizard.

   You have performed your y2ysync job.

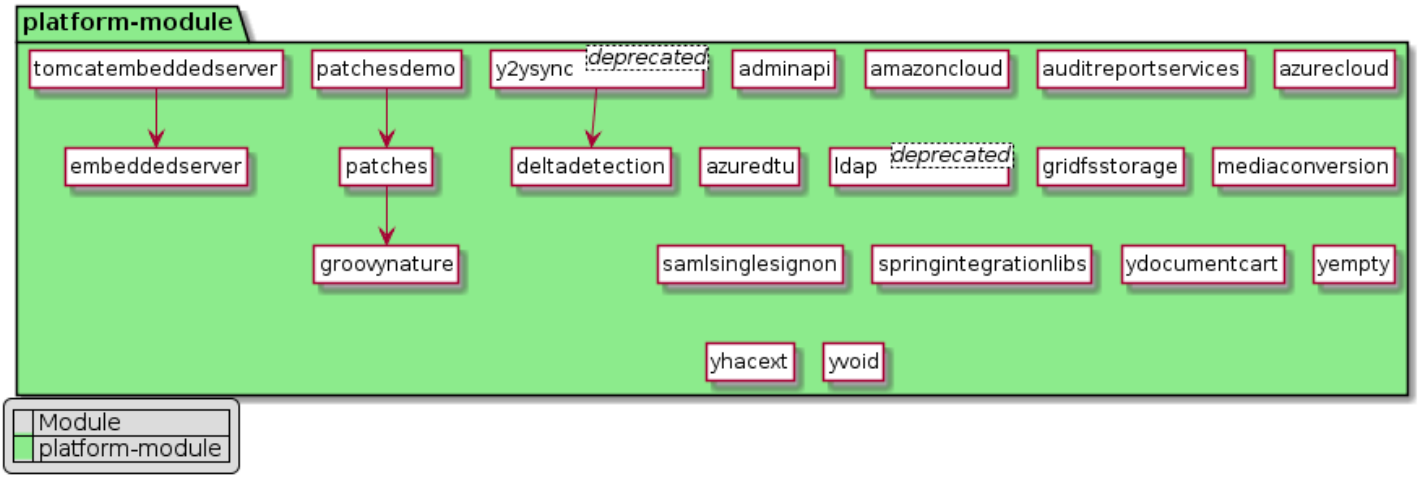
Related Information

y2ysync

Platform Module Architecture

Platform is a set of extensions that provide optional features of SAP Commerce.

Dependencies



For more information, see SAP Commerce Directory Structure.

Recipes

All SAP Commerce recipes include the Platform module in the modules directory.

For a full list of available installation recipes, see Installer Recipes.

Extensions

The Platform module in the modules directory consists of the following extensions:

azuredtu Extension

SAP Commerce Cloud provides an easy way to get the database load of a given Azure database at a given time by using the `azuredtu` extension.

adminapi Extension

The `adminapi` extension allows you to have a highly extensible extension that you can use to expose admin-related functionalities through a REST API.

amazoncloud Extension

The `amazoncloud` extension allows you to use the Amazon S3 media storage strategy.

auditreportservices Extension

The `auditreportservices` extension is responsible for providing audit reports for Backoffice.

azurecloud Extension

The `azurecloud` extension allows you to use the Windows Azure Blob media storage strategy.

deltadetection Extension

The `deltadetection` extension enables the `y2ysync` extension to query for items that have changed since the most recent synchronization.

ldap Extension

The `ldap` extension allows authentification of users and user groups via the LDAP protocol and the import and/or synchronization of data which is stored on an LDAP server.

y2ysync Extension

The `y2ysync` extension contains all the logic on the Platform side. It enables you to synchronize data between the source and target platforms using the Data Hub.

embeddedserver Extension

The `embeddedserver` extension provides an API to run an embedded servlet container.

gridfsstorage Extension

The `gridfsstorage` extension allows you to use the MongoDB GridFS media storage strategy.

groovynature Extension

The `groovynature` extension contains all necessary libraries and tools to use Groovy as the programming language for your extension.

[mediaconversion Extension]

The `mediaconversion` extension provides a framework to convert media of one format to media of another format utilizing the Media Container model.

[patches Extension]

Patches enable you to effectively manage data in a project lifecycle. Patches improve the approach to system maintenance and solve the problem of managing project data across different environments.

[patchesdemo Extension]

The `patchesdemo` extension demonstrates how to implement and use Patches.

[samlsinglesignon Extension]

The `samlsinglesignon` extension allows you to connect SAP Commerce with external identity provider (IDP) systems allowing for a cross-domain single sign-on functionality.

[springintegrationlibs Extension]

The `springintegrationlibs` extension provides additional open source libraries for IDoc processing.

[tomcatembeddedserver Extension]

The `embeddedserver` extension provides an API to run an embedded servlet container. The `tomcatembeddedserver` extension provides a Tomcat-based implementation of this API.

[ydocumentcart Extension Template]

The `ydocumentcart` extension is a template extension that allows you to store selected types in an alternative storage. It uses the polyglot persistence query language.

[yempty Extension Template]

The `yempty` extension template is a predefined extension to be duplicated. The copy serves as starting point for creating a new extension, typically used for customer specific implementations.

[yhacext Extension Template]

The `yhacext` extension allows you to add your own functionalities to SAP Commerce Administration Console.

[yvoid Extension Template]

The `yvoid` extension template is a predefined extension to be duplicated. The copy serves as starting point for creating a new extension, typically used for customer specific implementations. It is a simpler alternative for the `yempty` extension template.

azuredtu Extension

SAP Commerce Cloud provides an easy way to get the database load of a given Azure database at a given time by using the `azuredtu` extension.

ℹ Note

The `azuredtu` extension can be used only with the Azure SQL database.

The extension uses the `DatabaseUtilizationService` interface:

```
public interface DatabaseUtilizationService
{
    List<DatabaseUtilization> getUtilization(Duration duration);
}
```

The interface provides two implementations:

- `AzureDatabaseUtilizationService`: queries the Azure database for current database load
- `BufferedDatabaseUtilizationService`: wraps `AzureDatabaseUtilizationService` and buffers its results

You can access the default implementation of the extension through the `databaseUtilizationService` Spring alias.

Configuration

The `azuredtu` extension allows you to configure the following properties:

- `database.utilization.query.interval.seconds=`: specifies the intervals at which the `BufferedDatabaseUtilizationService` service reads the database load
- `database.utilization.buffer.size.seconds=`: specifies how long values are buffered when the service call doesn't read the database load value

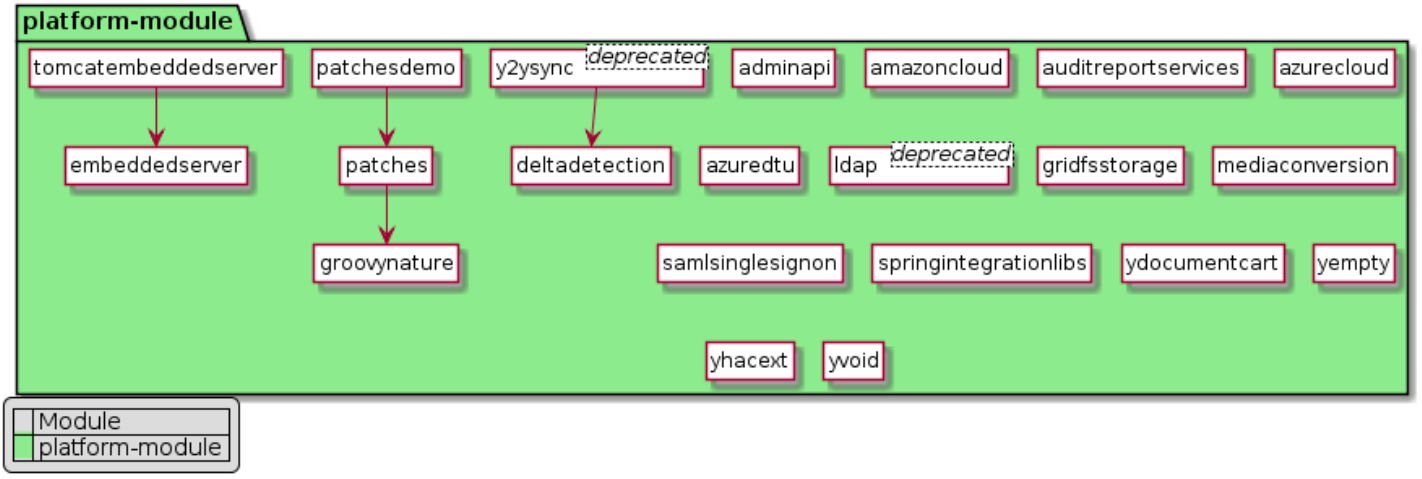
adminapi Extension

The `adminapi` extension allows you to have a highly extensible extension that you can use to expose admin-related functionalities through a REST API.

About the Extension

Name	Directory	Related Module
`adminapi`	`hybris/bin/modules/platform`	Platform Module Architecture

Dependencies



Dependencies

Related Information

AdminAPI

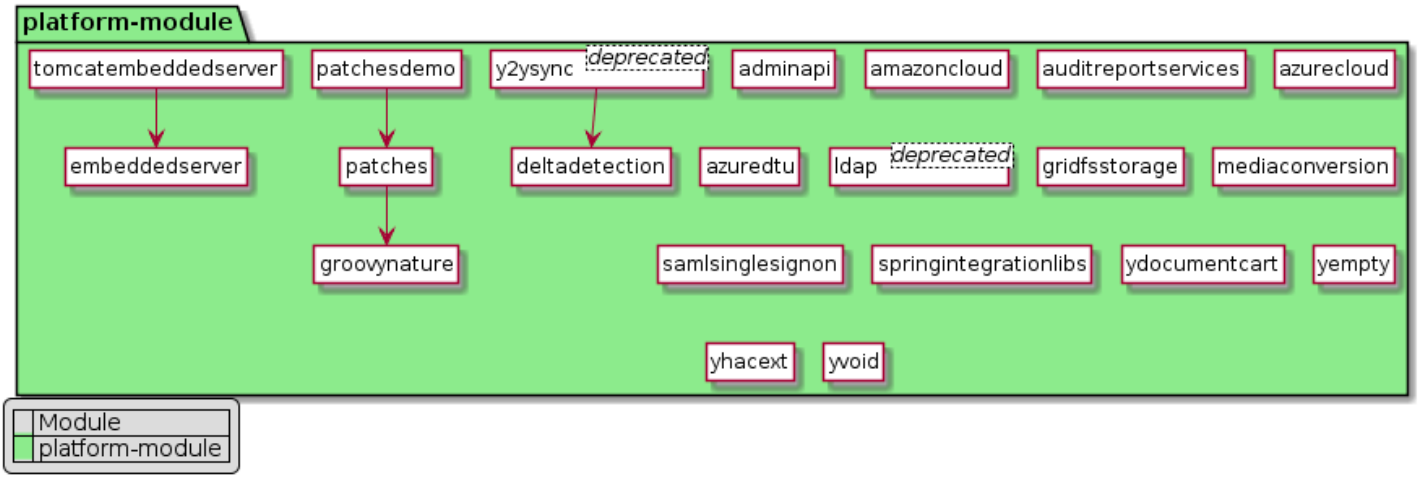
amazoncloud Extension

The `amazoncloud` extension allows you to use the Amazon S3 media storage strategy.

About the Extension

Name	Directory	Related Module
`amazoncloud`	`hybris/bin/modules/platform`	Platform Module Architecture

Dependencies



Dependencies

Related Information

Using Amazon S3 Media Storage Strategy

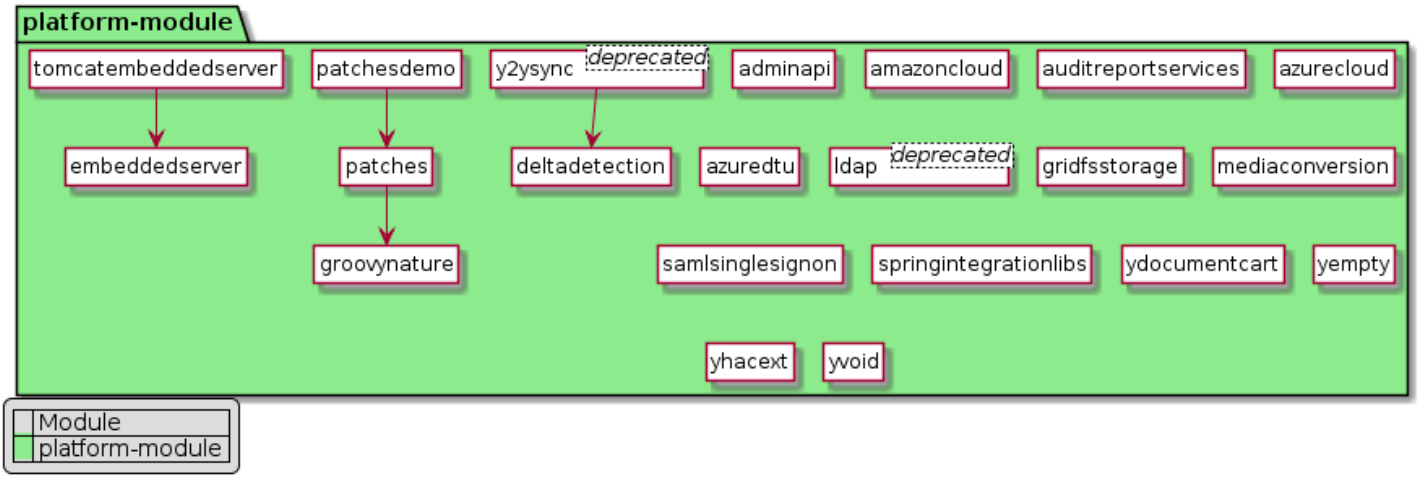
auditreportservices Extension

The `auditreportservices` extension is responsible for providing audit reports for Backoffice.

About the Extension

Name	Directory	Related Module
`auditreportservices`	`hybris/bin/modules/platform`	Platform Module Architecture

Dependencies



Dependencies

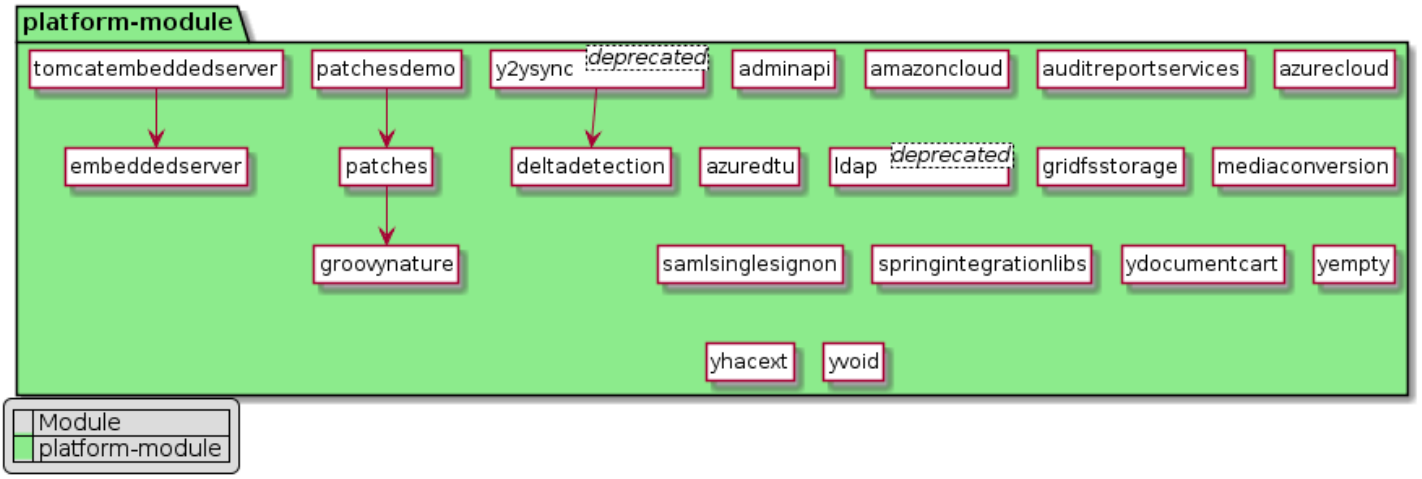
azurecloud Extension

The `azurecloud` extension allows you to use the Windows Azure Blob media storage strategy.

About the Extension

Name	Directory	Related Module
`azurecloud`	`hybris/bin/modules/platform`	Platform Module Architecture

Dependencies



Dependencies

Related Information

Windows Azure Blob Media Storage Strategy

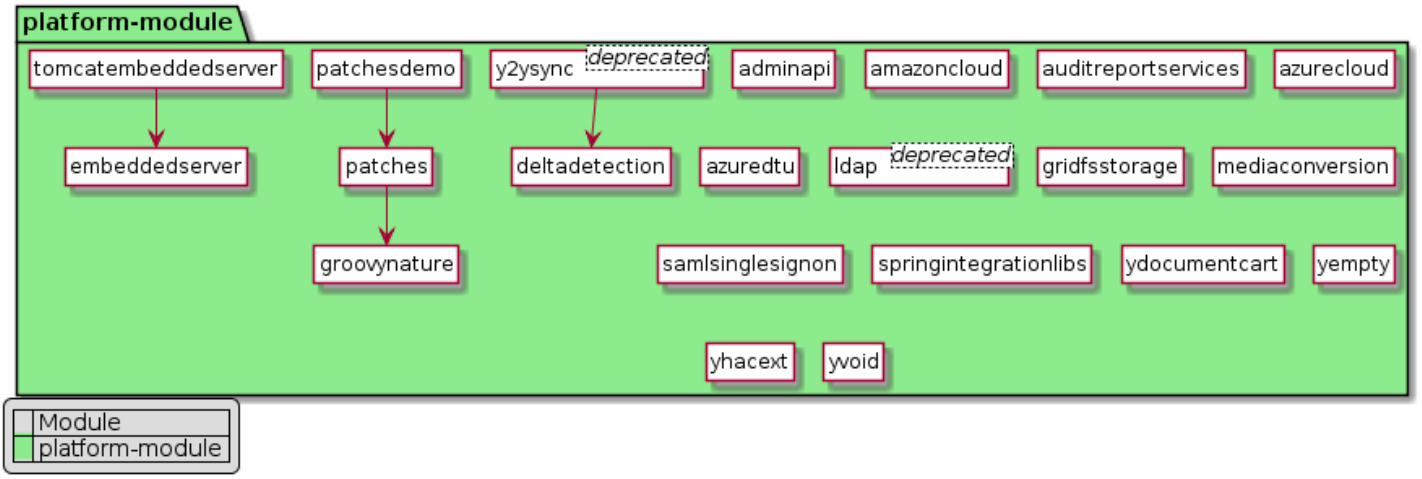
deltadetection Extension

The `deltadetection` extension enables the `y2ysync` extension to query for items that have changed since the most recent synchronization.

About the Extension

Name	Directory	Related Module
`deltadetection`	`hybris/bin/modules/platform`	Platform Module Architecture

Dependencies



Dependencies

Related Information

Delta Detection

ldap Extension

The `ldap` extension allows authentification of users and user groups via the LDAP protocol and the import and/or synchronization of data which is stored on an LDAP server.

ℹ Note

An SAP Commerce extension may provide functionality that is licensed through different SAP Commerce modules. Make sure to limit your implementation to the features defined in your contract license. In case of doubt, please contact your sales representative.

Function Overview

⚠ Caution

This page refers to deprecated software. For further details, see Deprecation Status.

The SAP Commerce `ldap` extension allows SAP Commerce to verify the identity of user accounts against an LDAP server.

Supported authentication modes are:

- Anonymous
- User and password
- Anonymous, SSL-encrypted
- User and password, SSL-encrypted
- SASL and keystore password, SSL-encrypted
- The `ldap` extension gives you the chance to implement a Single-Sign-On concept.
- ImpEx-based import of `LDIF` files and search results (LDAP query language).

Limitations

- Missing DSML support.
- Missing LDIF export.
- Missing LDAP write access. (LDAP server is the leading system).
- SSL support only tested on OpenLDAP.
- Pooling of SSL connections not implemented, yet.

Related Information

ImpEx

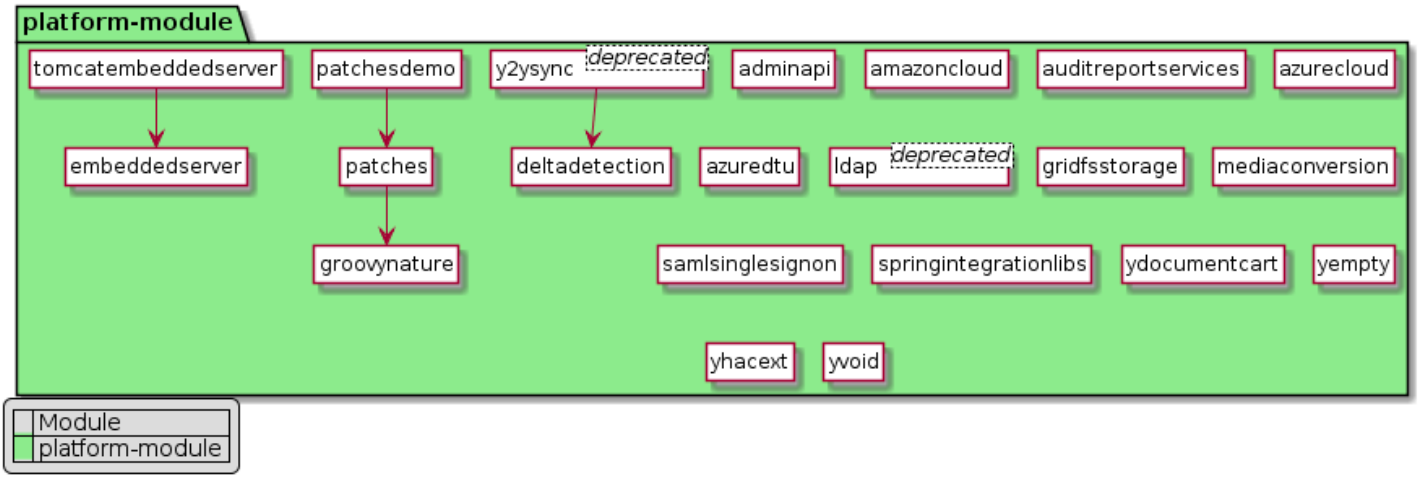
y2ysync Extension

The `y2ysync` extension contains all the logic on the Platform side. It enables you to synchronize data between the source and target platforms using the Data Hub.

About the Extension

Name	Directory	Related Module
`y2ysync`	`hybris/bin/modules/platform`	Platform Module Architecture

Dependencies



Dependencies

Related Information

y2ysync Framework

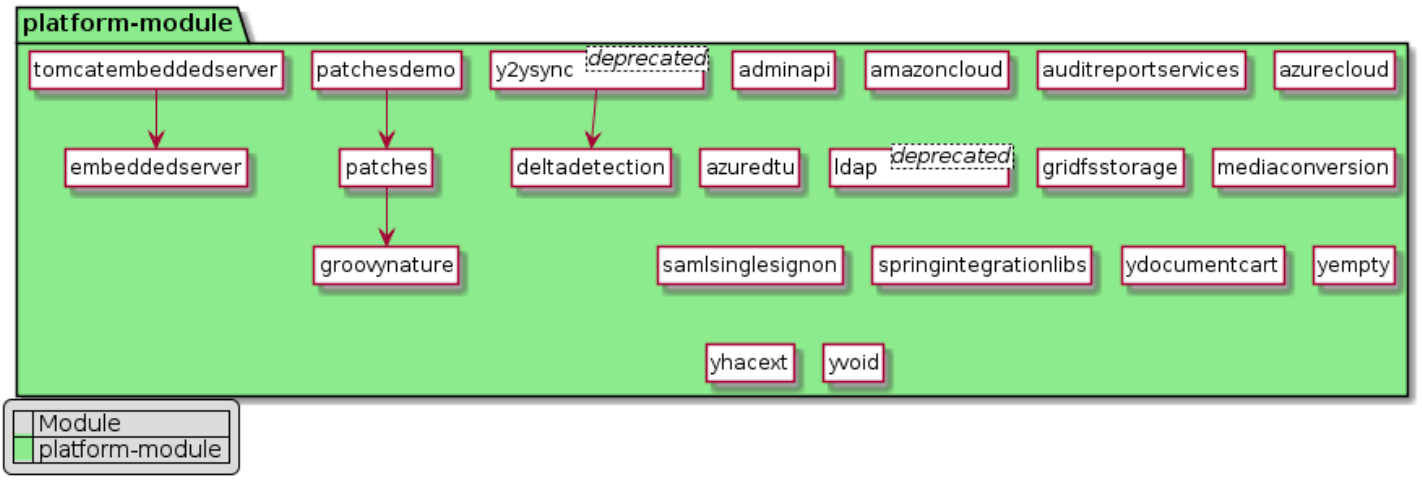
embeddedserver Extension

The `embeddedserver` extension provides an API to run an embedded servlet container.

About the Extension

Name	Directory	Related Module
`embeddedserver`	`hybris/bin/modules/platform`	Platform Module Architecture

Dependencies



Dependencies

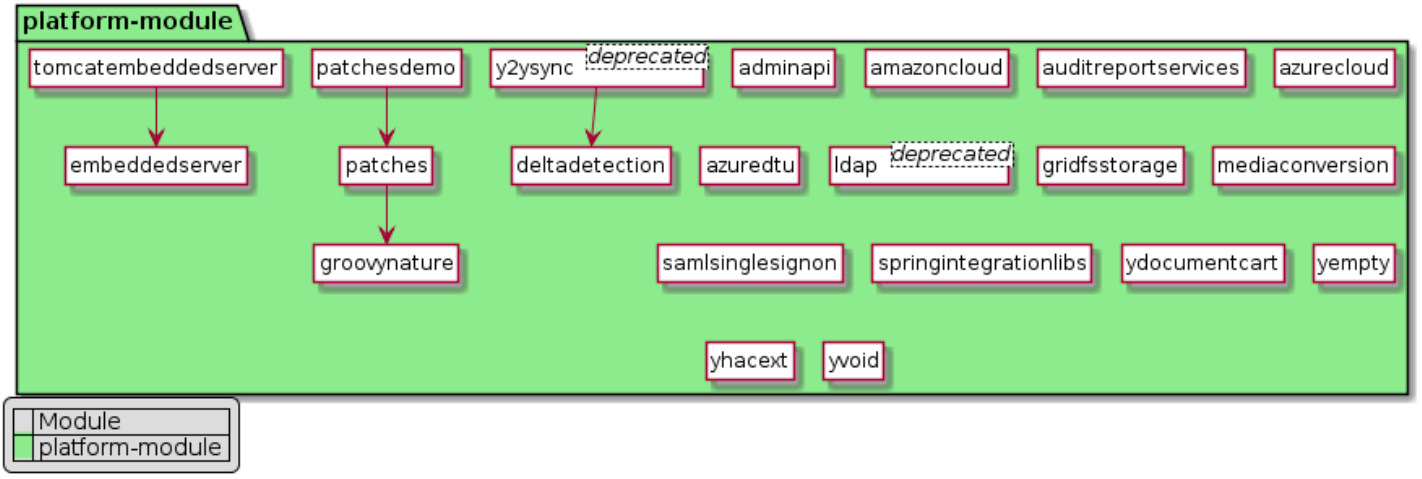
gridfsstorage Extension

The `gridfsstorage` extension allows you to use the MongoDB GridFS media storage strategy.

About the Extension

Name	Directory	Related Module
`gridfsstorage`	`hybris/bin/modules/platform`	Platform Module Architecture

Dependencies



Dependencies

Related Information

Using MongoDB GridFS Media Storage Strategy

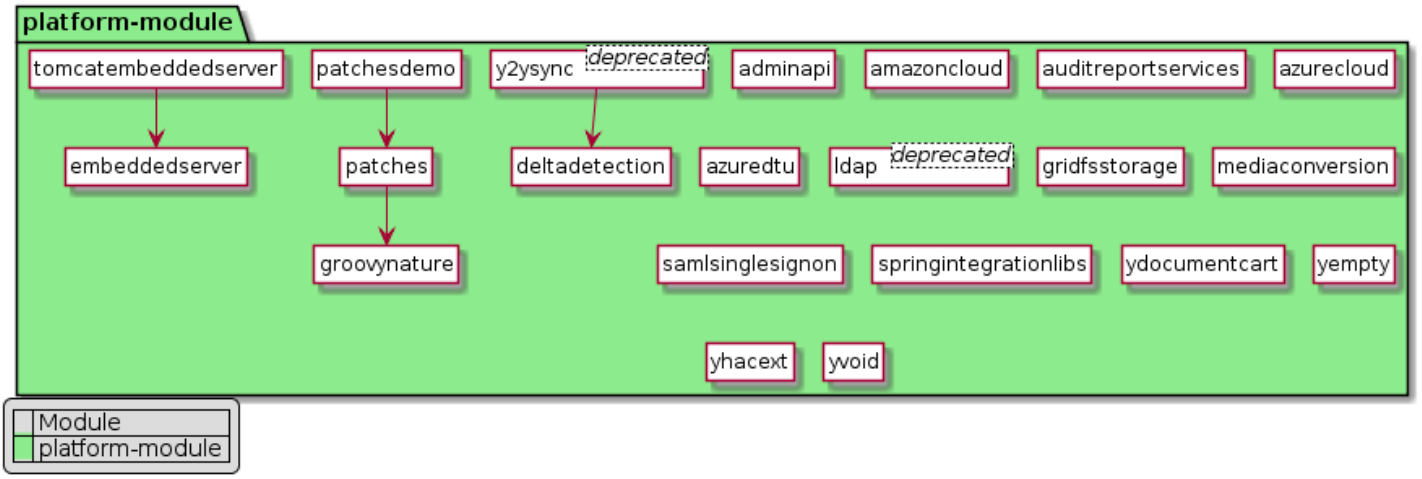
groovynature Extension

The `groovynature` extension contains all necessary libraries and tools to use Groovy as the programming language for your extension.

About the Extension

Name	Directory	Related Module
`groovynature`	`hybris/bin/modules/platform`	Platform Module Architecture

Dependencies



Dependencies

mediaconversion Extension

The `mediaconversion` extension provides a framework to convert media of one format to media of another format utilizing the Media Container model.

To do this, it features a special kind of media format including conversion information and strategy. Furthermore, it hosts an implementation to scale, convert, and modify bitmap image media using the ImageMagick open-source image manipulation toolkit.

ℹ Note

The `mediaconversion` extension is disabled by default. If you want to enable it, you have to add the following entry to the content of <extensions> element in your **`localextensions.xml`** file:

```
<extension dir="${HYBRIS_BIN_DIR}/modules/platform/mediaconversion"/>
```

Then you have to build and initialize your system. Find more information on building in the Installation Based on Specified Extensions document.

ℹ Note

An SAP Commerce extension may provide functionality that is licensed through different SAP Commerce modules. Make sure to limit your implementation to the features defined in your contract license. In case of doubt, please contact your sales

representative.

Media Conversion Overview

SAP Commerce data management is rapidly evolving into a centralized repository for all sorts of data and serves as a backbone for all channel publishing activities. Since each channel has different media asset requirements in terms of quality or resolution, it is mandatory that SAP provides media conversion functionality.

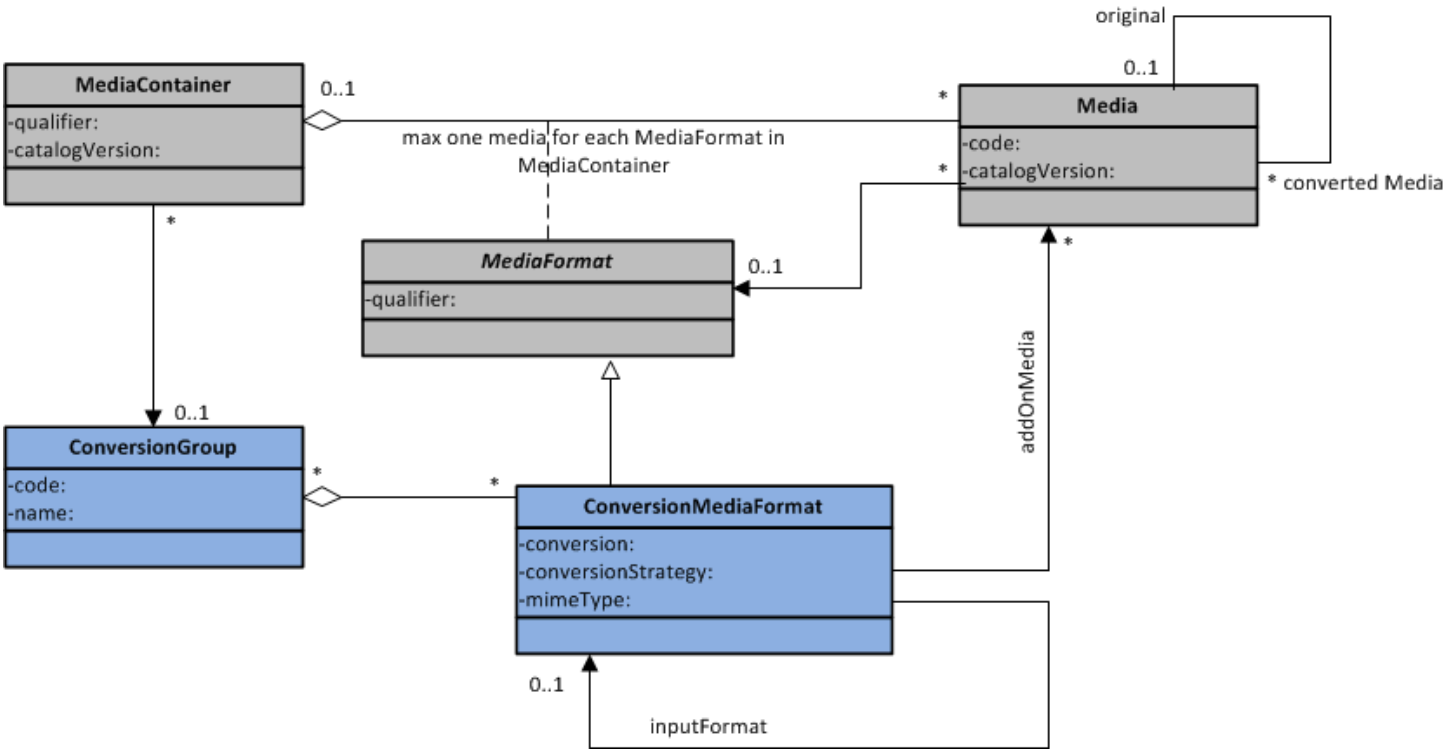
Media Conversion Concept

The idea of a Media Container aggregating multiple representations of one and the same logical media asset already existing in the SAP system. The `mediaconversion` extension adds a framework to automatically generate derived media asset. In other words, to convert an input to a desired output media.

Data Model

To aggregate media of the same kind but in different formats, the MediaContainerModel is leveraged. With this, a MediaContainerModel can be seen as a logical media having several physical representations, for example, medias. Within a MediaContainerModel, there can be one representation for each MediaFormat in the system retaining the original image from which other MediaModel is derived or converted from.

This original media, also referred to as **master**, has a special role. It is identified by the media within the MediaContainerFormat that has neither the original nor the originalDataPK attribute set. These two attributes are manageable information stored by the `mediaconversion` extension. It is used to track the conversion paths and to detect changes and required reconversion. As illustrated, upon conversion the newly created or updated MediaModel gets a reference to its original media set, `original` attribute. The originalDataPK is set representing the exact version of the conversion input. The conversion is also responsible for setting the derived mediaContainer, catalogVersion, and mediaFormat accordingly.



Defining the Media Master

When discussing conversion, it is necessary to understand which media is chosen as the initial source of all conversions. The `mediaconversion` extension mechanism is used to pick the one media within a MediaContainer that does not have the `original` nor the `originalDataPK` attribute set, for example, is NULL. If there is no such media or there are multiple medias matching this condition within a MediaContainer, the master media of this will not be determined and the MediaContainer is treated accordingly as `emtpy` or `not convertible`. These two attributes are considered as administrative data of the `mediaconversion` extension and should not be modified. Also, any modifications on MediaContainer contents of containers containing converted medias is not recommended.

The source media for a particular conversion could be a different media as described in the Conversion Chaining section below.

i Note

Manual Changes

Users should not manually delete converted medias from their media containers without deleting them entirely from the database. All subsequent conversion attempts fail because of unique constraint violations, same generated code, and catalogVersion.

Conversion Media Format

Platform. The ConversionMediaFormat represents a physical format in that context. The MediaFormat instance can still be used to represent logical format which is mapped to a physical one by a MediaContext. The ConversionMediaFormat introduced with the new ConversionMediaFormat extends the MediaFormat and denotes a format that can be converted by the `mediaconversion`.

The SAP Commerce can be shipped with the sample data of ConversionMediaFormat. Find description in the Conversion Media Formats - Sample Data document.

Conversion Command and Strategy

The ConversionMediaFormat or one of its subtypes contains all the information required to convert a MediaModel into a new format. The main attributes to do so are:

- conversionStrategy: An attribute that contains the Spring bean ID of the MediaConversionStrategy to use. This bean must be available in the Spring context.
- conversion: An attribute that contains the conversion instructions. This attribute is dependent on the MediaConversionStrategy used.

  To follow examples of bitmap image conversions, see the *ImageMagick Usage* section of ImageMagick Configuration.
- Mime Type of the output: Optional. It can be set depending on the conversionStrategy used . If no mime type is provided, the mime type of the media input is used.

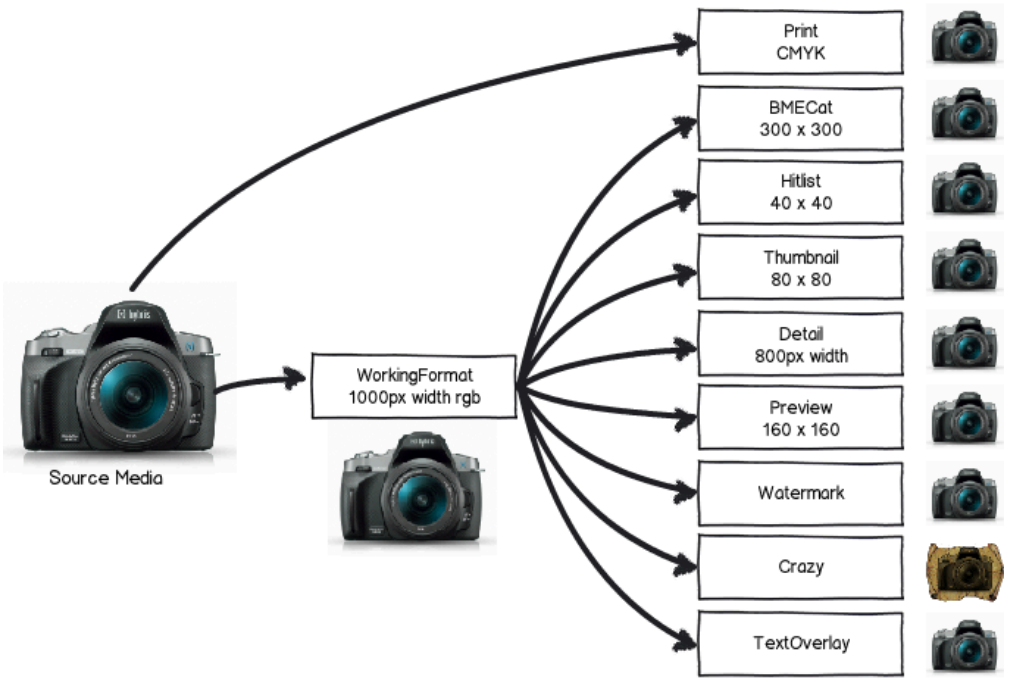
Additional Medias Used in the Conversion

As some conversions need additional input besides the input media, the ConversionMediaFormat hosts a list of mediaAddOns. For example, you may want to put a watermark image over an image or apply a special color profile in the color space transformation for a print production. Having an attached XSLT stylesheet, it is also possible to convert XML documents from one format to another. The interpretation and usage of additional media is up to the MediaConversionStrategy implementation.

Conversion Chaining

The ConversionMediaFormat can be chained. For example, a ConversionMediaFormat can restrict its input media to be of a certain ConversionMediaFormat. This is important as some conversion operations are consuming procedures and should not be redone for each target format. When you have very large input images are very large, it does not make sense to derive all web frontend images from this huge binary data. A working format could be defined which is then reused as a conversion input. With this, it is also possible to build complete conversion pipelines.

The following diagram shows such scenario for images:



Conversion Group

Not all input media are supported or appropriate for every converted media format. For example, your banner or promotion images need to be converted to completely different formats than your product detail pictures. You have technical documentation such as XML assigned to your product that must be converted, but in a different way than your images. To reflect this, the `mediaconversion` extension introduces the concept of a ConversionGroup, which is a simple aggregation of target conversion formats attached to the MediaContainer. If this optional attribute is set on the MediaContainer, all conversion mechanisms are converted to the referenced ConversionMediaFormat, including parent formats in the conversion chain.

Specifying a ConversionGroup is optional. If none is set, all ConversionMediaFormats in the system are considered as target formats. If you wrap your product pictures in a MediaContainer and want to leverage conversion within SAP Commerce for any asset, you need to consider ConversionGroups.

Conversion Status

The `mediaconversion` extension exposes a dynamic attribute to monitor the current conversion status of a MediaContainer. The states are defined as follows:

Status	Description
CONVERTED	There is a defined master media and all conversion formats (of the associated ConversionGroup or in general) have a corresponding representation in this MediaContainer.

Status	Description
EMPTY	There is no identified master media for this MediaContainer. This is either because there are no medias associated to this MediaContainer or no media has the `original` or the `originalDataPK` attribute set.
PARTIALLY_CONVERTED	There is a defined master media and some converted media present in this MediaContainer, but there are still conversion formats (either in the associated ConversionGroup or in general) which do not have a representation in the MediaContainer.
UNCONVERTED	There is a defined master media but no converted medias in this MediaContainer.

Conversion Error Log

If a conversion failed, the `mediaconversion` extension generates a ConversionErrorLog entry within the database that contains the fault information. These entries are automatically deleted upon successful conversion of the media to the specified format. The latter behavior can be turned off, for example the ConversionErrorLog is kept forever by adjusting the `mediaconversion.removeConversionErrorLogUponSuccess` property to false.

Conversion Mechanisms

Spring Service

The new MediaConversionService exposes the functionality to convert media within the MediaContainer to different formats declared in the ConversionMediaFormat. The MediaConversionService functionality to retrieve media in a specify format, `getMediaByFormat()`, remains unchanged and queries the database for a matching media. It throws a ModelNotFoundException if no such media exists. It is important to note that the MediaContainer becomes the central item for automatic conversion. The media must reside in a MediaContainer with a marked master, for example, media of format `original`, to be converted by the provided service or cron jobs. The targeted format must be a ConversionMediaFormat instance.

Media Conversion Cron Job

In addition to the above-mentioned MediaConversionService that supports explicit conversion calls, the `mediaconversion` extension comes with a MediaConversionCronJob. The MediaConversionCronJob queries the database for all missing media in any MediaContainer and triggers the conversion accordingly.

The MediaConversionCronJob supports the following configuration options:

- maxThreads: The number of threads to use for conversion. Setting this value higher than the limit of the **processExecutor** is counterproductive.
- catalogVersion: Restricts the examined MediaContainer to a certain catalogVersion.
- includedFormats: Specifies a list of ConversionMediaFormat which are to be converted. If this property is not set, all ConversionMediaFormat in the system are converted.
- asynchronous: If set to `true`, the MediaConversionCronJob only creates a task instance to make the conversion—one for each MediaContainer for which some action needed. With this, you can leverage the whole cluster to make conversions, but

the tasks created are not restricted to a specific cluster node.

A MediaConversionCronJob is used to generate derived medias: one to extract media metadata and one to delete them. They are set up in the `mediaconversion` extension project data upon initialization or update. The cron jobs are deactivated and serve as a template for further customization.

Media Conversion Task Runner

Additionally, there is a MediaConversionTaskRunner that leverages asynchronous media conversion with the help of the Task Engine. This implementation is used by the MediaConversionCronJob with set `asynchronous` configuration option. Although the MediaConversionTaskRunner accepts standard TaskModel instances for processing, it expects a MediaConversionTaskContext instance set as the `TaskContext` attribute.

On Demand Conversion

Another integrated way to convert media to different formats is called `on demand` or `on the fly` conversion. For example, converting the media to the desired format when they are first accessed like on a website. This feature is useful in scenarios in which medias are uploaded and updated very frequently.

Flow

When converting medias on the fly on a website, it is important to separate the HTML rendering request from the request that makes the conversion. As shown in the following figure, the complete flow is separated into three phases:

1. Upon the page request (1) the controller, servlet, or JSP checks for the availability of the media in the required format (2). If such a media exists, its URL is returned as usual. If not, a substitute URL is embedded in the HTML page and returned (3).
2. The client browser requests the media from the substitute URL (4). A simple servlet makes the conversion (5) and sends back a redirect to the URL of the newly created image (6).
3. The client requests the media (7) and retrieves it (8). Note that the media delivery through a HTTP Deamon is still possible.

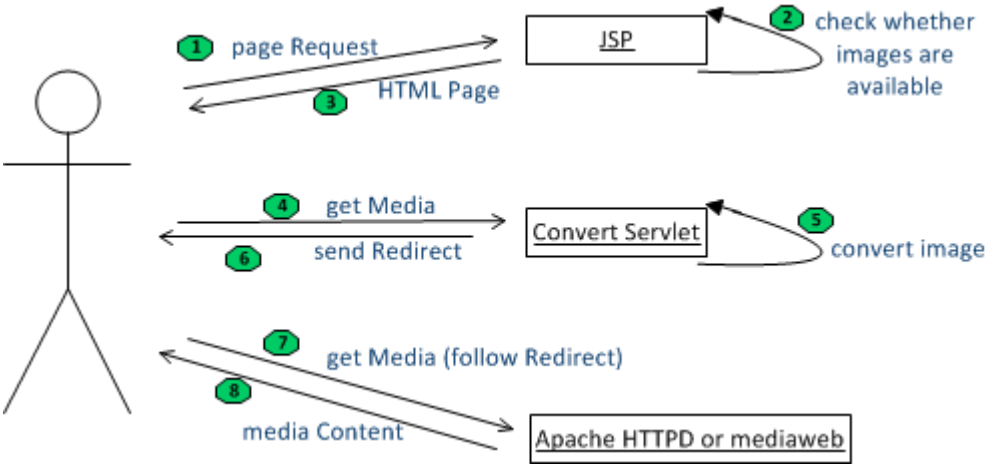


Figure: The sample process of on demand conversion.

Convert Servlet

This feature is already incorporated in the `mediaconversion` extension. Therefore the `mediaconversion` extension exposes a servlet, `/mediaconversion/convert`, that handles the appropriate substitute URLs. You need to expose and secure the conversion servlet accordingly. The accepted URL is `/mediaconversion/convert/` ***PK of MediaContainer /qualifier of ConversionMediaFormat***.

Tag Library

To make it easier to leverage on-demand conversion, the `mediaconversion` extension comes with a little tag library (taglib) to include task described in step 3 in the Flow section. The taglib URL is **`http://www.hybris.de/jsp/mediaconversion`**.

Enabling the Taglib

To enable the taglib, you have to copy over the taglib descriptor `/mediaconversion/resources/mediaconversion/taglib/mediaconversion.tld` to your web application `WEB-INF/tld` directory. You can automize this by adding the following snippet to your web extension `buildcallbacks.xml` file. To do so, you need to replace all ***yourextension*** occurrences in the code with your extension name

```
<macrodef name="yourextension_before_compile_web">
        <sequential>
                <mkdir dir="${ext.yourextension.path}/web/webroot/WEB-INF/tld" />
                <copy file="${ext.mediaconversion.path}/resources/mediaconversion/taglib/mediaconve
                        tofile="${ext.yourextension.path}/web/webroot/WEB-INF/tld/mediaconversion.t
        </sequential>
</macrodef>
```

URL Tag

URL tag is a tag to output or set a variable to the URL of a media in a given format. The URL generated either points directly to the media or, if the specified format has to be generated first, to the `mediaconversion` extension servlet that makes conversion in a different thread.

The following table describes the attributes:

Name	Description	Required	Runtime Expression	Type
container	The container in which the media in question resides.	true	true	MediaContainerModel
format	The qualifier of the output format (MediaFormatModel). If no format is specified or the specified format is the empty string, the master media of the container is used.	false	true	java.lang.String
scope	The scope of the variable to set. One of page, request, session, or application.	false	false	java.lang.String
var	The name of the variable to set.	false	false	java.lang.String

HTML Image Tag

HTML Image Tag renders a HTML <img> element that references a (converted) media in a specified format. The generated URL either points directly to the media or, if the specified format has to be generated first, to the mediaconversion convert servlet that makes conversion in a different thread.

The following table describes the attributes:

Name	Description	Required	Runtime Expression	Type
align	Optional attribute for the generated HTML <img> element. The possible values for this attribute are top, middle, bottom, left, and right.	false	true	java.lang.String
alt	Optional alternative text to override alternative text set on the media.	false	true	java.lang.String
border	Optional attribute for the generated HTML <img> element.	false	true	java.lang.String
container	The container to output.	true	true	MediaContainerModel
cssClass	Optional CSS class to be used.	false	true	java.lang.String
format	The qualifier of the output format (MediaFormatModel). If no format is specified or the specified format is the empty string, the master media of the container is addressed.	false	true	java.lang.String
height	Optional height of the generated <img> element. This height is not obeyed in an image scaling, for example it is only used in the generated HTML output.	false	true	java.lang.String
id	Optional attribute for the generated HTML <img> element.	false	true	java.lang.String
ismap	Optional attribute for the generated HTML <img> element.	false	true	java.lang.Boolean
longdesc	Optional attribute for the generated HTML <img> element.	false	true	java.lang.String

Name	Description	Required	Runtime Expression	Type
name	Optional attribute for the generated HTML `<img>` element.	false	true	java.lang.String
style	Optional (CSS) style attribute.	false	true	java.lang.String
title	Optional attribute for the generated HTML `<img>` element.	false	true	java.lang.String
usemap	Optional attribute for the generated HTML `<img>` element.	false	true	java.lang.String
vspace	Optional attribute for the generated HTML `<img>` element.	false	true	java.lang.String
width	Optional width of the generated `<img>` element. This width is not obeyed in an image scaling, for example it is only used in the generated HTML output.	false	true	java.lang.String

Media MetaData Extraction

The `mediaconversion` extension furthermore hosts an extensible mechanism to extract metadata from arbitrary medias. It can be triggered within Backoffice or by cron job. The `mediaconversion` extension comes with an implementation for bitmap image data based on ImageMagick. It is used to extract various information stored in the image itself such as dimensions, color space, Exif data.

The metadata extraction can be easily extended by implementing the MediaMetaDataProvider interface and registering the implementation in the Spring context. All implementations of MediaMetaDataProvider are picked up and called. It is important to restrict implementation to the applicable mime types.

Related Information

The Task Service
ImageMagick Configuration
Media Conversion
ImageMagick - Integration Guide
task Extension - Technical Guide

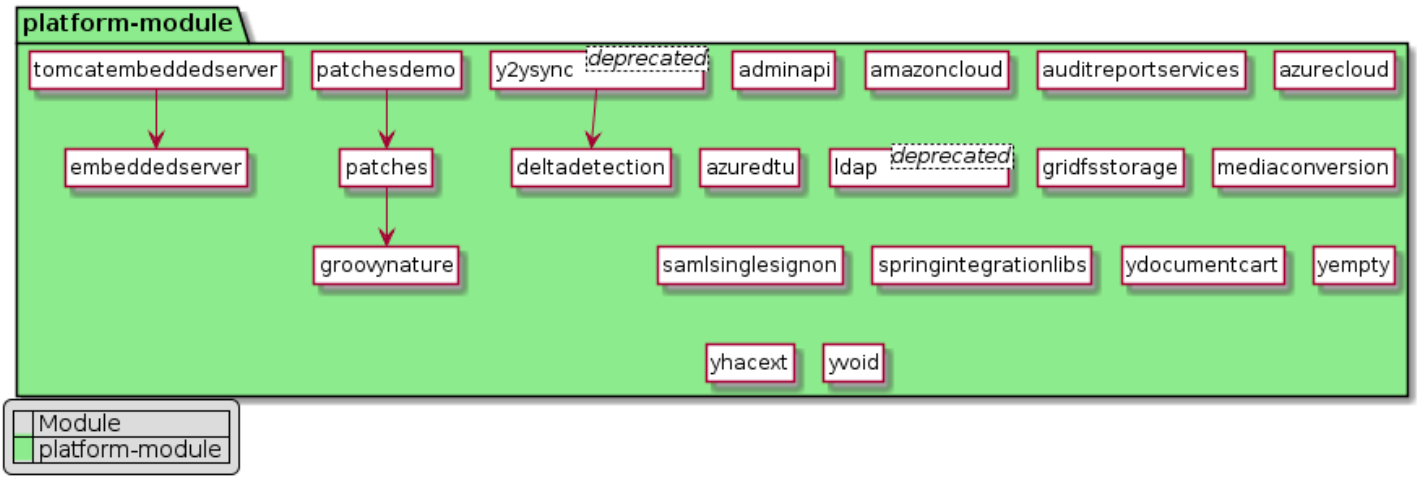
patches Extension

Patches enable you to effectively manage data in a project lifecycle. Patches improve the approach to system maintenance and solve the problem of managing project data across different environments.

About the Extension

Name	Directory	Related Module
patches	hybris/bin/modules/platform	Platform Module Architecture

Dependencies



Dependencies

Related Information

Patches

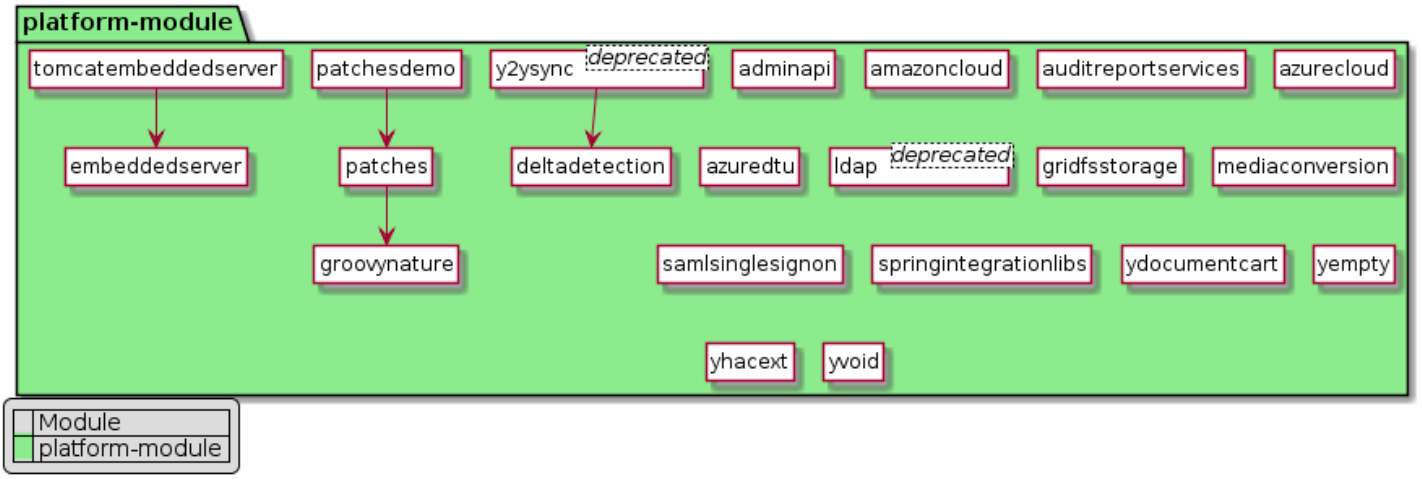
patchesdemo Extension

The `patchesdemo` extension demonstrates how to implement and use Patches.

About the Extension

Name	Directory	Related Module
patches	hybris/bin/modules/platform	Platform Module Architecture

Dependencies



Dependencies

Related Information

Patches

samlsinglesignon Extension

The `samlsinglesignon` extension allows you to connect SAP Commerce with external identity provider (IDP) systems allowing for a cross-domain single sign-on functionality.

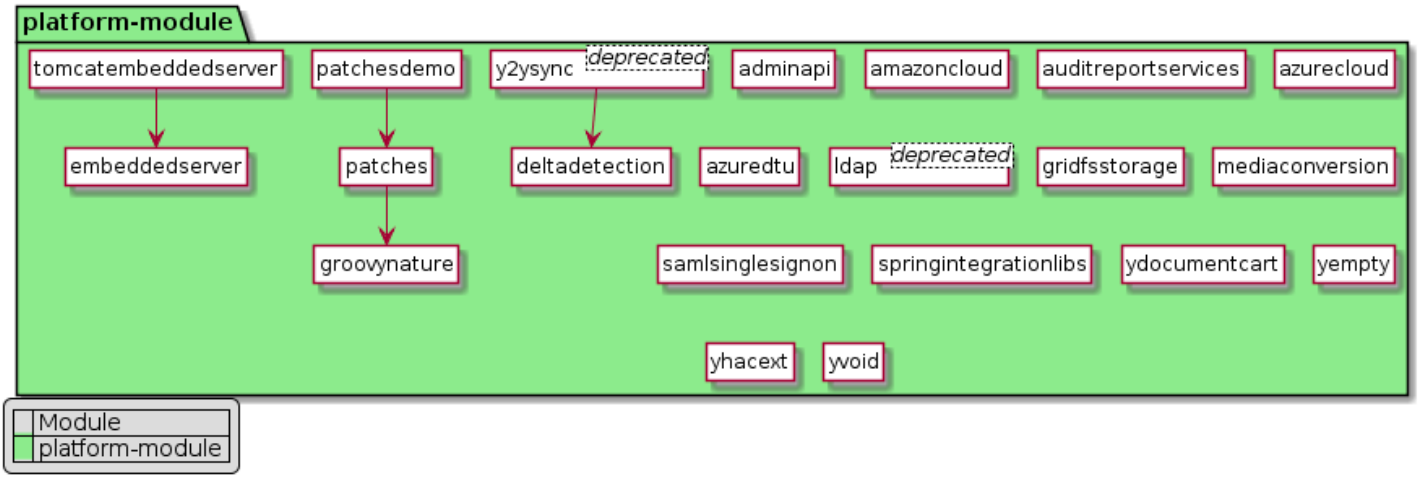
ℹ Note

An SAP Commerce extension may provide functionality that is licensed through different SAP Commerce modules. Make sure to limit your implementation to the features defined in your contract license. In case of doubt, please contact your sales representative.

About the Extension

Name	Directory	Related Module
`samlsinglesignon`	`hybris/bin/modules/platform`	Platform Module Architecture

Dependencies



Dependencies

Related Information

Single Sign-On in SAP Commerce

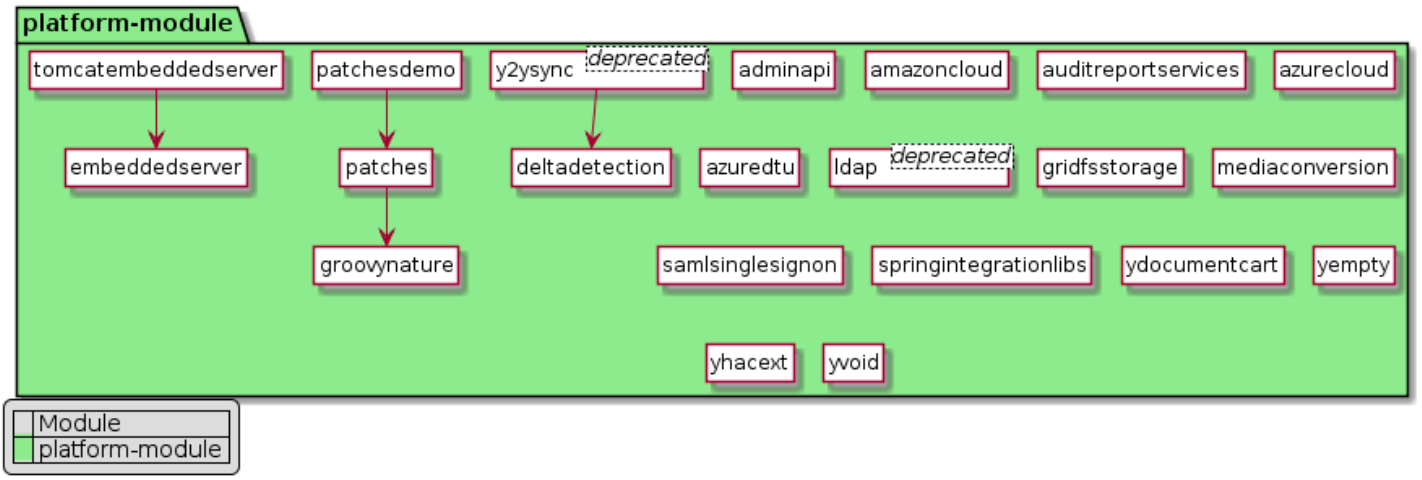
springintegrationlibs Extension

The `springintegrationlibs` extension provides additional open source libraries for IDoc processing.

About the Extension

Name	Directory	Related Module
`springintegrationlibs`	`hybris/bin/modules/platform`	Platform Module Architecture

Dependencies



Dependencies

Related Information

Spring Framework in SAP Commerce

tomcatembeddedserver Extension

The `embeddedserver` extension provides an API to run an embedded servlet container. The `tomcatembeddedserver` extension provides a Tomcat-based implementation of this API.

ℹ Note

An SAP Commerce extension may provide functionality that is licensed through different SAP Commerce modules. Make sure to limit your implementation to the features defined in your contract license. In case of doubt, please contact your sales representative.

ydocumentcart Extension Template

The `ydocumentcart` extension is a template extension that allows you to store selected types in an alternative storage. It uses the polyglot persistence query language.

The `ydocumentcart` extension provides `de.hybris.platform.persistence.polyglot.repository.documentcart.Repository` that is an implementation of the `ItemStateRepository` interface. The `Repository` implementation stores the instances of the following types in the HSQL database:

- `Cart`
- `CartEntry`
- `AbstractOrderEntryProductInfo[orderEntry]`
- `PromotionResult[order]`
- `AbstractPromotionAction[promotionResult]`
- `PaymentTransaction[order]`
- `PaymentTransactionEntry[paymentTransaction]`
- `PromotionOrderEntryConsumed[promotionResult]`

`ydocumentcart` extension template supports the following databases:

- Azure SQL Server
- HSQL
- MySQL

To enable MySQL support, uncomment the following section in your `local.properties` file:

```
## MYSQL DB settings
ydocumentcart.storage.jdbc.url=jdbc:mysql://localhost:3306/hybris?useConfigs=maxPerformance&charact
ydocumentcart.storage.jdbc.driver=com.mysql.jdbc.Driver
ydocumentcart.storage.jdbc.user=hybris
ydocumentcart.storage.jdbc.password=change_me
ydocumentcart.storage.jdbc.props.cachePrepStmts=true
ydocumentcart.storage.jdbc.props.prepStmtCacheSize=250
ydocumentcart.storage.jdbc.props.prepStmtCacheSqlLimit=2048
```

Each entry is stored in a JSON format, which is similar to handling data in a document-based database. For better performance, key values such as identifiers are stored in separate columns.

The `ydocumentcart` extension template provides the `QueryFactory` class that makes conversation with a document database easier.

To store other item types in an alternative storage, extend supported types list or implement an `ItemStateRepository` class in your custom code.

Using Cache

The provided implementation of the `ydocumentcart` repository contains an internal request-wide cache to store documents used by a given request. To use this cache, add a callback responsible for initializing the cache before processing the request and flushing it once the request has been processed. The callback is `ydocumentcartpackage.servicelayer.web.DocumentCartRepositoryCallback`. It is an implementation of `de.hybris.platform.servicelayer.web.PolyglotPersistenceCallbackFilter.PolyglotCallback` and is automatically enabled once an extension containing it is added to the setup. For this mechanism to work, include the

`de.hybris.platform.servicelayer.web.PolyglotPersistenceCallbackFilter` filter in your filter chain in `web.xml` of the Tomcat server.

Deleting Cart Tables

In an extension generated from the `ydocumentcart` template, cart tables in the database are not deleted during initialization - they have to be deleted manually.

Support for Serializable Types

You can create a list of comma-separated serializable POJO types that can be stored in extensions created from the `ydocumentcart` template by adding the following property to the `local.properties` file:

```
<ydocumentcart>.storage.jsonSerializer.supportedSerializableTypes=
```

Replace `<ydocumentcart>` with the name of your extension that is based on `ydocumentcart`. Add the serializable POJO types as the value of the property, for example:

```
ydocumentcart.storage.jsonSerializer.supportedSerializableTypes=de.hybris.platform.core.order.Entry
```

Any instances of the types listed in the value of this property are serialized, encoded with Base64, and stored as a string. When the document is read from the storage, the string is decoded and deserialized.

Note

Due to security reasons, it's impossible to store a serializable object without adding its type to the value of the property.

Related Information

Polyglot Persistence Query Language
Polyglot Persistence

yempty Extension Template

The `yempty` extension template is a predefined extension to be duplicated. The copy serves as starting point for creating a new extension, typically used for customer specific implementations.

The yempty extension is an empty extension with minimal implementations of the following extension modules[1]:

- `core`
- `web`

[1] Extension modules are structural elements of an extension. They do not denote those modules that are referred to as product components, such as the `core` extension.

Related Information

core Extension
Creating Web Applications

yhacext Extension Template

The `yhacext` extension allows you to add your own functionalities to SAP Commerce Administration Console.

For more information, see Adding Functionality to Administration Console.

yvoid Extension Template

The `yvoid` extension template is a predefined extension to be duplicated. The copy serves as starting point for creating a new extension, typically used for customer specific implementations. It is a simpler alternative for the `yempty` extension template.

A `yvoid` extension is an empty extension with a highly simplified implementation of the extension`core` module[1]. An extension configured out of `yvoid` is a Jalo-logic-free extension.

[1] Extension modules are structural elements of an extension. They do not denote those modules that are referred to as product components, such as the `core` extension.

Related Information

core Extension
yempty Extension Template

Platform Module Implementation

Learn how to implement features provided by the Platform module in the modules directory.

AdminAPI

AdminAPI is a highly extensible REST API designed for exposing admin-related functionalities.

Delta Detection

Thanks to delta detection, you can detect changes in items. It's intended to be used by everyone who needs to react to changes done on item data inside the Platform, for example for updating 3rd party systems, updating other items or performing validation. Delta detection exists as a standalone extension and is therefore independent from the catalog synchronization.

Handle Media Using the MediaContainer

A `MediaContainer` is a representation of a given `Media` and all its formats acquired in a conversion procedure. A `MediaContainer` can hold any number of `MediaFormats` associated with the given `Media`. The main role of `MediaContainer` is to group `Media` and all its formats after making conversion of that `Media`. A `MediaFormat` is implied as a media with changed format or resized dimensions.

Using Amazon S3 Media Storage Strategy

Amazon S3 provides a simple web services interface that can be used to store and retrieve any amount of data. You can configure a specific MediaFolder to store binary data of a Media item directly in Amazon S3.

Using MongoDB GridFS Media Storage Strategy

MongoDB database provides specification for storing large files called GridFS. You can configure a specific MediaFolder to store binary data of a Media item directly in MongoDB.

Windows Azure Blob Media Storage Strategy

Windows Azure Blob provides a simple web services interface that can be used to store and retrieve any amount of data. You can configure a specific MediaFolder to store binary data of a Media item directly in Windows Azure Blob.

Converting Media with ImageMagick

Create, edit, compose, or convert almost all available bitmap image formats using media conversion and ImageMagick.

Patch Concepts and Usage

The Patches feature is based on the concepts of a setup class, an organization unit, a patch, and an action. There are various configuration options for enabling the Patch functionality.

[About the ldap Extension]

LDAP is a protocol that defines the method by which directory data is accessed. It defines and describes how data is represented in the directory service (the Data Model). It also defines how data is imported and exported in a directory service (using LDIF).

[Single Sign-On in SAP Commerce]

Single Sign-On (SSO) shares login authentication across multiple applications. It allows users to sign into one system and access multiple associated systems without having to sign into these again.

AdminAPI

AdminAPI is a highly extensible REST API designed for exposing admin-related functionalities.

Admin Controller

You can create the admin controller from any extension. In the global context, create a package that contains the `adminapi` name, for example `de.hybris.platform.adminapi.controller`. Annotate the controller with `@AdminApiController`.

Example:

```
package de.hybris.platform.adminapi.components.console;

import de.hybris.platform.adminapi.annotation.AdminApiController;

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;


@AdminApiController
@RequestMapping("/admincontroller")
public class AdminController
{
   @GetMapping
   ResponseEntity<String> sayHello()
   {
      return ResponseEntity.ok("Hello !");
   }
}
```

The `/admincontroller` endpoint address is `https://localhost:9002/adminapi/admincontroller`.

! Restriction

Do not use occupied request mappings.

AdminApiSystemException

The `AdminApiSystemException` exception is handled by `@ControllerAdvice` and is returned in a response. You can throw this exception in a few ways:

```
AdminApiSystemException(final String message, final HttpStatus httpStatus)
AdminApiSystemException(final Throwable cause, final HttpStatus httpStatus)
AdminApiSystemException(final String message, final Throwable cause, final HttpStatus httpStatus)
```

Security

The `adminapi` extension uses Spring security to secure the endpoints out of the box. The OAuth Resource Server configuration is located in the `adminapi-spring-security-config.xml` file in the `adminapi` extension. To extend this configuration, create a `yourextension-security-adminapi.xml` configuration file in `/yourextension/resources`.

See a sample configuration file:

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:sec="http://www.springframework.org/schema/security"
       xsi:schemaLocation="http://www.springframework.org/schema/beans http://www.springframework.c
       http://www.springframework.org/schema/security
       http://www.springframework.org/schema/security/spring-security.xsd">

    <sec:http pattern="/admincontroller/**" security="none"/>

</beans>
```

Swagger

The `adminapi` extension uses Swagger to document REST API.

You can change some properties in the `SwaggerConfig` class. It's located in `/adminapi/web/src/de.hybris.platform.adminapi.components.config`. You can configure the endpoint using the `adminapi.springfox.documentation.swagger.v2.path` property. The default value for the property is set to `/api-docs` in the `project.properties` file in the `adminapi` extension.

To document your REST API, use Swagger annotations. For example:

```
package de.hybris.platform.adminapi.components.console;
import de.hybris.platform.adminapi.annotation.AdminApiController;

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;

import io.swagger.annotations.ApiOperation;
import io.swagger.annotations.ApiResponse;
import io.swagger.annotations.ApiResponses;

@AdminApiController
@RequestMapping("/admincontroller")
public class AdminController
{
   @ApiOperation(value = "Say hello")
   @ApiResponses(
   { @ApiResponse(code = 200, message = "Operation succeeded") })
   @GetMapping
   ResponseEntity<String> sayHello()
   {
      return ResponseEntity.ok("Hello !");

   }
}
```

For details, see https://github.com/swagger-api/swagger-core/wiki/Annotations#quick-annotation-overview

The Swagger UI address is `https://localhost:9002/adminapi/swagger-ui.html`.

Delta Detection

Thanks to delta detection, you can detect changes in items. It's intended to be used by everyone who needs to react to changes done on item data inside the Platform, for example for updating 3rd party systems, updating other items or performing validation. Delta detection exists as a standalone extension and is therefore independent from the catalog synchronization.

How Delta Detection Works

ChangeDetectionService can be used for finding changes either by given item or by type. Finding changes by type returns all items for which any kind of changes (**new**, **modified**, **deleted**) have been found. With **ChangeDetectionService** it is possible to consume the changes so that they do not turn up again when running another detection.

ItemVersionMarker is used to mark changes; it holds a reference to the changed item and some additional information like the change type, or the version timestamp.

Another important thing is the so-called "stream awareness", which is taken into account by the API. It means that every stream can see different states of the current items. Using the API, it is required to provide a proper **streamID ,** which looks up the changes only for a given stream. The **ItemVersionMarker** mentioned above holds additionally information about the stream id. If you consume changes for a given stream so that it no longer detects ''old'' changes, yet the other stream will still be able to detect changes.

Finding Items by Type

It is possible to get all the changed items for a given type (**composedType**), with a method from **ChangeDetectionService** *:*

```
void collectChangesForType(ComposedTypeModel composedType, String streamId, ChangesCollector collec
```

How the changes are collected is defined inside the given **ChangesCollector** implementation. The delta detection feature provides by default two implementations: Collecting the changes in memory or storing the changes as a csv file.

The result could be all the changes as an **ItemChangeDTO** List (using the in-memory Collector).

New Items

By new items, we understand items without an **ItemVersionMarker**. The item has been created, but no **ItemVersionMarker** (which should hold the reference for the created item) exists yet.

Deleted Items

Deleted items do not exist in the database anymore, but the **ItemVersionMarker** referencing them still exists (changes have not been consumed yet, see Up-to-date State).

Modified Items

Modifed items have an existing **ItemVersionMarker** referencing them, but the modification date of the Item is newer than the version timestamp saved in the **ItemVersionMarker**

Up-to-date State

This state can be achieved by calling **consumeChanges(),** for a given changes list. For any existing Item in the up-to-date state, there is an existing **ItemVersionMarker** with the same versionTS as the last item modified time.

For non-existing items, there are no **ItemVersionMarkers** anymore.

Consuming Changes

After detecting changes, we have an option to "consume" them, which means bringing all the items with their **ItemVersionMarkers** to the **Up-to-date** state.

This is possible by calling:

```
void consumeChanges(List<ItemChangeDTO> changes);
```

Finding Changes By Item(PK)

To find the changes by given item (or ItemPk), use the following **ChangeDetectionService** methods:

```
ItemChangeDTO getChangeForExistingItem(ItemModel item, String streamId);
ItemChangeDTO getChangeForRemovedItem(PK pk, String streamId);
```

The result is one **ItemChangeDTO** object, which contains all the necessary info about the change. **Null** value may also be returned in the following cases:

1) for an existing Item - **ItemVersionMarker** exists and is up to date

2) for a removed Item - **ItemVersionMarker** does not exist

Generating CSV Reports

It is possible to generate a CSV report of detected changes. To generate a report, use the **changeDetectionService** method **collectChangesForType(...),** providing the **CsvReportChangesCollector** as parameter.

Generating reports is also possible with some dedicated jobs (like **ChangeDetectionJob**), which may internally call the API method above.

Detecting and Consuming Changes with Cronjobs

Delta detection provides two dedicated jobs:

- **ChangeDetectionJob** - it is used by cronjob to detect all changes for a given **composedType** and generates a Media Item, referencing the csv file that lists those changes.

- **ConsumeAllChangesJob** - it is used by cronjob to consume all changes for a given resource (referenced by Media item)

- **ScriptChangeConsumptionJob** - it is used by cronjob to consume the changes for a given resource (referenced by Media item). The consuming process is delegated to proper script (by given scriptURI)

Using Delta Detection

The following examples demonstrate the delta detection feature using the scripting languages console (see hybris Scripting Engine for more information). All the steps can be executed at runtime, and rebuilding the platform is not needed.

Detecting and consuming changes for Single Item (PK)

The following steps demonstrate detecting and consuming changes for a single item

1. Prepare a new item

```
import de.hybris.platform.core.model.user.CustomerModel
CustomerModel jan = modelService.create(CustomerModel.class);
jan.setName('jan');
jan.setUid('jan');
modelService.save(jan)
```

The user should be persisted in the database (users table)

Go to the Scripting Console, enable the commit mode, and create a new client.

2. Since there is no ItemVersionMarker saved yet, there should be a change detected for the new customer, execute the lines below to find the change

```
import de.hybris.platform.core.model.user.CustomerModel
CustomerModel jan = userService.getUserForUID('jan')
change = changeDetectionService.getChangeForExistingItem(jan, 'streamXXX')

//change.properties.each{ println it}  - uncomment this to see a nicer view of the changes
```

As a result, you should see the **ItemChangeDTO** containing all the necessary information about the change.

3. Consume the changes and verify they are not detected anymore

Enable the commit mode (we will store the **ItemVersionMarker** internally) and execute the following

```
changeDetectionService.consumeChanges([change])
```

The **ItemVersionMarker** for the given customer has been stored in the database

4. Now, try to find the changes again:

```
change = changeDetectionService.getChangeForExistingItem(jan, 'streamXXX')

//change.properties.each{ println it}  - uncomment this to see a nicer view of the changes
```

No changes should be found, they have been just consumed (by setting the **ItemVersionMarker** property).

5. Let's update our customer.

```
jan.name='jan changed'
modelService.save(jan)
```

and detect the changes again:

```
change = changeDetectionService.getChangeForExistingItem(jan, 'streamXXX')

//change.properties.each{ println it}  - uncomment this to see a nicer view of the changes
```

You should see now again the change detected, this time showing the **ChangeType = MODIFIED.**

Try consuming changes and detecting them again; the expected result is the same is in previous scenario::

```
changeDetectionService.consumeChanges([change])

// ItemVersionMarker is updated now - no changes should be detected anymore:

change = changeDetectionService.getChangeForExistingItem(jan, 'streamXXX')
```

Detecting and consuming changes by Type

Here the task is to find what changes have taken place (**New**, **Modified**, **Deleted**) for a given type. Next, consume those changes so that they are not detected again.

1. Import a few items in the Impex console

```
INSERT_UPDATE Title; code[unique=true];
;titleFoo
;TitleBar

INSERT_UPDATE DeliveryMode; code[unique=true];
;testDeliveryModeX
;testDeliveryModeY
;testDeliveryModeZ
```

   This will create two new **Titles** and three **deliveryModes**.

2. Find the changes by type

```
import de.hybris.platform.core.model.user.TitleModel
import de.hybris.deltadetection.impl.InMemoryChangesCollector

collector = new InMemoryChangesCollector()
changeDetectionService.collectChangesForType(typeService.getComposedTypeForClass(TitleModel.c
//try this as well
//import de.hybris.platform.core.model.order.delivery.DeliveryModeModel
//changeDetectionService.collectChangesForType(typeService.getComposedTypeForClass(DeliveryMo

collector.getChanges()        // use this for better overview - .each {println it}
```

   As a result, you should see the detected changes for a given type. Only changes with changeType **NEW** should be found

3. Consume changes so that they are not detected anymore.

   Enable the commit mode, c onsume all the changes for type **Title** (you need to get the changes first, just like in the previous step):

```
changeDetectionService.consumeChanges(changes)

//to verify - try that:
//import de.hybris.deltadetection.impl.InMemoryChangesCollector
//collector = new InMemoryChangesCollector()
//changeDetectionService.collectChangesForType(typeService.getComposedTypeForClass(TitleModel
//collector.getChanges().each {println it}
```

   Consume the changes for Delivery Mode items, too.

4. Now that we have all our items in the up-to-date state, go to the impex import console and introduce new changes:

```
INSERT_UPDATE Title; code[unique=true];name
;titleFoo;foo
;TitleNew;new

INSERT_UPDATE DeliveryMode; code[unique=true];name
;testDeliveryModeX;xxx
;testDeliveryModeY;yyy
;testDeliveryModeNew;New

REMOVE Title;code[unique=true];
;TitleBar
REMOVE DeliveryMode;code[unique=true];
;testDeliveryModeZ
```

   The following changes have been made:

   Title - 1 new item (TitleNew), 1 modified (TitleFoo), 1 removed (TitleBar)

   DeliveryMode - 1 new item (testDeliveryModeNew), 2 modified (xxx, yyy), 1 removed (testDeliveryModeZ)

5. Detect the new changes by type

```
import de.hybris.platform.core.model.user.TitleModel

import de.hybris.deltadetection.impl.InMemoryChangesCollector
collector = new InMemoryChangesCollector()
changeDetectionService.collectChangesForType(typeService.getComposedTypeForClass(TitleModel.c
//try this as well
//import de.hybris.platform.core.model.order.delivery.DeliveryModeModel
//changeDetectionService.collectChangesForType(typeService.getComposedTypeForClass(DeliveryMo

collector.getChanges().each {println it}
```

6. Try generating a CSV file with changes

   Based on the state above, detect the changes this time like follows:

```
import de.hybris.platform.core.model.user.TitleModel
import de.hybris.deltadetection.impl.CsvReportChangesCollector

writer = new FileWriter(new File("e://my_report_title.csv"))
collector = new CsvReportChangesCollector(writer)
changeDetectionService.collectChangesForType(typeService.getComposedTypeForClass(TitleModel.c
writer.close()
//try this as well
//import de.hybris.platform.core.model.order.delivery.DeliveryModeModel
//changeDetectionService.collectChangesForType(typeService.getComposedTypeForClass(DeliveryMo
```

   This generates a csv file containing the changes. You will find the file under the given location (in this case `e://myReport`)

7. Try detecting with cronjob

   You can trigger a dedicated cronjob to execute detecting changes by type. For that you need to create a cronjob, for example in Impex console like follows:

```
INSERT_UPDATE ChangeDetectionJob; code[unique=true];typePK(code);streamId
;myDetectionJobForTitle;Title;streamXXX;

INSERT_UPDATE CronJob; code[unique=true];job(code);singleExecutable;sessionLanguage(isocode)
;myDetectionCronJobForTitle;myDetectionJobForTitle;true;en
```

   Now go to the scripting console and execute the cronjob (enable the commit mode in this case):

```
cronjob = cronJobService.getCronJob("myDetectionCronJobForTitle")
cronJobService.performCronJob(cronjob,true)
```

   This should detect the changes and generate a csv file and a Media item, which references that file. The Media item is stored under the ***job.output*** attribute.

8. consuming changes with cronjob

   Using a dedicated cronjob - it's quite easy to consume the given changes (given as Media item). The following consumption example will show the Script based job usage

   prepare a script in the scripting console:

```
log.info('input resource (Media code): ' + cronjob.job.input.code)
changeDetectionService.consumeChanges([change])
log.info('Consumed: ' + change)
true
```

   The script will be responsible of process the given change. In this case it will just consume the change and print out some useful logs.

   The returned result - ***true***, means the processing can continue with another changes given the job as input (returning ***false*** will stop the processing).

Click on save button to store the script in database (set the name ***consumeAllScript***).

Now we need to prepare our consumption job. We need the scriptURI and the input (as Media) for that (enable the commit mode for that):

```
import de.hybris.platform.deltadetection.model.ScriptChangeConsumptionJobModel
myConsumeJobForTitle = modelService.create(ScriptChangeConsumptionJobModel.class);
myConsumeJobForTitle.setCode("myConsumeJobForTitle");
myConsumeJobForTitle.setScriptURI("model://consumeAllScript");

cronjob = cronJobService.getCronJob("myDetectionCronJobForTitle")
myConsumeJobForTitle.input=cronjob.job.output
//to verify your model before saving - myConsumeJobForTitle.properties.each {println it}
modelService.save(myConsumeJobForTitle)
```

And of course, we need a cronjob for our job as well. Use the impex import console to create it:

```
INSERT_UPDATE CronJob; code[unique=true];job(code);singleExecutable;sessionLanguage(isocode)
;myConsumeCronJobForTitle;myConsumeJobForTitle;true;en
```

Finally - let's execute the consumption cronjob in the scripting console (enable the commit mode here):

```
cronjob = cronJobService.getCronJob("myConsumeCronJobForTitle")
cronJobService.performCronJob(cronjob,true)
```

Observe the logs - there should de the output of the prepared script visible. The changes should be consumed now. Verify That with the changeDetectionService (see step e)

Related Information

catalog Extension - Technical Guide

Change Detection

To detect changes, delta detection executes a query by using the `processSearchRows` method from the `FlexibleSearchService` API .

The method is defined as:

```
/**
 * Search.
 *
 * @param searchQuery the search query
 * @param rowConsumer consumer for rows fetched from DB
 */
default <T> void processSearchRows(final FlexibleSearchQuery searchQuery, final Consumer<T> rowCons
```

The combination of the `disableAutoCommit` property and a `PreparedStatementHint` that adjusts the `fetchSize` of retrieved data allows you to achieve a small memory footprint without significant performance loss. Such an approach prevents out-of-memory errors for large results returned.

With the provided default implementation of the service, when FlexibleSearch retrieves data from the database, the `processSearchRows` method executes the passed `rowConsumer` function against each row. The `processSearchRows` method returns the same results as the standard search method. The method doesn't cache resulting rows - when you call it, it fetches data directly from the database.

See the fetch and auto commit property settings for chosen databases:

Database	Property Settings	Description
MySQL	`fetchSize = Integer.MIN_VALUE`	This fetch size enables the streaming mode for a result set.
HSQLDB	Not Applicable	This in-memory local database doesn't suffer from out-of-memory errors, but for large data sets it is extremely slow (uses GC extensively).
Microsoft SQL Server	`fetchSize = 1` `disableAutoCommit = true`	These settings enable streaming-like fetching of the result set.
PostgreSQL	`fetchSize = 1000` `disableAutoCommit = true`	Setting the fetch size for 1000 rows is optimal as it has rather low memory footprint and allows you to keep reading performance at a good level. Disabling auto commit enables fetch size when fetching rows from the database.
SAP S/4HANA	`fetchSize = 1000` `disableAutoCommit = true`	Same as PostgreSQL, good memory to performance ratio.
Oracle	`fetchSize = 100` `disableAutoCommit = true`	These settings allow good memory to performance ratio.

ⓘ Note

The property settings are experimental, may not be efficient, and may depend on a deployment platform.

Handle Media Using the MediaContainer

A `MediaContainer` is a representation of a given `Media` and all its formats acquired in a conversion procedure. A `MediaContainer` can hold any number of `MediaFormats` associated with the given `Media`. The main role of `MediaContainer` is to group `Media` and all its formats after making conversion of that `Media`. A `MediaFormat` is implied as a media with changed format or resized dimensions.

Every `MediaContainers` needs to have a unique `qualifier`.

A `MediaContainer` enables you to provide conversion procedure: getting different formats of a given `Media`. You can define the `MediaFormats` in the `ConversionGroups`.

→ Tip

To follow the concept of processing conversion of a media see mediaconversion Extension.

Using a MediaContainer via the SAP Commerce API

A media container offers simple way to retrieve / store a media objects in a specified media format. Simple example of using container and formats:

```
final MediaContainerModel mediaContainer = modelService.create( MediaContainerModel.class );
mediaContainer.setQualifier( "testContainer" );

final MediaFormatModel format1 = modelService.create( MediaFormatModel.class );
format1.setQualifier( "640x480" );

final MediaFormatModel format2 = modelService.create( MediaFormatModel.class );
format2.setQualifier( "1280x1024" );

final MediaModel mediaInFormat1 = modelService.create( MediaModel.class );
mediaInFormat1.setCode( "media1" );
mediaInFormat1.setMediaFormat( format1 );
mediaInFormat1.setMediaContainer( mediaContainer );

final MediaModel mediaInFormat2 = modelService.create( MediaModel.class );
mediaInFormat2.setCode( "media2" );
mediaInFormat2.setMediaFormat( format2 );
mediaInFormat2.setMediaContainer( mediaContainer );
modelService.saveAll();
```

Now you can get media in format2 from original one:

```
MediaModel retrievedMediaInFormat2 = mediaService.getMediaByFormat( mediaInFormat1, format2 );
```

Using a `MediaContextModel` you can specify fixed mappings from one media format to another.

```
final MediaContextModel mediaContext = modelService.create( MediaContextModel.class );
mediaContext.setQualifier( "testContext" );
modelService.save( mediaContext );

final MediaFormatMappingModel mapping = modelService.create( MediaFormatMappingModel.class );
mapping.setMediaContext( mediaContext );
mapping.setSource( mediaInFormat1 );
mapping.setTarget( mediaInFormat2 );
modelService.save( mapping );
```

Now you can get media in format 2 from media in format 1:

```
MediaModel retrievedMediaInFormat2 = mediaService.getMediaByContext( mediaInFormat1, mediaContext );
```

Using Amazon S3 Media Storage Strategy

Amazon S3 provides a simple web services interface that can be used to store and retrieve any amount of data. You can configure a specific MediaFolder to store binary data of a Media item directly in Amazon S3.

Configuring Amazon S3 Storage Strategy

To configure your custom MediaFolder to store binary data of a Media item in Amazon S3, provide custom values for set of properties in your `local.properties` file. All available properties are listed in the `project.properties` file of `amazoncloud` extension.

ℹ Note

Amazon S3 Media Storage Strategy Configuration Preconditions

To configure your MediaFolder to use Amazon S3, you need to have:

- Amazon Web Services account

- Bucket
- Create access keys pair:
  - Access Key Id
  - Secret Access Key

For more details, read http://aws.amazon.com/documentation/s3/ .

Assume that you want to set up newly created MediaFolder called `s3medias1` to work with Amazon S3. To see how the configuration should look like for particular URL strategy, go to the following sections:

- For local URL strategy: Use Local URL Strategy
- For S3 URL strategy: Use S3 URL Strategy

Use Local URL Strategy

To use local URL strategy, put the following properties in your `local.properties` file:

`local.properties`

```
// Put Spring bean id of the S3 strategy that keeps the logic responsible for communication betweer
                          Commerce and Amazon S3
media.folder.s3medias1.storage.strategy=s3MediaStorageStrategy
media.folder.s3medias1.accessKeyId=YourAccessKey
media.folder.s3medias1.secretAccessKey=YourSecretAccessKey

// Set up your URL strategy
media.folder.s3medias1.url.strategy=localMediaWebURLStrategy
```

Note

Find Out More

Keep in mind that `localMediaWebURLStrategy` is used by default for rendering URLs but data stream is still coming directly from Amazon S3 thanks to `S3MediaStorageStrategy`.

Use S3 URL Strategy

S3 URL strategy is recommended if you want to use direct URLs to Amazon S3. It renders direct URLs, signed or unsigned.

You can use S3 URL strategy, by default with signed URLs and set custom time for how long the URL is valid:

`local.properties`

```
// Put Spring bean id of the S3 strategy that keeps the logic responsible for communication betweer
                          Commerce and Amazon S3
media.folder.s3medias1.storage.strategy=s3MediaStorageStrategy
media.folder.s3medias1.url.strategy=s3MediaURLStrategy
// Set that URL is valid for 5 minutes
media.folder.s3medias1.url.signed.validFor=5
media.folder.s3medias1.accessKeyId=YourAccessKey
media.folder.s3medias1.secretAccessKey=YourSecretAccessKey
```

If you do not want to use signed URLs, provide the following configuration:

`local.properties`

```
// Put Spring bean id of the S3 strategy that keeps the logic responsible for communication betweer
                              Commerce and Amazon S3
media.folder.s3medias1.storage.strategy=s3MediaStorageStrategy
media.folder.s3medias1.url.strategy=s3MediaURLStrategy
media.folder.s3medias1.url.signed=false
media.folder.s3medias1.accessKeyId=YourAccessKey
media.folder.s3medias1.secretAccessKey=YourSecretAccessKey
```

All available properties are listed in the `project.properties` file of `amazoncloud` extension.

Bucket Creation

S3 Storage strategy requires providing bucketId in configuration which points to already existing and configured bucket in S3 storage. To configure SAP Commerce to use such bucket, use following setting:

`local.properties`

```
media.folder.s3medias1.bucketId=myBucket
```

To set up one global bucket for all media:

`local.properties`

```
media.globalSettings.s3MediaStorageStrategy.bucketId=myBucket
```

Use More Than One URL Strategy for One MediaFolder

It is possible to bind more than one URL strategy to the MediaFolder by providing list of comma-separated Spring bean ids. By default first strategy from left is used. It is possible to change used strategy by using local session context and by providing the Spring bean id to use for instance for one particular URL.

For example your MediaFolder is configured like in the following example:

`local.properties`

```
media.folder.s3medias1.url.strategy=localMediaWebURLStrategy,someSpecialURLStrategy,otherUrlStrateg
```

You can change for a moment URL strategy in local session context like in the following code sample:

```
// SessionService is injected by Spring
final SessionService sessionService;

sessionService.executeInLocalView(new SessionExecutionBody()
{
  @Override
  public TranslationResult execute()
  {
    sessionService.setAttribute(MediaManager.PREFERRED_URL_STRATEGY_ID, "someSpecialURLStrategy");
    // render URL here
  }
});
```

Common Configuration Properties

You may configure more than one MediaFolder with the Amazon S3 media storage strategy. Not to repeat the same properties separately for every MediaFolder, you can use global configuration properties. For example, you have the following MediaFolders:

`s3medias1`, `s3medias2`, and `s3medias3` and you want them to work with Amazon S3 media storage strategy. You can shorten the configuration like in the following example:

`local.properties`

```
media.globalSettings.s3MediaStorageStrategy.accessKeyId=YourAccessKey
media.globalSettings.s3MediaStorageStrategy.secretAccessKey=YourSecretAccessKey
media.globalSettings.s3MediaStorageStrategy.endpoint=endpointAddress

media.globalSettings.s3MediaStorageStrategy.url.signed=false
media.globalSettings.s3MediaStorageStrategy.url.unsigned.https=true
media.globalSettings.s3MediaStorageStrategy.url.unsigned.virtualHost=false
media.globalSettings.s3MediaStorageStrategy.url.validFor=120

media.folder.s3medias1.storage.strategy=s3MediaStorageStrategy
media.folder.s3medias1.url.strategy=localMediaWebURLStrategy

media.folder.s3medias2.storage.strategy=s3MediaStorageStrategy
media.folder.s3medias2.url.strategy=localMediaWebURLStrategy

media.folder.s3medias3.storage.strategy=s3MediaStorageStrategy
media.folder.s3medias3.url.strategy=localMediaWebURLStrategy
```

To control cleaning S3 storage on fresh initialization of Platform, use following global property:

`local.properties`

```
media.globalSettings.s3MediaStorageStrategy.cleanOnInit=true
```

For more information, see Media Storage Overview, section **Cleaning Media Storage on Initialization**.

You can find the list of all available common configuration properties in the `project.properties` file of `amazoncloud` extension.

Unsecure Media Storage Strategies

Mixing secured and unsecured media in the same container might weaken the security guarantees related to handling secure media.

When working with the Amazon S3 media storage strategy, make sure that the secured media folder has its own `bucketId` that is not shared with any other media folder, either secured or unsecured. See an example of a configuration that is **not recommended**:

```
media.folder.PRIVATE.storage.strategy=s3MediaStorageStrategy
media.folder.PRIVATE.bucketId=MY_BUCKET
media.folder.PRIVATE.secured=true

media.folder.PUBLIC.storage.strategy=s3MediaStorageStrategy
media.folder.PUBLIC.bucketId=MY_BUCKET
media.folder.PUBLIC.secured=false
```

Cache Setting for a Specific mediaFolder

If you want to use S3-Integration for a specific mediaFolder and your system doesn't work properly with the default `media.default.local.cache.rootCacheFolder=cache` property setting, you can explicitly set this property to apply for S3-Integration on that level:

```
media.folder.s3medias1.storage.strategy=s3MediaStorageStrategy
media.folder.s3medias1.url.strategy=localMediaWebURLStrategy
```

```
media.folder.s3medias1.local.cache.rootCacheFolder=custom
```

Related Information

http://aws.amazon.com/

Using MongoDB GridFS Media Storage Strategy

MongoDB database provides specification for storing large files called GridFS. You can configure a specific MediaFolder to store binary data of a Media item directly in MongoDB.

Configuring MongoDB GridFS Storage Strategy

To configure your custom MediaFolder to store binary data of a Media item in MongoDB, you need to provide custom values for a set of properties in your `local.properties` file. All available properties are listed in the `project.properties` file of the `gridfsstorage` extension.

ℹ Note

MongoDB GridFS Media Storage Strategy Configuration Preconditions

To configure your MediaFolder to use MongoDB, you need to have a MongoDB server up and running. For details, see http://www.mongodb.org/display/DOCS/Home .

Assume that you want to set up a newly created MediaFolder called `mongoFiles` to work with MongoDB. Provide the following configuration:

`local.properties`

```
media.globalSettings.gridFsStorageStrategy.mongo.host=localhost
media.globalSettings.gridFsStorageStrategy.mongo.port=27017
media.globalSettings.gridFsStorageStrategy.mongo.dbname=hybris_storage
media.globalSettings.gridFsStorageStrategy.mongo.username=mongoDbUserName
media.globalSettings.gridFsStorageStrategy.mongo.password=mongoDbSecret

// Put Spring bean id of the GridFS storage strategy that keeps the logic responsible for communica
                                Commerce and MongoDB
media.folder.mongoFiles.storage.strategy=gridFsStorageStrategy
```

Your MongoDB instance must have `security.authorization` enabled. Provide MongoDB credentials with these properties:

- `media.globalSettings.gridFsStorageStrategy.mongo.username`
- `media.globalSettings.gridFsStorageStrategy.mongo.password`

The MongoDB Spring xml configuration uses `mongo:mongo-client` instead of setting credentials directly in `mongo:db-factory`.

ℹ Note

Find Out More

Keep in mind that `localMediaWebURLStrategy` is used by default for rendering URLs but data stream is still coming directly from MongoDB thanks to `GridFSMediaStorageStrategy`.

Automatic Bucket Creation

By default MongoDB GridFS Storage strategy provides automatic bucket creation. Bucket name is computed from current `tenantId` and from the name of used MediaFolder. For example, if current `tenantId` is `master` and MediaFolder name is `root`, then created bucket name is `sys_master_root`. If you like to override this behavior for custom folder, use the configuration property:

`local.properties`

```
media.folder.mongoFiles.bucketId=myBucket
```

To set up one global bucket for all medias:

`local.properties`

```
media.globalSettings.gridFsStorageStrategy.bucketId=myBucket
```

→ Tip

Current tenantId Prefix is Always Added

Keep in mind that even if name of custom bucket is `myBucket`, then prefix with `tenantId` is added automatically, so finally bucket name is `sys_master_myBucket`. The pattern is `sys_<tenantID>_<bucketName>`.

Use More Than One URL Strategy for One MediaFolder

It's possible to bind more than one URL strategy to the MediaFolder by providing list of comma-separated Spring bean ids. By default first strategy from left is used. It's possible to change used strategy by using local session context and by providing the Spring bean id to use for instance for one particular URL. However, `gridfsstorage` extensions do not provide special MongoDB related strategy. You can create your own URL strategy. For more details read Media Storage Overview, section **Creating Custom Media Storage and URL Strategy** For example, your MediaFolder is configured like in the following example:

`local.properties`

```
folder.mongoFiles.url.strategy=localMediaWebURLStrategy,someSpecialURLStrategy,otherUrlStrategy
```

You can change for a moment URL strategy in local session context like in the following code sample:

```
// SessionService is injected by Spring
final SessionService sessionService;

sessionService.executeInLocalView(new SessionExecutionBody()
{
  @Override
  public TranslationResult execute()
  {
    sessionService.setAttribute(MediaManager.PREFERRED_URL_STRATEGY_ID, "someSpecialURLStrategy");
    // render URL here
  }
});
```

Common Configuration Properties

You may configure more than one MediaFolder with the MongoDB GridFS media storage strategy. To not repeat the same properties separately for every MediaFolder, you can use global configuration properties. For example, you have the following MediaFolders: `mongoFiles1`, `mongoFiles2`, and `mongoFiles3` and you want them to work with MongoDB GridFS media storage strategy with the same bucket ID. You can shorten the configuration like in the following example:

`local.properties`

```
media.globalSettings.gridFsStorageStrategy.bucketId=mySpecialBucket

folder.mongoFiles1.storage.strategy=gridFsStorageStrategy
folder.mongoFiles2.storage.strategy=gridFsStorageStrategy
folder.mongoFiles3.storage.strategy=gridFsStorageStrategy
```

To control cleaning MongoDB GridFS storage on fresh initialization of Platform, use following global property:

`local.properties`

```
media.globalSettings.gridFsStorageStrategy.cleanOnInit=true
```

For more information, see Media Storage Overview, section **Cleaning Media Storage on Initialization**.

You can find the list of all available common configuration properties in the `project.properties` file of `gridfsstorage` extension.

Unsecure Media Storage Strategies

Mixing secured and unsecured media in the same container might weaken the security guarantees related to handling secure media.

By default, `bucketId` is derived from the media folder qualifier. If you want to modify this behavior, make sure that the secured folder is mapped to its own `bucket` that is not shared with any other media folder, either secured or unsecured. See an example of a configuration that is **not recommended**:

```
media.folder.PRIVATE.storage.strategy=gridFsStorageStrategy
media.folder.PRIVATE.bucketId=MY_BUCKET
media.folder.PRIVATE.secured=true

media.folder.PUBLIC.storage.strategy=gridFsStorageStrategy
media.folder.PUBLIC.bucketId=MY_BUCKET
media.folder.PUBLIC.secured=false
```

Related Information

http://www.mongodb.org/

Windows Azure Blob Media Storage Strategy

Windows Azure Blob provides a simple web services interface that can be used to store and retrieve any amount of data. You can configure a specific MediaFolder to store binary data of a Media item directly in Windows Azure Blob.

To configure your custom MediaFolder to store binary data of a Media item in Windows Azure Blob you need to provide custom values for set of properties in your **`local.properties`** file. All available properties are listed in the **`project.properties`** file

of **azurecloud** extension.

ℹ Note

Windows Azure Media Storage Strategy Configuration Preconditions

To configure your folder to use Windows Azure Blob you need to have:

- Windows Azure account
- Properly created Access Keys

For more details read http://www.windowsazure.com/en-us/develop/net/how-to-guides/blob-storage/ .

Assume that you want to setup newly created MediaFolder called **azureMedias1** to work with Windows Azure Blob and your container called **test_container**. You need to provide the following configuration:

local.properties

```
media.folder.azureMedias1.storage.strategy=windowsAzureBlobStorageStrategy
media.folder.azureMedias1.connection=DefaultEndpointsProtocol=http;AccountName=yourAccountName;Acc
media.folder.azureMedias1.public.base.url=publicBaseURL
media.folder.azureMedias1.containerAddress=test_container
```

ℹ Note

Find Out More

Keep in mind that **localMediaWebURLStrategy** is used by default for rendering URLs but data stream is still coming directly from Windows Azure Blob thanks to **WindowsAzureBlobStorageStrategy**.

If you would like to use direct URLs to Windows Azure Blob, then you can use another URL strategy called **windowsAzureBlobURLStrategy** that is responsible to render direct URLs to medias. Below you have an example of direct URL to specific Media item:

```
http://hybrisazure.blob.core.windows.net/hybris/sys_master/root/h3e/hd7/8796157378590.jpg
```

ℹ Note

Local File Caching

At the moment Windows Azure Blob storage strategy requires local file caching to work properly, thus folder have to be configured to support caching:

```
media.folder.azureMedias1.local.cache=true
```

or global setting for strategy must be set:

```
media.globalSettings.windowsAzureBlobStorageStrategy.local.cache=true
```

For more details read Media Storage, section **Local File Caching**. If local caching will not be enabled following exception will be thrown during storing attempt:

```
java.lang.IllegalArgumentException: Object size as Long is required to store blobs in Azure Blob
```

Automatic Container Creation

By default Windows Azure strategy provides automatic container creation. Container name is computed from current **tenantId** and from the name of used MediaFolder. For instance if current **tenantId** is **master** and MediaFolder name is **root**, then created container name is **sys-master-root**. If you like to override this behavior for custom folder you can use the following property:

local.properties

```
media.folder.azureMedias1.containerAddress=myContainer
```

To set up one global container for all medias:

local.properties

```
media.globalSettings.windowsAzureBlobStorageStrategy.containerAddress=myContainer
```

→ Tip

Current tenantId Prefix is Always Added

Keep in mind that even if name of custom container is **myContainer**, then prefix with **tenantId** is added automatically, so finally container name is **sys-master-myContainer**. The pattern is **sys-<tenantID>-<containerName>**.

Use More Than One URL Strategy for One MediaFolder

It is possible to bind more than one URL strategy to the MediaFolder by providing list of comma separated Spring bean ids. By default first strategy from left is used. It is possible to change used strategy by using local session context and by providing the Spring bean id to use for instance for one particular URL.

For example your MediaFolder is configured like in the following example:

local.properties

```
media.folder.azureMedias1.url.strategy=localMediaWebURLStrategy,someSpecialURLStrategy,otherUrlStra
```

You can change for a moment URL strategy in local session context like in the following code sample:

```
// SessionService is injected by Spring
final SessionService sessionService;

sessionService.executeInLocalView(new SessionExecutionBody()
{
  @Override
  public TranslationResult execute()
  {
    sessionService.setAttribute(MediaManager.PREFERRED_URL_STRATEGY_ID, "someSpecialURLStrategy");
    // render URL here
  }
});
```

Common Configuration Properties

You may configure more than one MediaFolder with the Windows Azure Blob media storage strategy. To not repeat the same properties separately for every MediaFolder, you can use common configuration properties. For example you have the following

MediaFolders: **azureMedias1**, **azureMedias2** and **azureMedias3** and you want them to work with Windows Azure Blob media storage strategy. You can shorten the configuration like in the following example:

local.properties

```
media.globalSettings.windowsAzureBlobStorageStrategy.connection=DefaultEndpointsProtocol=http;Accou
media.globalSettings.windowsAzureBlobStorageStrategy.public.base.url=publicBaseURL
media.globalSettings.windowsAzureBlobStorageStrategy.containerAddress=myContainer
media.globalSettings.windowsAzureBlobStorageStrategy.local.cache=true

media.folder.azureMedias1.storage.strategy=windowsAzureBlobStorageStrategy
media.folder.azureMedias2.storage.strategy=windowsAzureBlobStorageStrategy
media.folder.azureMedias3.storage.strategy=windowsAzureBlobStorageStrategy
```

To control cleaning Windows Azure storage on fresh initialization use following global property:

local.properties

```
media.globalSettings.windowsAzureBlobStorageStrategy.cleanOnInit=true
```

For more information, see Media Storage Overview, section **Cleaning Media Storage on Initialization**.

You can find the list of all available common configuration properties in the **`project.properties`** file of **azurecloud** extension.

Unsecure Media Storage Strategies

Mixing secured and unsecured media in the same container might weaken the security guarantees related to handling secure media.

By default, the Blob storage's container is identified by the folder qualifier. If you want to change this behavior, make sure that secured folder is mapped to its own container that is not shared with any other media folder, either secured or unsecured. See an example of a configuration that is **not recommended**:

```
media.folder.PRIVATE.storage.strategy=windowsAzureBlobStorageStrategy
media.folder.PRIVATE.containerAddress=MY_CONTAINER
media.folder.PRIVATE.secured=true

media.folder.PUBLIC.storage.strategy=windowsAzureBlobStorageStrategy
media.folder.PUBLIC.containerAddress=MY_CONTAINER
media.folder.PUBLIC.secured=false
```

Related Information

http://www.windowsazure.com

Converting Media with ImageMagick

Create, edit, compose, or convert almost all available bitmap image formats using media conversion and ImageMagick.

Using SAP Commerce media conversion functionality, you can convert medias manually or set conversion of a media as a repetitive process.

To convert media, you need to:

- Create media.

- Define your conversion media formats.
- Create a conversion group and assign to it your conversion media formats.
- Create a media container. When creating a media container, you must:
  - Assign a conversion group to it. This way, you specify which media formats you will obtain after conversion.
  - Assign to the container the media that you want to convert.
- Convert your media.

Working with Media Conversion in Backoffice

The items related to media conversions are available after clicking **Multimedia** in the Backoffice navigation tree.

After you click **Multimedia**, you have access to the following components:

- **Media**
- **Media Folders**
- **Media Containers**
- **Media Formats**
- **Media Contexts**
- **Conversion Group**

To use media conversion, make sure to use the `mediaconversion` and `mediaconversionbackoffice` extensions.

Related Information

mediaconversion Extension
Media
Media Folders

Integrating ImageMagick

SAP Commerce doesn't deliver a prebundled ImageMagick. Install ImageMagick yourself, if necessary.

Prerequisites

To download and install ImageMagick, go to https://imagemagick.org/script/download.php where you can find all installation instructions.

Inside the `project.properties` file in the `mediaconversion` extension, find various ImageMagick specific configuration details, such as the installation directory. Adjust these properties to your requirements in your `local.properties`. Set at least these two properties:

- `imagemagick.bindir`
- `imagemagick.configuration.directory`

Procedure

1. Navigate to `hybris/config` and open the `local.properties` file for editing.

2. Add the `imagemagick.bindir` property with the ImageMagick binaries filepath.

   For Unix systems:

   ```
   imagemagick.bindir=/usr/local/bin/
   ```

   For Windows:

   ```
   imagemagick.bindir=<yourImageMagickFilepath>
   ```

3. Add the `imagemagick.configuration.directory` property to point to the ImageMagick location.

   For example:

   ```
   imagemagick.configuration.directory=<yourImageMagickFilepath>
   ```

Related Information

mediaconversion Extension
Third-Party Compatibility

ImageMagick Configuration

The `mediaconversion` extension enables you to convert bitmap image media with open source software.

> **Note**
> When using ImageMagick, make sure you work on a proper operating system. SAP Commerce only supports operating systems that match both the SAP Commerce and ImageMagick requirements.

The `conversion` attribute of a ConversionMediaFormat using the imagemagickConversionStrategy is interpreted as the command line arguments for the ImageMagick convert command .

There are a few extensions to the conversion string given:

- The **`{input}`** keyword is replaced by the absolute path of the conversion input file name. If the `{input}` marker is not found in the conversion string, the absolute path of the input file is prepended to the conversion command. For example, it is the first parameter passed to the ImageMagick convert command .
- The **`{output}`** keyword is replaced by the absolute path of the output file just as the `{input}` marker. If no **`{output}`** marker is found in the conversion string, the absolute path of output file is appended to the conversion string, for example as the last command line parameter.
- The **`{addOn#`*n*`}`** keyword is replaced the absolute path of the n^{th} media in the list of additional medias associated to the ConversionMediaFormat. This feature makes it possible to include additional (static) input files in the conversion like adding a watermaker, applying color profiles, or putting an image onto a custom background.

For more information, refer to the ImageMagick documentation and the ImageMagick extensive library of recipes .

ImageMagick Configuration

Key	Value Range	Default	Description
`imagemagick.bindir`	String	embedded imagemagick binaries directory	The binary directory of your local ImageMagick installation.

Key	Value Range	Default	Description
`imagemagick.configuration.directory`	String	MAGICK_CONFIGURE_PATH	Path to the ImageMagick configuration directory to use. Changes to this property are not reflected on runtime.
`imagemagick.executable.convert`	String	convert	The executable name to be called to run the convert command (in the given `imagemagick.bindir`). This is an advanced configuration option.
`imagemagick.executable.identify`	String	identify	The executable name to be called to run the identify command (in the given `imagemagick.bindir`). This is an advanced configuration option.

ImageMagick Environment Variables

All documented ImageMagick environment variables are routed through when set in the **JVM** environment. With this you can configured a variety of ImageMagick options. The SAP Commerce system sets only two of these environment variables as follows:

- MAGICK_TEMPORARY_PATH

  The environment variable is sequenced accordingly:

  - If the environment variable is set in the JVM environment, it is used.
  - If the `tmpDir` property of the imagemagickConversionStrategy Spring bean is set, it is used.
  - If the property ***HYBRIS_TEMP_DIR*** system property is set, the ImageMagick temporary directory is set to ***$HYBRIS_TEMP_DIR*** `/convert`.
  - Otherwise the temporary directory is set to a `/convert` directory in the JVM default temporary directory (`$java.io.tmpdir/convert`).

- MAGICK_CONFIGURE_PATH

  To evaluate the correct ImageMagcik configuration folder to use, the following lookup sequence is applied:

  - If the configuration directory was set through Spring injection, it is used (the property value is disobeyed).
  - If the `imagemagick.configuration.directory` property is set, this path is used.
  - If the JVM ***MAGICK_CONFIGURE_PATH*** environment variable is set, it is used.

MimeMapping Configuration

ImageMagick relies on Mime Type detection on file extensions. This means to convert a media to a specific Mime Type, the file extension of the output file must be set accordingly. The imagemagickService uses the injected MimeMappingStrategy to resolve file extensions and Mime Types for a given file. TheDefaultMimeMappingStrategy is backed by the custom (file) extension properties set in the SAP Commerce project properties. This makes it easy to extend the Mime Type/file extension mapping even at runtime.

To introduce a new Mime Type/file extension pair, add the following properties to your`local.properties` file:

```
media.customextension.${mimetype category}.${mimetypename}=${extension}
mediatype.by.fileextension.${file.extension}=${mimetype}
```

For example:

```
media.customextension.application.excel=xls
mediatype.by.fileextension.xls=application/excel
```

Preventing Vulnerable Operations

Some operations that ImageMagick offers are vulnerable. To secure SAP Commerce, a comma-separated blocklist and a comma-separated allowlist are defined to prevent those vulnerable operations.

The blocklist definitions is:

```
imagemagick.executable.convert.commands.blacklist=write,clip-mask,mask,...
```

The allowlist definition is:

```
imagemagick.executable.convert.commands.whitelist=resize,crop,compose,...
```

You can't use both of those lists at the same time. To switch between them, use a switcher:

```
imagemagick.executable.convert.commands.validation.type=whitelist
```

Related Information

Integrating ImageMagick
http://www.imagemagick.org

Conversion Media Formats - Sample Data

In the SAP `mediaconversion` extension you are enable to get the implemented sample data of ConversionMediaFormat. It gives you a set of examples what media formats you can get after making conversion of medias.

To get the sample data, when you initialize the Platform in the SAP Administration Console, select `true` from the **Create sample data** drop-down list in the **mediaconversion** setting.

The Default Conversion Media Formats

The following table presents the samples of the ConversionMediaFormat provided in the `mediaconversion` extension:

Format	Conversion Command	Input Format	Description	Example
Default-WorkingFormat	`-resize 1000x> -colorspace RGB`		Working format which resizes image to 1000 pixels width and converts to color space RGB.	
Default-Hitlist	`-resize 40x40`	Default-WorkingFormat	Conversion format resizes image to 40x40 pixels, for example searching hitlists.	
Default-Thumbnail	`-resize 80x80`	Default-WorkingFormat	Conversion format with resizing image to 80x80 pixels, for thumbnail views.	
Default-Detail	`-resize 800x>`	Default-WorkingFormat	Conversion format resizes image to 800 pixels width for detailed view, for example on rollover of an image.	
Default-Preview	`-resize 160x160`	Default-WorkingFormat	Conversion Format resizes image to 160x160 pixels, for example used as backend previews.	
Default-Print	`-colorspace CMYK`		Conversion format for Print Output converts to color space CMYK.	
Default-Watermark	`{addOn#1} -compose bumpmap -gravity center -composite`	Default-WorkingFormat	Conversion format that adds a watermark to the image: 	

Format	Conversion Command	Input Format	Description	Example
Default-Crazy	`{addOn#1} {input} [150x150] -compose bumpmap -gravity center -composite`	Default-WorkingFormat	Conversion format that desaturates the image and merges with another image: 	
Default-TextOverlay	`-font Arial -pointsize 25 -draw "gravity south fill black text 0,12 'Copyright by SAP' fill white text 1,11 'Copyright by SAP'"`	Default-WorkingFormat	Conversion format that adds rendered text to the image: Copyright by SAP	

Adding Watermarks on Images on in the Public Cloud

When it comes to adding watermarks on images in in the Public Cloud, keywords like {addOn#n} are not supported. Using the command in "Default-TextOverlay" can lead to errors and the system can't convert the images.

Instead of using the **-draw** command, use **-annotate** to add the textoverlay watermark. For example:

```
-font DejaVu-Sans-Bold -fill yellow -pointsize 24 -gravity Center -annotate 0 testmessage
```

However, because `imagemagick.executable.convert.commands.whitelist` does not include the `annotate` command, as defined in `/hybris/bin/modules/platform/mediaconversion/project.properties`), you need to redefine it in the `local.properties` file or `manifest.json` file.

The **imageprocessing** pod supports the following fonts. If you use other fonts, the foreign fonts can generate errors and the system can't convert the media.

- DejaVu-Sans
- DejaVu-Sans-Bold
- DejaVu-Sans-Mono
- DejaVu-Serif
- DejaVu-Serif-Bold
- Liberation-Mono
- Liberation-Mono-Bold
- Liberation-Mono-Bold-Italic
- Liberation-Mono-Italic
- Liberation-Sans

- Liberation-Sans-Italic
- Liberation-Sans-Bold
- Liberation-Sans-Bold-Italic
- Liberation-Serif
- Liberation-Serif-Bold
- Liberation-Serif-Bold-Italic

Process Execution Environment

As the `mediaconversion` extension spawns operation system processes to execute ImageMagick, special care is taken to have an efficient mechanism to do so. The `mediaconversion` extension provides a process execution environment for best performance and resource usage, specifically on Unix-based server applications.

Instead of forking a new process directly from the virtual machine server, which can cause high-memory consumption, it spawns an additional Java virtual machine and communicates over RMI with it.

API

The ProcessExecutor interface exposed in the Spring context is similar to the parts of java.lang.Runtime to fork processes. Instead of accessing the output stream (stdout, stderr) directly, they can be accessed on a (character) line by line basis by providing implemnentations of the Drain interface.

ℹ Note

There is neither support for binary output nor support for piping data to the forked process.

Configuration Options

The behavior of the process execution environment can be adjusted by the following properties, for example via `local.properties`:

Property	Default Value	Description
os.rmiregistry.port	2198	Port of the local RMI registry.
os.processexecutor.limit	10	The maximum amount of subprocesses to be spawned in parallel. If this value is zero or negative the amount of subprocesses is unlimited. Changes to this property is reflected in runtime.
os.rmi.processexecutor.classpath	***${HYBRIS_BIN_DIR}*** `/custom/os/bin/osserver.jar` ***${HYBRIS_BIN_DIR}*** `/custom/os/classes/`	Classpath used to launch the Process Executor RMI server.
os.rmi.processexecutor.java	***${java.home}*** `/bin/java`	The Java Virtual Machine (executable) to use.

Property	Default Value	Description
os.rmi.processexecutor.javaopts		Comma separated list of Java VM options to use when spawning the Process Executor RMI server.
os.processexecutor.windows.amd64 os.processexecutor.linux.amd64 os.processexecutor.mac_os_x.amd64	rmi	Mapping of process executor (embedded | rmi) by operating system. The `mediaconversion` extension uses string distance matching to find the best match to ***${os.name}*** and ***${os.arch}***. If embedded is used, there is no separate Virtual Machine and RMI communication not recommended.
os.rmi.loopback.address	IPv4 environment: 127.0.0.1 IPv6 environment: ::1	Communication with the embedded RMI server is restricted to the local loopback device for security reasons. This is the loopback address to use.

Related Information

ImageMagick - Integration Guide
http://www.imagemagick.org

Creating a Media Item

Backoffice allows you to create a media item by uploading an existing media file into a SAP Commerce directory.

Context

To upload a media stored in your local system into a SAP Commerce directory, log in to Backoffice and follow these steps:

Procedure

1. In the Backoffice navigation tree, click **Multimedia** > **Media**.

   A collection browser with a list of available media items opens up.

   Notice the + icon. It opens up the **Create New Media** wizard.

2. Click + to open up the **Create New Media** wizard.

   The wizard opens on the **Essentials** tab.

3. In the **Essentials** tab, provide an identifier for your media and choose the catalog version to which you want to assign the media. Click **Next** to move to the **Content** tab.

4. Upload your media by using one of the functionalities available in the **Content** tab.

5. Click **Finish** to close the wizard.

Results

You have uploaded your media file into the SAP Commerce system.

You can search for your media now and edit its attributes and properties.

Defining a Conversion Media Format

SAP Commerce enables you to define the required media format parameters, such as size, color space, and watermarks as conversion media formats that you can use to convert your media item.

Context

You need at least one conversion media format to convert your media item. After you define a conversion media format, you associate the media item that you want to convert to the conversion media format.

To define a new conversion media format, log in to Backoffice and follow these steps:

Procedure

1. In the Backoffice navigation tree, click **Multimedia** > **Media Formats**.

   A collection browser opens up with the **Media Format** drop-down menu from which you can open the **Conversion Media Format** wizard.

2. Navigate through the **Media Format** drop-down menu and click **Conversion Media Format**.

   The **Conversion Media Format** wizard opens.

3. Complete the **Qualifier** and **Name** fields in the **Conversion Media Format** wizard and click **Done**.

   The wizard closed. You created your conversion media format item.

4. Look up your conversion media format item in the **Media Formats** collection browser and click it.

   An editor area with information about your conversion media format opens.

   You can switch between the **COMMONS**, **CONVERSION**, and **ADMINISTRATION** tabs for more information.

5. Switch to the **CONVERSION** tab and complete the **Conversion command** field with a command of your choice. Click **SAVE**.

   Your conversion command should be one of the ImageMagick convert commands.

   We use the `-flop` command that flops image in the horizontal direction.

Results

You created a conversion media format. You can now assign your conversion media format to a conversion group.

Creating a Conversion Group

In the SAP Commerce media conversion, you can create and manage conversion groups that are containers for different conversion media formats, for example, different media asset types.

Context

To convert media, you need to create a conversion group. Your conversion group may contain several conversion media formats. When you convert a media item, the conversion mechanism uses all the conversion media formats from your conversion group.

These instructions show how to create a conversion group and then associate your conversion media formats with it.

To create a conversion group, log in to Backoffice and follow these steps:

Procedure

1. In the Backoffice navigation tree, select Multimedia > Conversion Group.

   The Conversion Group collection browser appears.

2. Select + to open the Create New Conversion Group wizard.
3. Complete the Identifier field and select Finish.

   You have created your conversion group. Now you can associate it with your media conversion format.

4. Look up your conversion group in the Conversion Group collection browser and click it.

   A window with information about your conversion group displays.

   You can switch between the Commons and Administration tabs for more information.

   In the Commons tab, under Properties, there is the Supported Formats field through which you can associate your conversion group with chosen media conversion formats.

5. Select ••• for the Supported Formats fields.

   A dialog box prompts you to look up your conversion media format.

6. Choose your conversion media format and click Select.

   The window closes and you are back in the Conversion Group collection browser.

7. Select Save to save the changes in your conversion group.

Results

You created a conversion group and assigned your media conversion format to it. When you look up your conversion group, your media conversion format appears in the list in the Supported Formats field.

Creating a Media Container

In SAP Commerce media conversion, a media container is used as a business representation of a digital asset containing all the required media derivatives and one source (master) media.

Context

Creating a media container is a necessary step in the process of converting media. When you create a media container, you also need to assign a media and a conversion group to it. When you assign a conversion group to a media container, you define specific conversion media formats that should be processed while converting a media.

To create a media container, log in to Backoffice and follow these steps:

Procedure

1. In the Backoffice navigation tree, click Multimedia > Media Containers.

   The Media Containers collection browser opens. Notice the "+" icon that opens the Create New Media Container wizard.

2. Select + to open the Create New Media Container wizard.
3. Complete the Qualifier and Catalog version fields. Click Finish.

   You created a media container. You can now assign a conversion group and a media to the media container.

4. Assign your conversion group and your media to your media container.
    a. Look up your media container and click it to display information about it.

       You can notice that the **Conversion Group** and **Medias** fields don't contain anything. As a result, the **Convert Missing Media** function available under the **Media** filed is inactive.

    b. Select ••• in the **Conversion Group** field to search for your conversion group.

       A search window opens up.

    c. Select your conversion group and click **Select**.

       The search window closes.

    d. Perform similar steps to assign your media to the container.

    e. Select **Save** to save the changes you made to your container, and reload the view.

Results

You created a media container, and assigned your media and conversion group to it. Now the **Convert Missing Media** function is active and you can convert your media.

Converting Media

Convert your media in SAP Commerce.

Prerequisites

Before you start the procedure, you must have the following items ready:

- A media file uploaded into the SAP Commerce system
- A defined conversion media format
- A conversion group with the conversion media format assigned to it
- A media container with both the media and the assigned conversion group

Procedure

1. Log in to Backoffice.
2. In the Backoffice navigation tree, select **Multimedia** > **Media Containers**.

   A list with available containers opens.

3. Look up your container and select it to open its editor.

   You can see that the **Convert Missing Media** button is active:

4. Select the **Convert Missing Media** button.

   You converted your media. You can see that there are now two media items available in the **Media** fields. One of them is the original and the other is your converted media:

5. **Optional:** Select the name of your converted media to see the result.

Results

You converted your media.

Patch Concepts and Usage

The Patches feature is based on the concepts of a setup class, an organization unit, a patch, and an action. There are various configuration options for enabling the Patch functionality.

The topics include:

Setup Classes

Patches use the concept of the setup classes.

The default `de.hybris.platform.commerceservices.setup.AbstractSystemSetup` class offers methods to:

- generate options for the **Initialization** or **Update** tabs in Administration Console
- perform an import of the essential data
- perform an import of the project data
- display in Administration Console information or errors during an initialization or an update
- use some additional utility methods

In standard projects, these classes are extended in different extensions with a project-specific logic. The Patches feature provides an extended idea of the `AbstractSystemSetup` class in the `de.hybris.platform.patches.AbstractPatchesSystemSetup` class. Besides new functionalities, `AbstractPatchesSystemSetup` also takes care of splitting concerns. The logic related to importing data or performing SQL updates is located in Patch classes. A Setup class itself is responsible for high level management of what should happen during an initialization or an update process.

`AbstractPatchesSystemSetup` enables you to:

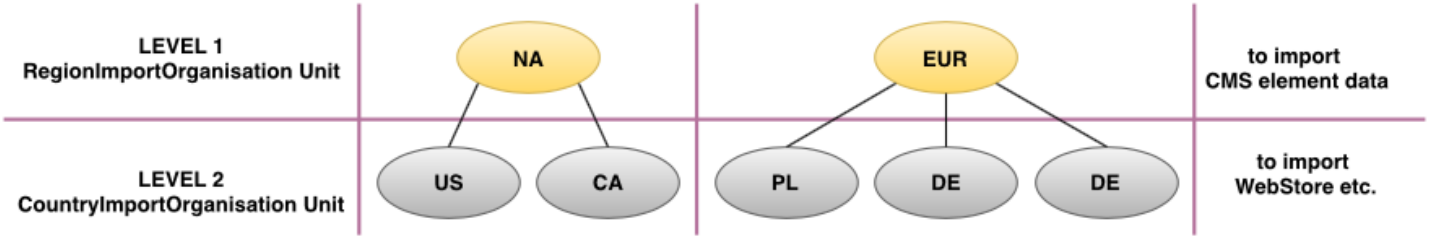
- generate options in Administration Console for initialization and update. While generating these options, a previous execution is taken into consideration (some Patches should be executed only once), and project properties.
- perform an import of the essential data (based on the previous execution and project properties for Administration Console and ant)
- perform import of the project data (based on the previous execution and project properties for Administration Console and ant)
- pass a reference to Administration Console (`SystemSetupContext`) to a dedicated utility class (`JspContextHolder`). You can now ask the utility class for this context from any place in code. It allows you to print out information related to initialization or update processes in Administration Console in a much more convenient way.

With this new approach, a Setup class should not be used for uploading direct ImpExes.

Organisation Units

An Organization Unit allows you to reflect a customer-specific structure related to data.

The most common case for Organisation Units is to use them for shops. Usually, a customer project requires more than one WebStore or BaseSite. Using Organisation Units in a proper way with patches reduces data (ImpEx files) duplication and maintenance effort. In case of more complex projects, you may require more levels of Organisation Units. You may group shops by other Organisation Units (Regions) that share the same CMS content.



Based on the example above, in a customer-specific code the `ImportOrganisationUnit` interface delivered with Patches should be extended with enumerations that reflect different levels for which data import is required. Depending on the complexity, there may be one or more enumerations required. The values of these enumerations create a tree structure (values of different relation create a parent-child relation).

Patch

A Patch is a Java class that you execute during a system initialization or an update. It defines a list of actions and a list of `ImportOrganisationUnits` for which you execute those actions.

There are two basic actions available:

- importing ImpEx files
- performing direct SQL commands

You can execute actions in two types of contexts:

- in the global context - for data that is independent of the Organization Units, where you should import to the system only once. The examples of data that fits in here may be Solr Server Configuration (usually one per project), new currency, or new enumeration values.
- in one of the Organization Units contexts - for example WebStore and BaseSite instances must be created for each store, therefore this data should be imported in the context of the Store Organization Unit

The best practice for using Patches is to create a new patch each time a new piece of code is delivered for production and some action is required.

EXAMPLES

At the end of Release 1, Patch1x0 should be delivered. Once it is run on PROD, it shouldn't be changed anymore. It is expected that this Patch will not be performed anymore on PROD, since any changes in this area will only make test environment different from PROD.

In case of issues, a new Patch1x1 should be introduced that contains actions to help solve a given problem. If there is a fix for the problem and it doesn't require any actions during an update (for example a fix in Java code), then the new Patch doesn't have to be introduced.

For a new release, a new Patch2x0 should be introduced.

These cases are just an example. You may have a different case: a given project can have different naming convention; maybe Patches have to be delivered for every sprint; previous patches may be executed with new Organizations.

Rerunnable patches

Apart from standard Patches that should be executed only once, there are Patches dedicated to Actions that can be performed multiple times. Consider a Patch that only performs a synchronization of CMS content catalog versions (Staged to Online). Such a Patch should usually be the last in the list (so it performs the synchronization after all data is loaded to the Staged catalog). Running this Patch more than once shouldn't cause any data issues. Moreover, it may be useful to reuse the same Patch after each update to synchronize the data.

Executing Patches in this way multiple times is possible without any manual actions thanks to the `Rerunnable` interface. It is sufficient that a given Patch implements this interface and the mechanism will allow to run it regardless of how many times it was executed in the past, and regardless of how the flag for re-running patches was set.

Actions

Patches use Actions classes to perform the actual work.

All Actions have to implement the `PatchAction` interface that introduces the `perform(PatchActionData)` method. Thanks to the common interface, it was possible to design a generic solution for the logging and tracking mechanism.

PatchActionData

`PatchActionData` is a data object that helps with transferring all required data from a Patch to an Action.

`PatchAction` uses these `PatchActionData` attributes:

- String `name` contains a human friendly description of an action
- Map<PatchActionDataOption, Object> `options` is a Map containing all additional attributes - the keys are defined in `PatchActionDataOptions`, the values depend on usage
- Patch `patch` references a `Patch` in which an action was defined

PatchActionDataOptions

The following are the Enums used in specific actions.

ImpEx:

- `IMPORT_OPTIONS` - a key for changing a default ImpEx import configuration (legacy mode, single thread)
- `FILE_NAME` - a key under which the file name that should be imported is stored
- `HEADER_OPTIONS` - a key for additional options (Strings) that should be merged to a file stream on the top of the file; usually used to define ImpEx macros
- `RUN_AGAIN` - a key for letting the mechanism know whether this is the first run of a given ImpEx option or whether it is just an update of new languages; in the second case, the main ImpEx is not loaded, and only language specific files are imported
- `IMPORT_LANGUAGES` - a key for a list containing languages that the mechanism uses to search for language-specific data files (ImpExes)

SQL:

- `QUERY` - a key under which an SQL query is stored

Error Handling for Patch Actions

While you are executing Patch Actions, it is possible that errors will show up. Further running of the initialization or update processes depends on the error type. Exceptions such as a missing ImpEx file, invalid query/impex syntax, model saving problems don't have any impact on the continuation of the execution. Major errors stop the whole initialization or update process. It means that current Patch won't be continued and everything that should be executed after this Patch (next Patch, creating project data for other extensions, and others) won't be done and the whole process will stop.

PatchActionException

`PatchActionException` is a default Exception used for errors related to executing Actions except `ImportPatchAction`. This kind of exception doesn't have any impact on further import. It must be thrown from Action, where earlier it can be logged to Console as a specific message without a stack trace. This Exception is caught in `PatchExecutionUnitAspect` where it is logged and tracked. It can be later extended, so specific Actions will have specific Exceptions that will extend `PatchActionException`.

Stopping Import Execution

If there are major exceptions while importing data, such as NPE, Data Base Connection problems, the Platform mechanism catches these errors and stops the whole initialization or update process.

Catch them before the Platform functionality starts running to be able to add your specific logging in Administration Console/Console, and to track the current exception under `PatchExecution: errorLog` field, if it is possible.

Caught exceptions are saved to the database.

Current PatchExecutionUnit

When a major exception is thrown during execution of an ImpEx import or specific action, the status and errorLog for `PatchExecutionUnit` isn't updated. Therefore, the user sees information for such `PatchExecutionUnit`, next to the UNKNOWN status, and should check errorLog of `PatchExecution`.

Tracking and Logging Patches When Errors Appear

ImpExException in Importing ImpEx

✅ Importing is continued

✅ The whole execution of the process after the current Patch is continued

	executionStatus	errorLog	Class where tracking is handled	Class where exception is logged
`PatchExecutionUnit`	ERROR	✅	`PatchExecutionUnitAspect`	`ImportPatchAction`
`PatchExecution`	ERROR	⛔	`AbstractPatchesSystemSetup` after `createProjectData`	

PatchActionException for Action Execution

✅ Importing is continued

✅ The whole execution of the process after the current Patch is continued

	executionStatus	errorLog	Class where tracking is handled	Class where exception is logged
`PatchExecutionUnit`	ERROR	✅	`PatchExecutionUnitAspect`	specific Action class
`PatchExecution`	ERROR	⛔		

Major exception for all Action types

⛔ Importing is continued

⛔ The whole execution of the process after the current Patch is continued

	executionStatus	errorLog	Class where tracking is handled	Class where exception is logged
`PatchExecutionUnit`	UNKNOWN	⛔		
`PatchExecution`	ERROR	✅	`AbstractPatchesSystemSetup` in `executePatches()`	`AbstractPatchesSystemSetup`

Enabling Patches

There are various configuration options for enabling the rerunnable Patch functionality as well as for preselecting and hiding Patches in different scenarios.

Rerunnable Patches

A configuration for enabling the option to rerun all the Patches that were already executed. In Administration Console there are two options available per Patch, both for an update and initialization. Possible values are `true` or `false`.

For the initialization process type, the default value is always `true`, but it can be changed - the property for rerunning is not taken under consideration for an initialization.

For the update process type, `false` is set if a Patch was executed, but it can be changed if the property for rerunning Patches is set to `true`.

In a DEV scenario, the option to rerun Patches should be available. To enable it, add the configuration in the `properties` file:

```
patches.allow.rerun.patches=true
```

In a PROD scenario, Patches cannot be executed again by default (no changes in `properties` file are needed).

Pre-selected Patches

A configuration for each Patch, used for:

- pre-selecting patches when accessing Administration Console
- executing Patches in an `ant initialize` / `ant updatesystem` script

To exclude a specified Patch, add the configuration to the `properties` file:

```
patches.<patchId>.notSelected=true
```

When the property is not specified in the update mode for an already executed re-runnable Patch, the Patch isn't executed.

In a DEV scenario, all Patches are pre-selected by default. If you want to exclude a few Patches, for each Patch explicitly add the `notSelected` property to the `properties` file.

In a PROD scenario, only Patches that were not executed are pre-selected by default.

Hidden Patches

A configuration for each Patch determining whether a specified Patch should be hidden - not visible in Administration Console.

The first configuration taken into consideration is a configuration per Patch:

```
patches.<patchId>.hide=true/false
```

If a configuration per Patch is not defined, the value of the property for all Patches is taken into consideration:

```
patches.hideAll=true/false
```

If neither of the properties is specified, all Patches are visible.

Configuration Setting	**patches.<patchId>.hide=true**	**patches.<patchId>.hide=false**	**patches.<patchId>.hide** not defined
`patches.hideAll=true`	not visible	visible	not visible
`patches.hideAll=false`	not visible	visible	visible

In a DEV scenario, all Patches are visible by default (no changes in the `properties` file are needed).

In a PROD scenario, all Patches should be hidden by default, so a configuration in the `properties` file is needed:

```
patches.hideAll=true
```

It is possible to show a chosen Patch:

```
patches.<patchId>.hide=false
```

Pre-selected vs Hidden

Patches not executed	**patches.<patchId>.notSelected=true**	**patches.<patchId>.notSelected=false**
`patches.<patchId>.hide=true`	A Patch won't be executed/selected and won't be visible in Administration Console	A Patch will be executed/selected and won't be visible in Administration Console
`patches.<patchId>.hide=false`	A Patch won't be executed/selected and will be visible in Administration Console	A Patch will be executed/selected and will be visible in Administration Console

If none of the above properties is specified in the `properties` file, then a Patch will be executed and will be visible in Administration Console.

Non-executed Patches Selection

	patches.<patchId>.notSelected=true	**patches.<patchId>.notSelected=false** or **property not specified**
Update and the rerunnable functionality enabled	Patch won't be executed/selected. Default value: `false` Possible values: `true/false`	Patch will be executed. Default value: `true` Possible values: `true/false`
Update and the rerunnable functionality disabled	Patch won't be executed. Default value: `false` possible values: `true/false`	Patch will be executed. Default value: true Possible values: `true/false`
Initialization	Patch won't be executed. Default value: `false` Possible values: `true/false`	Patch will be executed. Default value: `true` Possible values: `true/false`

In case of not executed patches, the `patches.allow.rerun.patches` flag is ignored and the system behaves the same way regardless of the scenario. The only difference is that when `patches.<patchId>.notSelected=true`, the default value is set to `false`, however both options are available.

Executed Patches Selection

The `notSelected` value doesn't have impact on the `rerun` property - the executed Patch won't be executed during `ant updatesystem`.

	patches.<patchId>.notSelected=true	**patches.<patchId>.notSelected=false**
Update and the rerunnable functionality enabled	Patch won't be executed/selected. Default value: `false` Possible values: `true/false`	Patch won't be executed in Administration Console even if the `notSelected` property is set to `false`. It must be

	patches.<patchId>.notSelected=true	patches.<patchId>.notSelected=false
		manually changed in Administration Console For `ant updatesystem`, it will be executed and for a hidden Patch in Administration Console. Default value: `false` Possible values: `true/false`
Update and the rerunnable functionality disabled	Patch won't be executed Default value: `false` Possible values: `false`	Patch won't be executed. Default value: `false` Possible values: `false`
Initialization	Patch won't be executed. Default value: `false` Possible values: `true/false`	Patch will be executed. Default value: `true` Possible values: `true/false`

The initialization behaves the same way whether a Patch was already executed or not. The `patches.allow.rerun.patches` attribute is ignored in case of initialization.

In all cases of an update, when a Patch was already executed, the default option is `false`. `true` is only available for `patches.allow.rerun.patches=true`.

Logging and Tracking

You can track Patches, and log information about them.

Logging

The Patches logging mechanism is responsible for displaying proper information for Administration Console (if an initialization/update has been started from Administration Console), and logs.

The information that appears there has to be detailed enough to support the development process by presenting problems in a detailed and a user-friendly way.

There is a mechanism to take care of proper log levels, for example when a file is not found, the mechanism must take into consideration whether this was a mandatory or an optional file, to display the message in proper way.

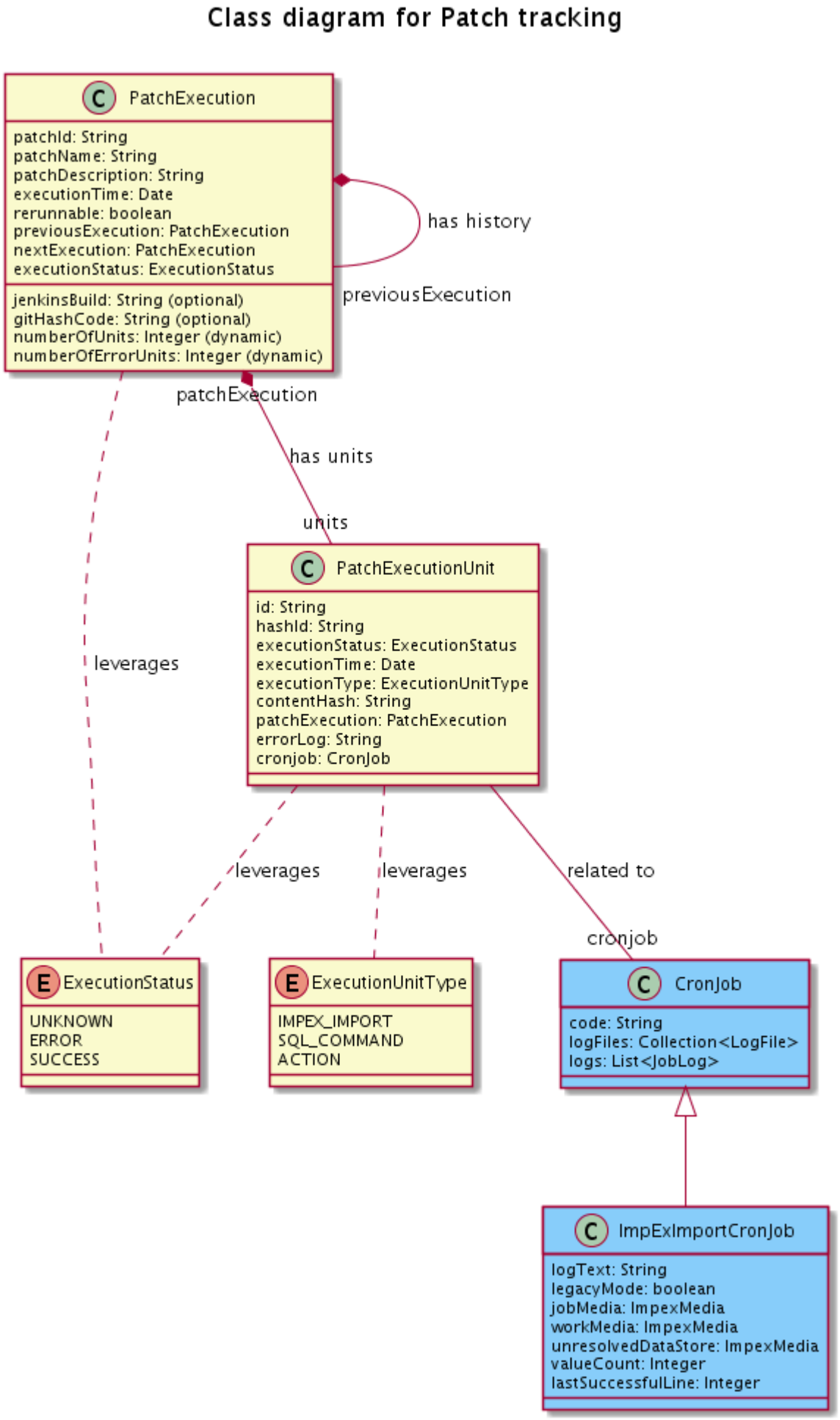
```
Patch 01_00::Updating project data
Executing SQL statement: update dbo.basestore set p_uid = 'electronics3' where pk = 8796093122525 ...
Executing SQL statement: update dbo.basestore set p_uid = 'electronics3' where pk = 8796093122525 failed
Creating global data for de_DE, fr_FR
Executing import: r01_00_010-globalDataExample.impex ...
Executing import: r01_00_020-globalDataExample.impex ...
Executing import: r01_00_030-globalDataExample.impex ...
Creating shop specific data for Europe
Executing import: r01_00_010-shopDataExample.impex ...
Executing import: r01_00_020-shopDataExample.impex ...
Executing import: r01_00_030-shopDataExample.impex ...
Executing import: r01_00_040-shopDataExample.impex ...
Executing import: r01_00_050-shopDataExample.impex ...
Executing import: r01_00_060-shopDataExample.impex ...
Creating country specific data for France
Executing import: r01_00_010-countryDataExample.impex ...
Executing import: r01_00_020-countryDataExample.impex ...
Executing import: r01_00_030-countryDataExample.impex ...
Executing import: r01_00_040-countryDataExample.impex ...
Creating country specific data for Germany
Executing import: r01_00_010-countryDataExample.impex ...
Executing import: r01_00_020-countryDataExample.impex ...
Executing import: r01_00_030-countryDataExample.impex ...
Executing import: r01_00_040-countryDataExample.impex ...
Patch 02_00::Updating project data
Creating global data for en_CA, en_US, fr_CA
Executing import: r01_00_010-globalDataExample.impex ...
Executing import: r01_00_020-globalDataExample.impex ...
Executing import: r01_00_030-globalDataExample.impex ...
Creating shop specific data for North America
Executing import: r01_00_010-shopDataExample.impex ...
Executing import: r01_00_020-shopDataExample.impex ...
Executing import: r01_00_030-shopDataExample.impex ...
Executing import: r01_00_040-shopDataExample.impex ...
Executing import: r01_00_050-shopDataExample.impex ...
Executing import: r01_00_060-shopDataExample.impex ...
Creating country specific data for Canada
Executing import: r01_00_010-countryDataExample.impex ...
Executing import: r01_00_010-countryDataExample.impex failed
Executing import: r01_00_020-countryDataExample.impex ...
Executing import: r01_00_020-countryDataExample.impex failed
Executing import: r01_00_030-countryDataExample.impex ...
Executing import: r01_00_030-countryDataExample.impex failed
Executing import: r01_00_040-countryDataExample.impex ...
Executing import: r01_00_040-countryDataExample.impex failed
Creating country specific data for United States
Executing import: r01_00_010-countryDataExample.impex ...
```

Tracking

The Patches tracking mechanism is responsible for storing in the database the information about Patches execution. There are several main reasons to do so:

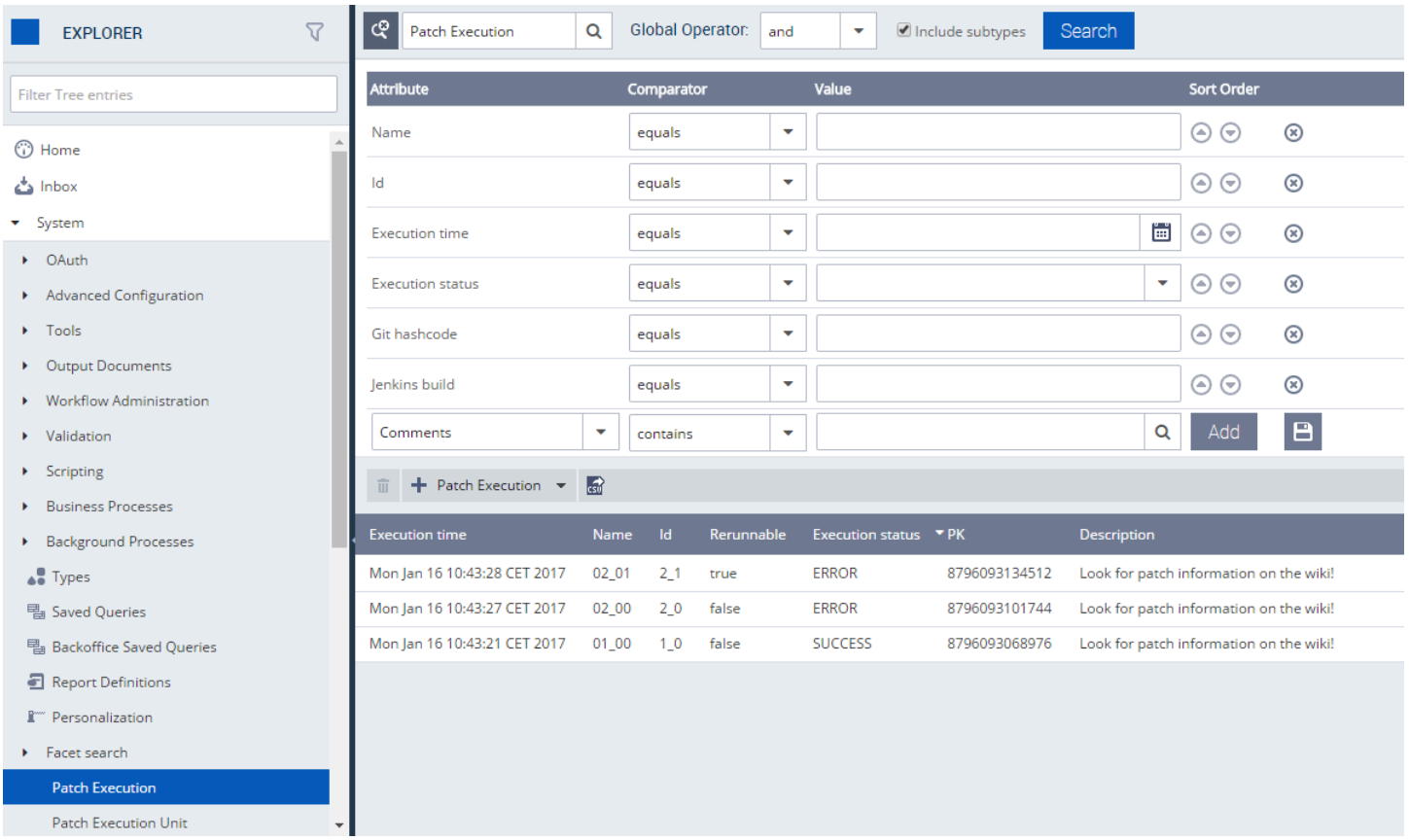
- to know when a given Patch was already executed
- to know from which Actions this patch consists of

- to track in the database the information about Patch execution and what kind of issues appears during that process. It allows you to verify data issues by checking previous data import execution statuses.

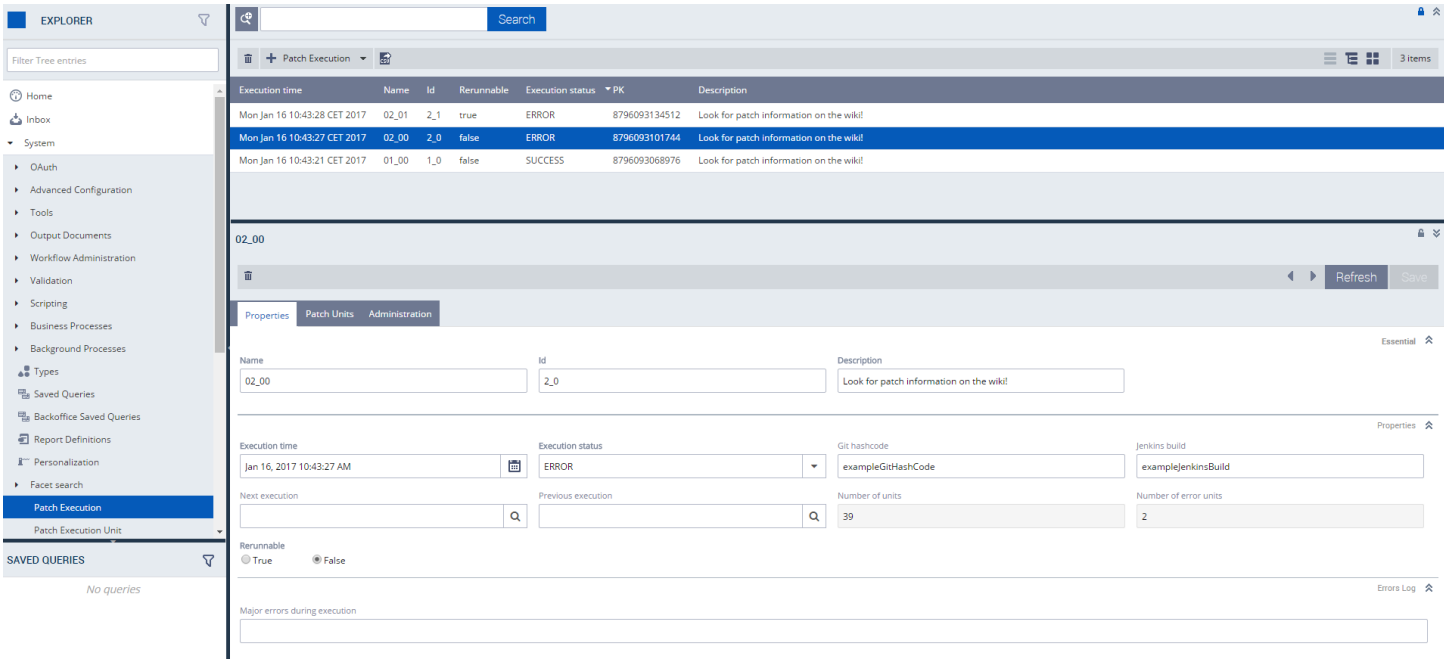


Backoffice Views

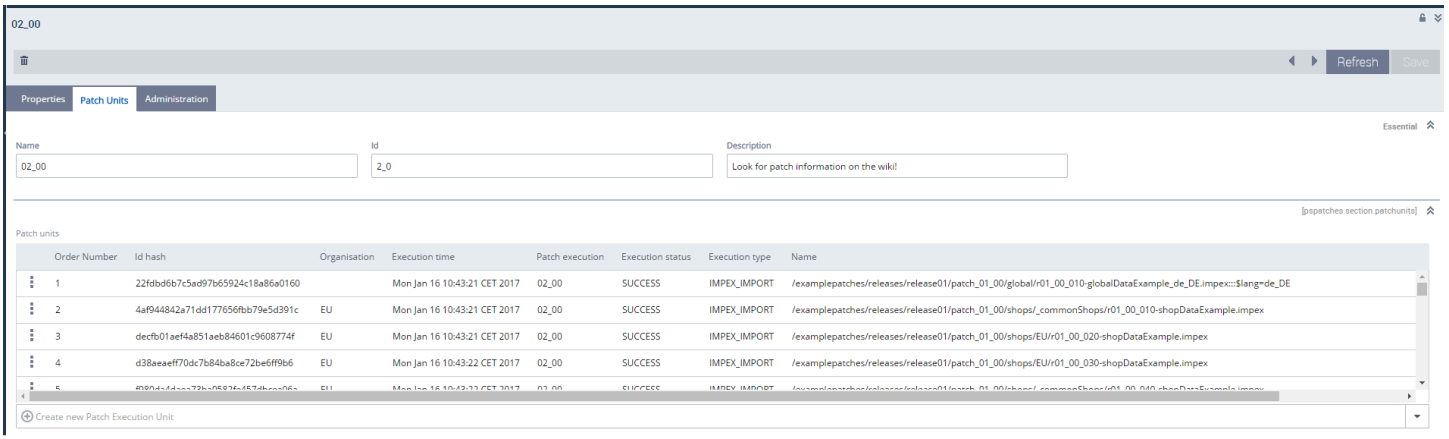
Patch Execution View



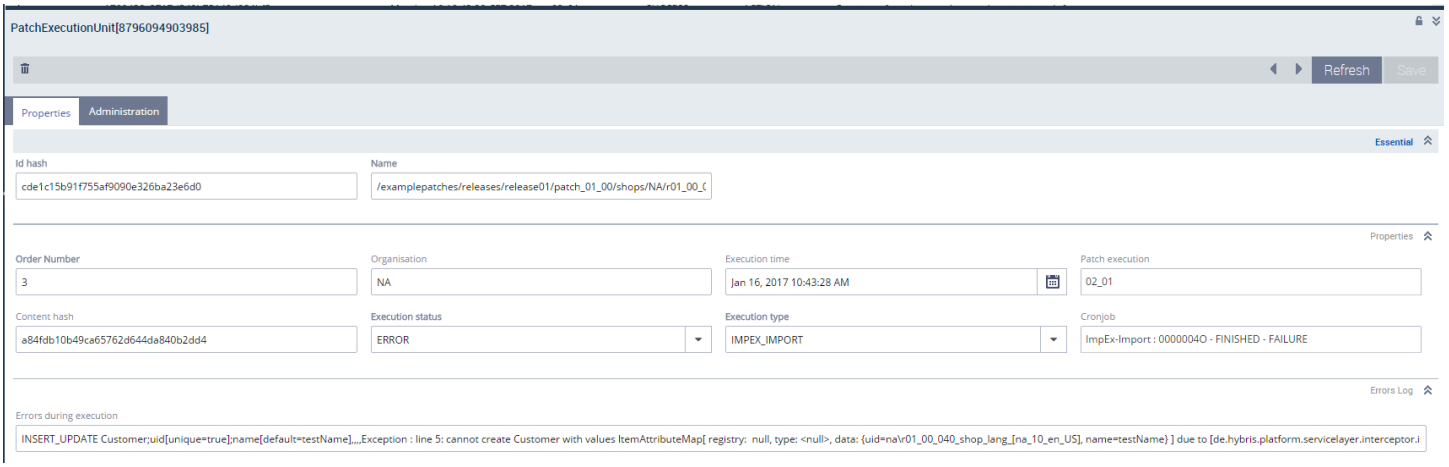
Patch Execution Properties



Patch Execution Units



Patch Execution Unit Properties



Patches Project Properties

Find a list of project properties for Patches.

Property Name	Description	Possible Values
`patches.allow.rerun.patches`	Allows you to run (rerun) patches multiple times; the property can be handy during a development. It is a global switch - you can use it for all Patches.	`true` or `false`
`patches.<patchId>.notSelected`	Allows you to switch off a given Patch. Switching a patch off means that: • during an update or initialization from ant, a given patch is ignored • in Administration Console, the option for a given Patch is by default set to prevent the Patch from being imported; you can still change it manually	`true` or `false`

Property Name	Description	Possible Values
`patches.<patchId>.hidden`	Allows you to hide a given patch from Administration Console.	`true` or `false`
`patches.hideAll`	Allows you to hide all patches from Administration Console.	`true` or `false`
`impex.importConfig.cronJob.removeOnSuccess.enabled`	Allows you to remove a successfully imported `ImportCronJob`. Cron jobs for ImpEx executed with success are removed by default.	`true` or `false`
`impex.importConfig.failOnError.enabled`	Allows you to define whether an ImpEx import should stop in case of errors or try to continue and import as much as possible.	`true` or `false`

Ant Update Properties

Ant Update Properties

These properties allow the same level of control separated for an update from ant as the one available when executing an update from Administration Console. These properties reflect the options available on the **Update** tab in Administration Console.

Property Name	Description	Possible Values
`patches.update.updateRunningSystem.enabled`	Allows you to define whether `ant updatesystem` should perform update running system action.	`true` or `false`
`patches.update.importEssentialData.enabled`	Allows you to define whether `ant updatesystem` should load essential data.	`true` or `false`
`patches.update.localizeTypes.enabled`	Allows you to define whether `ant updatesystem` should update localized types.	`true` or `false`
`patches.update.executeProjectData.extensionName.list`	Allows you to define a list of names of extensions for which the project data should be loaded during the running of the `ant updatesystem` script.	List of comma-separated extension names

Tracking Patches

Property Name	Descriptions	Possible Values
`patches.jenkinsBuild`	Allows you to keep information about the Jenkins build that should be set to PatchExecution.	Build number

Property Name	Descriptions	Possible Values
`patches.gitHashCode`	Allows you to keep information about a git hash code that should be set to PatchExecution.	Hash code

About the ldap Extension

LDAP is a protocol that defines the method by which directory data is accessed. It defines and describes how data is represented in the directory service (the Data Model). It also defines how data is imported and exported in a directory service (using LDIF).

LDAP does not define how data is stored or manipulated. Data storage is a **hidden** process as far as the standard is concerned.

LDAP defines four models:

Information Model

The Information Model defines how the information or data is **represented** in an LDAP enabled system - this may, or may NOT, be the way the data is actually stored as explained above.

Naming Model

This model defines how we organize and refer to our data. It describes the types of structures we can build out of our individual building blocks. It specifies that entries are arranged in an inverted tree structure much like the UNIX file system hierarchy.

Functional Model

How to read, search, write or modify the LDAP is specified by the Functional Model.

Security Model

The framework for protecting the LDAP directory information from unauthorized accesses.

LDAP vs. Database

LDAP is characterized as a 'write-once-read-many-times' service. This is to say, the type of data that would normally be stored in an LDAP service would not be expected to change on every access.

To illustrate:

LDAP would NOT be suitable for maintaining banking transaction records since, by their nature, they change on every access (transaction). LDAP would, however, be eminently suitable for maintaining details of the bank branches, hours of opening, employees, etc.

Read optimized

It is never clear in the phrase 'write-once-read-many-times' just how many is 'many'?

The literature is sparse on this topic and tends to stick with 'dummy' LDAP applications like address books which change once in living memory.

Visibility of Data Organization

The primitives that are used to access LDAP enabled directories use a model of the data that is (may be) abstracted from its physical organization. The primitives assume an object data model without being aware of the actual structure of the data. Indeed the relative simplicity of LDAP comes from this characteristic. This is in contrast to SQL in which the SQL queries have complete and detailed knowledge of the data structures and organization into tables etc.

Data Synchronization

Relational and other databases go to extreme lengths to ensure that data is consistent during write/update cycles using transactions, locking and other methods. This is a vital and necessary requirement.

The data in a source LDAP server and its replicas use a simple asynchronous replication process. This has the effect of leaving the source and replica systems out of data synchronization during the replication cycle. A query to the source and replica during this (usually short) period of time may yield a different answer.

LDAP Data (Object) Model

LDAP-enabled directories use a data model that assumes or represents the data as a hierarchy of objects. This does not imply that LDAP is an object-oriented database. LDAP itself is a protocol that allows access to an LDAP-enabled service and does not define how the data is stored - but the operational primitives (read, delete, modify) operate on a model (description) of the data that has object-like characteristics (mostly).

Object Tree Structure

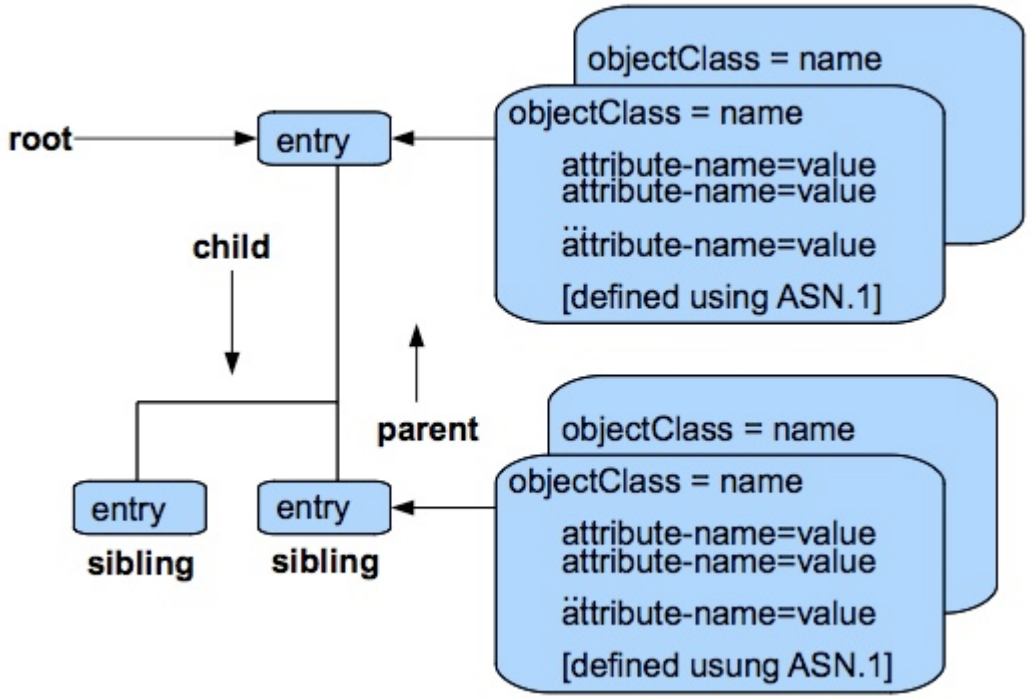
Data is represented in an LDAP-enabled directory as a hierarchy of objects, each of which is called an entry. The resulting tree structure is called a Data Information Tree (DIT). The top of the tree is commonly called the root (also known as the base or the suffix).

Each entry in the tree has one parent entry (object) and one or more child entries (objects). Each child entry (object) is a sibling of its parent's other child entries.

Each entry is composed of (is an instance of) one or more objectClasses. Objectclasses contain zero or more attributes. Attributes have names (and sometimes abbreviations or aliases) and typically contain data (at last!).

The characteristics (properties) of objectClasses and their attributes are described by ASN.1 definitions.

The diagram below illustrates these relationships:



Summary:

1. Each **Entry** is composed of one or more **objectClasses**
2. Each **objectClass** has a name.
3. Each **Attribute** has a name, usually contains data and is a member of an **object class**.

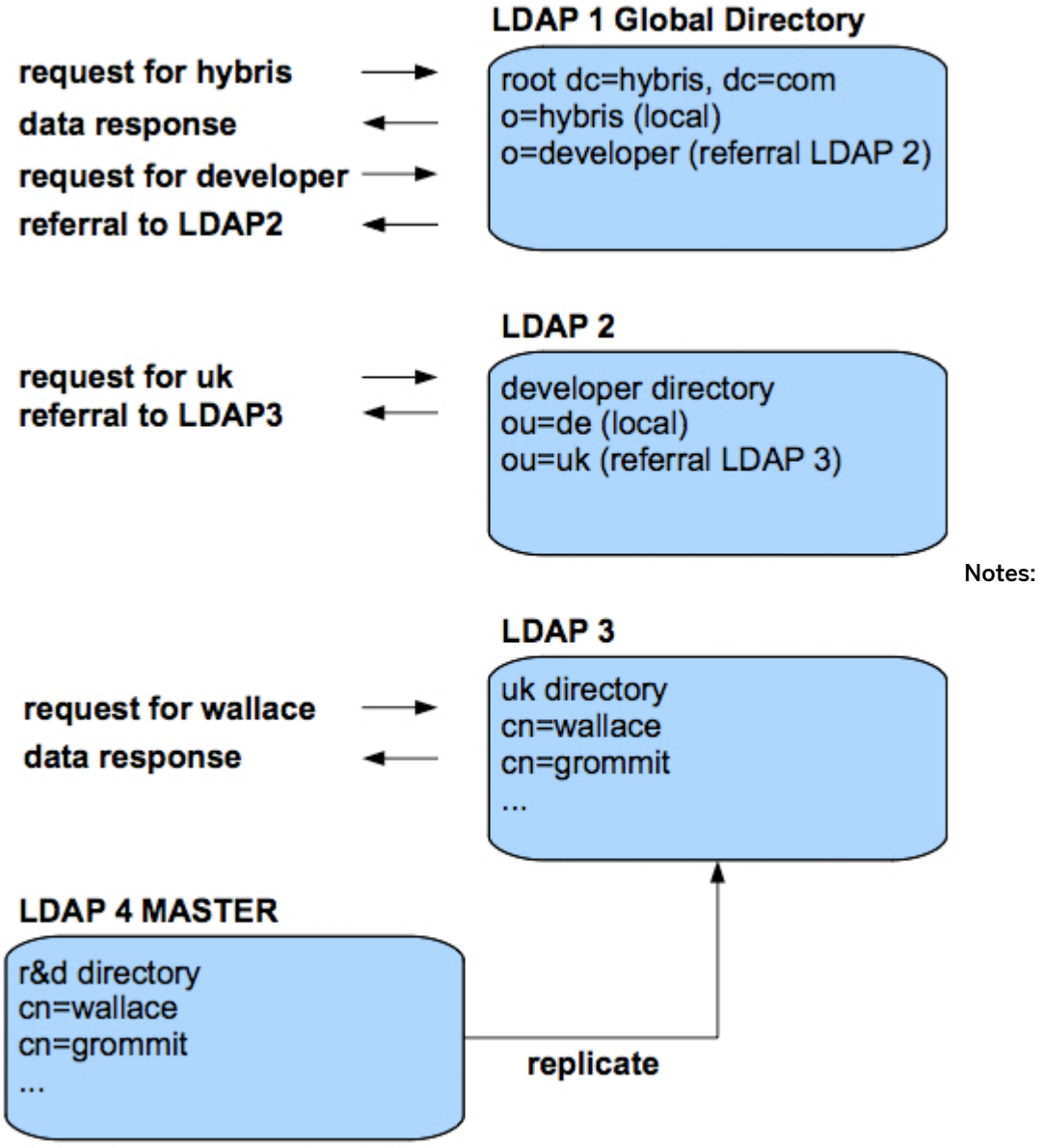
LDAP Organization (Referrals)

One of the more powerful aspects of LDAP (and X.500) is the inherent ability within the design to delegate the responsibility for maintenance of a part of the directory while continuing to see the directory as a consistent whole. Thus, a company directory may delegate (referral is the LDAP term) the responsibility for a particular department's part of the overall directory to that department. In this respect LDAP almost exactly mirrors the DNS delegation concept. Unlike the DNS system, there is no option in the standards to tell the LDAP server to follow (resolve) a referral - it is left to the LDAP client to directly contact the new server using the returned referral.

Because the standard does not define LDAP data organization, it does not contravene the standard for an LDAP server to follow (resolve) the link and some commercial LDAP servers perform this function automatically using a process that is sometimes called chaining. The built-in replication features of LDAP allow multiple copies of a directory to be replicated from a single source thus inherently creating a resilient structure.

It is important, however, to emphasize the difference between LDAP and a transactional database. When an update is performed on a master LDAP-enabled directory, it may take some time (in computing terms) to update all the replicas - the source and replicas may be unsynchronized for a period of time. In the LDAP context, temporary lack of synchronization is regarded as unimportant. In the case of a transactional database, even a temporary lack of synchronization, is regarded as catastrophic. This emphasizes the differences in the characteristics of data that should be maintained in an LDAP-enabled directory versus a transactional database.

All possible combinations of referral (delegation) and replication are possible, as shown in the diagram below:



Notes:

1. All client requests start at the global directory LDAP 1
2. At LDAP 1, requests for any data with **hybris** as an RDN in the DN are satisfied immediately from LDAP 1. e.g.

   ```
   dn: uid=wallace,ou=uk,o=hybris,dc=hybris,dc=com
   ```

3. At LDAP 1 requests for any data with **developer** as an RDN in the DN are referred to LDAP 2. e.g.

   ```
   dn: uid=wallace,ou=uk,o=developer,dc=hybris,dc=com
   ```

4. At LDAP 2, requests for any data with **uk** as an RDN in the DN are referred to LDAP 3. e.g.

   ```
   dn: uid=wallace,ou=uk,o=developer,dc=hybris,dc=com
   ```

5. LDAP 3 (a replica) is updated from LDAP 4 (the source). The state of LDAP 4 is **replicated** to LDAP 3.

Schemas, objectClasses and Attributes

When you create an entry in a DIT its data contents are contained in **attributes** which are grouped into **objectclasses** which are packaged into **schemas**.

The complexity and power of LDAP comes from the fact that there are bucket loads of attributes and bucket loads of objectclasses. Either you stick to some well-known applications in which case you can use some well-known objectclasses and

attributes or you invest some time to learn the language of objectclasses and attributes so that you can discover which objectclasses and attributes are truly best for your application - or even create your own.

Overview

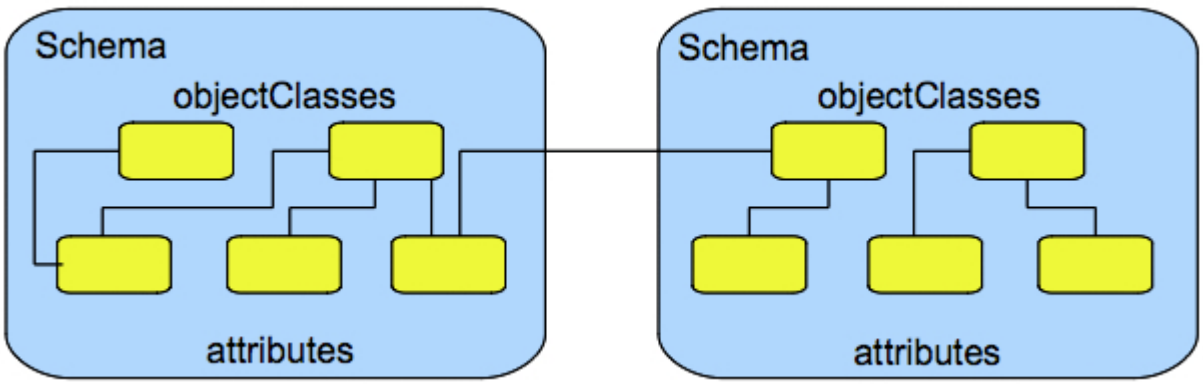
Everything in LDAP is hierarchical - so also with **objectclases** and **attributes**.

Schemas are important but so interesting being basically packaging units that **roughly** group together related **objectclasses** and **attributes**.

The important rules regarding each 'thing' are defined below:

1. **Schemas** are packaging units:
    - All **objectclasses** and all **attributes** are defined inside schemas (there are some objectclasses and attributes defined as operational which are embedded in the LDAP server software and do not need definition but we will ignore them just now).
    - All the schemas which include the **objectclasses** and **attributes** used in any LDAP implementation must be known to the LDAP server (in OpenLDAP these are defined using the include statement in the slapd.conf configuration file).
    - An **attribute** defined in one **schema** can be used by an **objectclass** defined in another **schema**.
2. **objectClasses** group sets of **attributes**:
    - **objectclasses** are defined inside **schemas**.
    - **objectclasses** may be organised in a hierarchy in which case they inherit all the properties of their parents.
    - **objectclasses** may be STRUCTURAL in which case they are used to create entries (data objects) or AUXILIARY in which case they may be added into any convenient entry or ABSTRACT - a non-existent 'thingie'. The most common ABSTRACT objectclass is **top** which forms the highest level of every objectclass hierarchy.
    - **objectclasses** are the means for including attributes (they are an attribute container in the jargon)
    - **objectclasses** define whether an attribute is mandatory (MUST be present) or optional (MAY be present).
    - **objectclasses** are defined using ASN.1 notation.
3. **Attributes** typically contain data:
    - Every **attribute** is included in one or more **objectclass**.
    - To use an **attribute** in an entry its **objectclass** must be included in the entry definition and its **objectclass** must be included in a **schema** which must be identified to the LDAP server.
    - An **attribute**'s characteristics are defined using ASN.1 notation.
    - An **attribute** definition may be part of a hierarchy in which case it inherits all the properties of its parents e.g. commonName (cn), givenName (gn), surname (sn) are all children of the **name** attribute.
    - An **attribute** definition includes its form e.g. string, number etc., how it behaves in certain conditions e.g. are compares case sensitive or case-insensitive and other characteristics (properties).

The following diagram illustrates some of these relationships:



Schemas

An LDAP schema is nothing more than a convenient packaging unit for containing broadly similar objectClasses and attributes.

There may have been a time when a single schema has designed to hold everything required for an LDAP implementation (like a relational database schema) but that is no longer true.

You will find useful attributes and objectclases scattered all over the place - the power of LDAP arguably comes from the ease of creating and using this apparent anarchy.

The rule is: Every attribute or objectclass (including its superior objectclass or attribute) used in an LDAP implementation must be defined in a **schema** and that schema must be **known** to the LDAP server.

ObjectClasses

An objectClass is a collection of attributes (or an attribute container) and has the following characteristics:

1. An objectclass is defined within a Schema
2. An objectclass may be a part of an objectclass hierarchy in which case it inherits all the properties of its parents, for example, inetOrgPerson is the child of organizationalPerson which is the child of person which is the child of **top** (the ABSTRACT objectClass which terminates every objectClass hirearchy).
3. An objectclass has a globally unique name or identifier
4. An objectclass, as well as being an attribute container, is also an attribute and may be searched on
5. An objectclass defines its member attributes and whether these MUST (mandatory) be present or MAY (optional) be present in an entry.
6. One or more objectclass(es) must be present in an LDAP entry.
7. Each objectclass supported by a LDAP server forms part of a **collection** called objectclasses which can be discovered via the subschema.

Attributes

Each attribute has a name and normally contains data. Attributes are always associated with (are members of) one or more ObjectClasses. Attributes have a number of interesting characteristics:

1. All attributes are members of one or more **objectclass(es)**
2. Each attribute defines the data type that it may contain.
3. Attributes can be optional (keyword is MAY) or mandatory (keyword is MUST) as described in the ASN.1 definitions for the objectclass of which they are a member. An attribute may be optional in one objectclass and mandatory in another. It is the objectclass which determines this property.

   One sees apparently random attributes being picked up from all over the place in the documentation - it's confusing at first but comes from the optional characteristic of most attributes. It allows a 'pick-n-mix' approach to populating an entry. Find the attribute you want, include its objectclass, and hope that all the other attributes that you don't want to use in the objectclass are optional! Try browsing here to get a feel for this.

4. Attributes can have single or multi values (as described in their ASN.1 definitions). Single means that only one data value may be present for the attribute. Multi means there can be one or or more data values for the attribute. If the attribute describes say, an email address, there can be one, two or 500 values - this is one of a number of methods of dealing with email aliases in directory designs. The default for an attribute is multi (allow multiple values).
5. Attributes have names and sometimes aliases or abbreviations (as described in their ASN.1 definitions) e.g. `commonName` is a member of the object class named `person` (and many others) and has an abbreviated name of `cn`. Either `commonName` or `cn` may be used to reference this attribute.
6. At each level in the hierarchy the data contained in one attribute should uniquely identify the entry. It can be any attribute in the entry. It can even be a combination of two or more attributes.

Suppose you have a classic white-pages directory containing names, phone number, addresses, favorite drink (yes there is a standard favoritedrink attribute) etc.. To uniquely identify a particular entry you may choose the person's name (the `commonName` or `cn` attribute). If the name is not unique in the LDAP directory e.g. 'Wallace' then a search for 'Wallace' will return all the entries containing the name 'Bob Smith' in the directory and the user will have to select the best one. For reading and searching this may be acceptable or even desirable since the human 'interface' uses well known information - the person's name.

For writing or updating an entry if 'Wallace' is not absolutely unique it is useless - which of the returned entries do we update? In this case it may be necessary to select another attribute that is absolutely unique. It is perfectly permissible to use more that one attribute to access the data depending on the context e.g. person's name for reading and searching, telephone number for writing. Furthermore, as defined above, it is perfectly permissible to use more that one attribute (`cn+favoritecheese`!) to define the unique entry.

The attribute value selected to contain the unique-ish data is called the naming attribute(s) or the Relatively Distinguished Name (RDN).

Matching Rules

Matching rules (matchingrules) are part of what is called the operational characteristics of the LDAP server.

Matching rules define the methods of comparison available in the LDAP server:

1. Matching rules are typically built-in to the LDAP server and do not need to be defined explicitly.
2. A matchingrule forms part of a collection called matchingrules which can be discovered via the subschema.
3. A matchingrule is defined for each attribute using the EQUALITY, SUBSTR, ORDERING properties as required - only those properties required are defined.

   If the search cannot use a wildcard there will be no SUBSTR property defined.

Defining matchingRule

Most matchingrules are built-in and you almost never need to define them but like everything in LDAP it has a defining syntax.

The following is an example of a matchingrule definition using `caseIgnoreMatch`:

```
matchingRule ( 2.5.13.2 NAME 'caseIgnoreMatch'
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )
```

The deconstruction shows the following:

matchingrule indicates this is a matchingrule definition.

2.5.13.2 NAME 'caseIgnoreMatch' defines the globally unique name for this matching rule and as always is comprised of two parts: **NAME 'caseIgnoreMatch'** allows reference to this matchingrule using some semi-understandable text and the OID **2.5.13.2** indicates the matching rule was defined by the X.500 standards group. Rule description:

"The Case Ignore Match rule compares for equality a presented string with an attibute value of type PrintableString, NumericString, TeletexString, BMPString, UniversalString or DirectoryString without regard for case (upper or lower) of the strings (e.g., "Hybris" and "HYBRIS" match).

The rule returns TRUE if the strings are the same length and corresponding characters are identical except possibly with regard to case.

SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 defines that this matchingrule operates on the type(s) defined - in this case a DirectoryString (a UTF-8 format string).

Operational Attributes and Objects

There are a bunch of attributes and objectclasses that are built into the LDAP server and govern how it works or functions. These attributes and object classes are typically called **operational**.

These **operational thingies** all live under the rootDSE and are not visible in normal operations.

LDIF

LDIF files are used in five general cases:

1. To initially construct the DIT structure.
2. To add (import) bulk records into a directory.
3. To restore (import) a directory.
4. To archive (export) a directory.
5. To apply bulk edits to a directory.

LDIF files are simple text files and can be created and edited with any suitable text editor. Since each line is terminated with EITHER <LF> (unix format) OR <CR><LF> (windows format) these files may be created on virtually any OS platform.

LDIF Format

LDIF can be pretty picky - with spaces or their lack being particularly important.

An LDIF consists of an number of LINE TYPES some of which can contain LDIF directives and which may be organized into ENTRY sequences and OPERATOR sequences.

Each line is terminated with EITHER <LF> (unix format) OR <CR><LF> (windows format).

To try and simplify further explanations we categorise the following LINE TYPES that we will refer to in the subsequent directive descriptions.

LDIF Terminology and Line Types

LDIF Line Types

1. DIRECTIVE LINE - a line which starts (in column 1) with any character EXCEPT SPACE or # (hash)
2. CONTINUATION LINE - a line which follows a DIRECTIVE LINE and starts (in column 1) with a SPACE - subsequent characters are assumed to be part of the previous line. Any number of CONTINUATION lines may exist up to any size limit imposed by the attribute.
3. BLANK LINE - a line which consists of no characters in column 1 (typically created with the ENTER key). BLANK LINES are typically used to separate ENTRY sequences.
4. COMMENT LINE - a line which starts (in column 1) with a # (hash) character.
5. SEPARATOR LINE - a line which starts (in column 1) with a - (dash) character. SEPARATOR LINES are typically used to terminate OPERATOR sequences.

LDIF Sequence

1. ENTRY sequence - a group of directives which normally start with a dn:directive and all of which apply to a single entry in the DIT.

   An ENTRY sequence normally starts and finishes with a BLANK line.

2. OPERATOR sequence - a group of directives used with a changetype: modify directive, and all of which apply to a single add, delete or replace operation.

An OPERATOR sequence normally terminates with either a SEPARATOR line or a BLANK line.

LDIF Sample

```
# this is a comment # MUST be in FIRST column - very picky

version: 1

## version not strictly necessary but good practice to include for future

## DEFINE DIT ROOT/BASE/SUFFIX ####
## uses RFC 2377 format

# this is an ENTRY sequence and is preceded by a BLANK line

dn: dc=hybris,dc=de
dc: hybris
description: My wonderful company as much text as you want to place
 in this line up to 32K continuation data for the line above must
 have <CR> or <CR><LF> i.e. ENTER works
 on both Windows and *nix system - new line MUST begin with ONE SPACE
objectClass: dcObject
objectClass: organization
o: hybris GmbH

## FIRST Level hierarchy - people
## uses mixed upper and lower case for objectclass
# this is an ENTRY sequence and is preceded by a BLANK line

dn: ou=people, dc=hybris,dc=de
ou: people
description: All people in organisation
objectclass: organizationalunit

## SECOND Level hierarchy
## ADD a single entry under FIRST (people) level
# this is an ENTRY sequence and is preceded by a BLANK line
# the ou: Human Resources is the department name

dn: cn=Herbert Mustermann,ou=people,dc=hybris,dc=de
objectclass: inetOrgPerson
cn: Herbert Mustermann
cn: Herbert A. Mustermann
sn: mustermann
uid: hmuster
userpassword: secret
carlicense: M HY-1234
homephone: 555-111-2222
mail: herbert.mustermann@hybris.de
mail: mustermann@hybris.de
mail: herbert.mustermann@hybris.com
mail: mustermann@hybris.com
description: nice guy
ou: Human Resources
```

Security

What security issues does LDAP (Lightweight Directory Access Protocol) have?

RFC 2829 - Authentication Methods for LDAP defines the basic threats to an LDAP directory service:

1. Unauthorized access to data via data-fetching operations,
2. Unauthorized access to reusable client authentication information by monitoring others' access,
3. Unauthorized access to data by monitoring others' access,
4. Unauthorized modification of data,

5. Unauthorized modification of configuration,
6. Unauthorized or excessive use of resources (denial of service), and
7. Spoofing of directory: Tricking a client into believing that information came from the directory when in fact it did not, either by modifying data in transit or misdirecting the client's connection.

Threats (1), (4), (5) and (6) are due to hostile clients. Threats (2), (3) and (7) are due to hostile agents on the path between client and server, or posing as a server.

Protecting LDAP Security

RFC 2829 - Authentication Methods for LDAP also defines the mechanisms by which the LDAP protocol suite can be protected:

1. Client authentication by means of the SASL mechanism set, possibly backed by the TLS credentials exchange mechanism,
2. Client authorization by means of access control based on the requestor's authenticated identity,
3. Data integrity protection by means of the TLS protocol or data-integrity SASL mechanisms,
4. Protection against snooping by means of the TLS protocol or data-encrypting SASL mechanisms,
5. Resource limitation by means of administrative limits on service controls, and
6. Server authentication by means of the TLS protocol or SASL mechanism.

LDAP Authentication Types

LDAP v3 specifies three authentication types:

- No Authentication
- Basic Authentication
- Simple Authentication and Security Layer (SASL)

The use of "No Authentication" is acceptable when sharing public data.

Basic Authentication is similar to Basic Authentication under HTTP. Authentication is accomplished through the use of a DN (Distinguished Name) and a password. This data is sent either in plaintext or encoded using Base64 encoding.

SASL (Simple Authentication and Security Layer) is a framework for plugging in alternative security mechanisms. These security mechanisms include:

- Kerberos Version 4
- S/Key
- GSSAPI
- CRAM-MD5
- TLS
- ANONYMOUS

OpenLDAP

Installation

Prerequisite software

OpenLDAP Software relies upon a number of third-party software packages.

Depending on the features you intend to use, you may have to download and install a number of additional software packages.

Obtaining and Extracting the Software

You can obtain OpenLDAP Software from the project's download page or directly from the project's FTP service.

Installation steps

Transport Layer Security

OpenLDAP clients and servers require installation of OpenSSL TLS libraries to provide Transport Layer Security services.

Though some operating systems may provide these libraries as part of the base system or as an optional software component, OpenSSL often requires separate installation.

OpenLDAP Software will not be fully LDAPv3 compliant unless OpenLDAP's configure detects a usable OpenSSL installation.

Kerberos Authentication Services

OpenLDAP clients and servers support Kerberos-based authentication services.

In particular, OpenLDAP supports the SASL/GSSAPI authentication mechanism using either Heimdal or MIT Kerberos V packages.

If you desire to use Kerberos-based SASL/GSSAPI authentication, you should install either Heimdal or MIT Kerberos V.

Simple Authentication and Security Layer

OpenLDAP clients and servers require installation of Cyrus's SASL libraries to provide Simple Authentication and Security Layer services. Though some operating systems may provide this library as part of the base system or as an optional software component, Cyrus SASL often requires separate installation.

Cyrus SASL will make use of OpenSSL and Kerberos/GSSAPI libraries if preinstalled.

OpenLDAP Software will not be fully LDAPv3 compliant unless OpenLDAP's configure detects a usable Cyrus SASL installation.

Database Software

OpenLDAP's slapd BDB and HDB

primary database backends require ***Berkeley DB***. If not available at configure time, you will not be able build slapd with these primary database backends.

Berkeley DB is available from Oracle's download page.

There are several versions available. Generally, the most recent release (with published patches) is recommended. This package is required if you wish to use the BDB or HDB database backends.

Common problems

Running ./configure fails

If the configuration process fails with the following message:

```
configure: error: BDB/HDB: BerkeleyDB not available
```

You have to add the following settings to your environment:

```
CPPFLAGS="-I/usr/local/BerkeleyDB.4.5/include"
export CPPFLAGS
LDFLAGS="-L/usr/local/lib -L/usr/local/BerkeleyDB.4.5/lib -R/usr/local/BerkeleyDB.4.5/lib"
export LDFLAGS
LD_LIBRARY_PATH=/usr/local/BerkeleyDB.4.5/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

ldap_bind: Invalid credentials (49)

On SLES 9, it looks like it's not possible to do a simple bind against the rootdn if your password is specified in cleartext.

Especially if you have build the version of OpenLDAP with --disable-cleartext

The workaround would be to use slappasswd like this:

```
% slappasswd -s secret
\{SSHA\}ioGadl0574KxRPecJ7Pb5q33j2x/Fi3w
```

Then you can paste in the resulting hash into your config file (/usr/local/etc/openldap/slapd.conf) as your rootpw directive.

LDAP Extension

Moved to here.

Resources

RFC's

RFC 1823	The LDAP Application Program Interface. T. Howes, M. Smith. August 1995. (Format: TXT=41081 bytes) (Status: INFORMATIONAL)
RFC 2247	Using Domains in LDAP/X.500 Distinguished Names. S. Kille, M. Wahl, A. Grimstad, R. Huber, S. Sataluri. January 1998. (Format: TXT=12411 bytes) (Status: PROPOSED STANDARD)
RFC 2251	Lightweight Directory Access Protocol (v3). M. Wahl, T. Howes, S. Kille. December 1997. (Format: TXT=114488 bytes) (Updated by RFC3377) (Status: PROPOSED STANDARD)
RFC 2252	Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions. M. Wahl, A. Coulbeck, T. Howes, S. Kille. December 1997. (Format: TXT=60204 bytes) (Updated by RFC3377) (Status: PROPOSED STANDARD)
RFC 2253	Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names. M. Wahl, S. Kille, T. Howes. December 1997. (Format: TXT=18226 bytes) (Obsoletes RFC1779) (Updated by RFC3377) (Status: PROPOSED STANDARD)
RFC 2254	The String Representation of LDAP Search Filters. T. Howes. December 1997. (Format: TXT=13511 bytes) (Obsoletes RFC1960) (Updated by RFC3377) (Status: PROPOSED STANDARD)
RFC 2255	The LDAP URL Format. T. Howes, M. Smith. December 1997. (Format: TXT=20685 bytes) (Obsoletes RFC1959) (Updated by RFC3377) (Status: PROPOSED STANDARD)

RFC 2256	A Summary of the X.500(96) User Schema for use with LDAPv3. M. Wahl. December 1997. (Format: TXT=32377 bytes) (Updated by RFC3377) (Status: PROPOSED STANDARD)
RFC 2307	An Approach for Using LDAP as a Network Information Service. L. Howard. March 1998. (Format: TXT=41396 bytes) (Status: EXPERIMENTAL)
RFC 2377	Naming Plan for Internet Directory-Enabled Applications. A. Grimstad, R. Huber, S. Sataluri, M. Wahl. September 1998. (Format: TXT=38274 bytes) (Status: INFORMATIONAL)
RFC 2425	A MIME Content-Type for Directory Information. T. Howes, M. Smith, F. Dawson. September 1998. (Format: TXT=64478 bytes) (Status: PROPOSED STANDARD)
RFC 2426	vCard MIME Directory Profile. F. Dawson, T. Howes. September 1998. (Format: TXT=74646 bytes) (Status: PROPOSED STANDARD)
RFC 2596	Use of Language Codes in LDAP. M. Wahl, T. Howes. May 1999. (Format: TXT=17413 bytes) (Status: PROPOSED STANDARD)
RFC 2696	LDAP Control Extension for Simple Paged Results Manipulation. C. Weider, A. Herron, A. Anantha, T. Howes. September 1999. (Format: TXT=12809 bytes) (Status: INFORMATIONAL)
RFC 2713	Schema for Representing Java(tm) Objects in an LDAP Directory. V. Ryan, S. Seligman, R. Lee. October 1999. (Format: TXT=40745 bytes) (Status: INFORMATIONAL)
RFC 2714	Schema for Representing CORBA Object References in an LDAP Directory. V. Ryan, R. Lee, S. Seligman. October 1999. (Format: TXT=14709 bytes) (Status: INFORMATIONAL)
RFC 2739	Calendar Attributes for vCard and LDAP. T. Small, D. Hennessy, F. Dawson. January 2000. (Format: TXT=25892 bytes) (Status: PROPOSED STANDARD)
RFC 2798	Definition of the inetOrgPerson LDAP Object Class. M. Smith. April 2000. (Format: TXT=32929 bytes) (Status: INFORMATIONAL)
RFC 2829	Authentication Methods for LDAP. M. Wahl, H. Alvestrand, J. Hodges, R. Morgan. May 2000. (Format: TXT=33471 bytes) (Updated by RFC3377) (Status: PROPOSED STANDARD)
RFC 2849	The LDAP Data Interchange Format (LDIF) - Technical Specification. G. Good. June 2000. (Format: TXT=26017 bytes) (Status: PROPOSED STANDARD)
RFC 2891	LDAP Control Extension for Server Side Sorting of Search Results. T. Howes, M. Wahl, A. Anantha. August 2000. (Format: TXT=15833 bytes) (Status: PROPOSED STANDARD)
RFC 2927	MIME Directory Profile for LDAP Schema. M. Wahl. September 2000. (Format: TXT=16122 bytes) (Status: INFORMATIONAL)
RFC 3045	Storing Vendor Information in the LDAP root DSE. M. Meredith. January 2001. (Format: TXT=10518 bytes) (Status: INFORMATIONAL)

RFC 3062	LDAP Password Modify Extended Operation. K. Zeilenga. February 2001. (Format: TXT=11807 bytes) (Status: PROPOSED STANDARD)
RFC 3296	Named Subordinate References in Lightweight Directory Access Protocol (LDAP) Directories. K. Zeilenga. July 2002. (Format: TXT=27389 bytes) (Status: PROPOSED STANDARD)
RFC 3377	Lightweight Directory Access Protocol (v3): Technical Specification. J. Hodges, R. Morgan. September 2002. (Format: TXT=9981 bytes) (Updates RFC2251, RFC2252, RFC2253, RFC2254, RFC2255, RFC2256, RFC2829, RFC2830) (Status: PROPOSED STANDARD)
RFC 3383	Internet Assigned Numbers Authority (IANA) Considerations for the Lightweight Directory Access Protocol (LDAP). K. Zeilenga. September 2002. (Format: TXT=45893 bytes) (Also BCP0064) (Status: BEST CURRENT PRACTICE)
RFC 3384	Lightweight Directory Access Protocol (version 3) Replication Requirements. E. Stokes, R. Weiser, R. Moats, R. Huber. October 2002. (Format: TXT=66871 bytes) (Status: INFORMATIONAL)
RFC 3671	Collective Attributes in the Lightweight Directory Access Protocol (LDAP). K. Zeilenga. December 2003. (Format: TXT=17912 bytes) (Status: PROPOSED STANDARD)
RFC 3672	Subentries in the Lightweight Directory Access Protocol (LDAP). K. Zeilenga. December 2003. (Format: TXT=24447 bytes) (Status: PROPOSED STANDARD)
RFC 3673	Lightweight Directory Access Protocol version 3 (LDAPv3): All Operational Attributes. K. Zeilenga. December 2003. (Format: TXT=10003 bytes) (Status: PROPOSED STANDARD)
RFC 3674	Feature Discovery in Lightweight Directory Access Protocol (LDAP). K. Zeilenga. December 2003. (Format: TXT=10282 bytes) (Status: PROPOSED STANDARD)
RFC 3687	Lightweight Directory Access Protocol (LDAP) and X.500 Component Matching Rules. S. Legg. February 2004. (Format: TXT=96256 bytes) (Status: PROPOSED STANDARD)
RFC 3698	Lightweight Directory Access Protocol (LDAP): Additional Matching Rules. K. Zeilenga, Ed.. February 2004. (Format: TXT=17562 bytes) (Updates RFC2798) (Status: PROPOSED STANDARD)
RFC 3703	Policy Core Lightweight Directory Access Protocol (LDAP) Schema. J. Strassner, B. Moore, R. Moats, E. Ellesson. February 2004. (Format: TXT=142983 bytes) (Status: PROPOSED STANDARD)
RFC 3712	Lightweight Directory Access Protocol (LDAP): Schema for Printer Services. P. Fleming, I. McDonald. February 2004. (Format: TXT=62301 bytes) (Status: INFORMATIONAL)
RFC 3727	ASN.1 Module Definition for the LDAP and X.500 Component Matching Rules. S. Legg. February 2004. (Format: TXT=8297 bytes) (Status: PROPOSED STANDARD)

RFC 3771	The Lightweight Directory Access Protocol (LDAP) Intermediate Response Message. R. Harrison, K. Zeilenga. April 2004. (Format: TXT=17114 bytes) (Updates RFC2251) (Status: PROPOSED STANDARD)
RFC 3829	Lightweight Directory Access Protocol (LDAP) Authorization Identity Request and Response Controls. R. Weltman, M. Smith, M. Wahl. July 2004. (Format: TXT=11986 bytes) (Status: INFORMATIONAL)
RFC 3866	Language Tags and Ranges in the Lightweight Directory Access Protocol (LDAP). K. Zeilenga, Ed.. July 2004. (Format: TXT=31501 bytes) (Obsoletes RFC2596) (Status: PROPOSED STANDARD)
RFC 3876	Returning Matched Values with the Lightweight Directory Access Protocol version 3 (LDAPv3). D. Chadwick, S. Mullan. September 2004. (Format: TXT=24233 bytes) (Status: PROPOSED STANDARD)
RFC 3909	Lightweight Directory Access Protocol (LDAP) Cancel Operation. K. Zeilenga. October 2004. (Format: TXT=13423 bytes) (Status: PROPOSED STANDARD)
RFC 3928	Lightweight Directory Access Protocol (LDAP) Client Update Protocol (LCUP). R. Megginson, Ed., M. Smith, O. Natkovich, J. Parham. October 2004. (Format: TXT=36892 bytes) (Status: PROPOSED STANDARD)
RFC 4370	Lightweight Directory Access Protocol (LDAP) Proxied Authorization Control. R. Weltman. February 2006. (Format: TXT=10624 bytes) (Status: PROPOSED STANDARD)
RFC 4373	Lightweight Directory Access Protocol (LDAP) Bulk Update/Replication Protocol (LBURP). R. Harrison, J. Sermersheim, Y. Dong. January 2006. (Format: TXT=31091 bytes) (Status: INFORMATIONAL)
RFC 4403	Lightweight Directory Access Protocol (LDAP) Schema for Universal Description, Discovery, and Integration version 3 (UDDIv3). B. Bergeson, K. Boogert, V. Nanjundaswamy. February 2006. (Format: TXT=78747 bytes) (Status: INFORMATIONAL)
RFC 4510	Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map. K. Zeilenga. June 2006. (Format: TXT=12354 bytes) (Obsoletes RFC2251, RFC2252, RFC2253, RFC2254, RFC2255, RFC2256, RFC2829, RFC2830, RFC3377, RFC3771) (Status: PROPOSED STANDARD)
RFC 4511	Lightweight Directory Access Protocol (LDAP): The Protocol. J. Sermersheim, Ed.. June 2006. (Format: TXT=150116 bytes) (Obsoletes RFC2251, RFC2830, RFC3771) (Status: PROPOSED STANDARD)
RFC 4512	Lightweight Directory Access Protocol (LDAP): Directory Information Models. K. Zeilenga, Ed.. June 2006. (Format: TXT=108377 bytes) (Obsoletes RFC2251, RFC2252, RFC2256, RFC3674) (Status: PROPOSED STANDARD)
RFC 4513	Lightweight Directory Access Protocol (LDAP): Authentication Methods and Security Mechanisms. R. Harrison, Ed.. June 2006.

	(Format: TXT=80546 bytes) (Obsoletes RFC2251, RFC2829, RFC2830) (Status: PROPOSED STANDARD)
RFC 4514	Lightweight Directory Access Protocol (LDAP): String Representation of Distinguished Names. K. Zeilenga, Ed.. June 2006. (Format: TXT=31859 bytes) (Obsoletes RFC2253) (Status: PROPOSED STANDARD)
RFC 4515	Lightweight Directory Access Protocol (LDAP): String Representation of Search Filters. M. Smith, Ed., T. Howes. June 2006. (Format: TXT=23885 bytes) (Obsoletes RFC2254) (Status: PROPOSED STANDARD)
RFC 4516	Lightweight Directory Access Protocol (LDAP): Uniform Resource Locator. M. Smith, Ed., T. Howes. June 2006. (Format: TXT=3056 bytes) (Obsoletes RFC2255) (Status: PROPOSED STANDARD)
RFC 4517	Lightweight Directory Access Protocol (LDAP): Syntaxes and Matching Rules. S. Legg, Ed.. June 2006. (Format: TXT=114285 bytes) (Obsoletes RFC2252, RFC2256) (Updates RFC3698) (Status: PROPOSED STANDARD)
RFC 4518	Lightweight Directory Access Protocol (LDAP): Internationalized String Preparation. K. Zeilenga. June 2006. (Format: TXT=28166 bytes) (Status: PROPOSED STANDARD)
RFC 4519	Lightweight Directory Access Protocol (LDAP): Schema for User Applications. A. Sciberras, Ed.. June 2006. (Format: TXT=64996 bytes) (Obsoletes RFC2256) (Updates RFC2247, RFC2798, RFC2377) (Status: PROPOSED STANDARD)
RFC 4520	Internet Assigned Numbers Authority (IANA) Considerations for the Lightweight Directory Access Protocol (LDAP). K. Zeilenga. June 2006. (Format: TXT=34298 bytes) (Obsoletes RFC3383) (Also BCP0064) (Status: BEST CURRENT PRACTICE)
RFC 4521	Considerations for Lightweight Directory Access Protocol (LDAP) Extensions. K. Zeilenga. June 2006. (Format: TXT=34585 bytes) (Also BCP0118) (Status: BEST CURRENT PRACTICE)
RFC 4522	Lightweight Directory Access Protocol (LDAP): The Binary Encoding Option. S. Legg. June 2006. (Format: TXT=16276 bytes) (Status: PROPOSED STANDARD)
RFC 4523	Lightweight Directory Access Protocol (LDAP) Schema Definitions for X.509 Certificates. K. Zeilenga. June 2006. (Format: TXT=43753 bytes) (Obsoletes RFC2252, RFC2256, RFC2587) (Status: PROPOSED STANDARD)
RFC 4524	COSINE LDAP/X.500 Schema. K. Zeilenga, Ed.. June 2006. (Format: TXT=11245 bytes) (Obsoletes RFC1274) (Updates RFC2247, RFC2798) (Status: PROPOSED STANDARD)
RFC 4525	Lightweight Directory Access Protocol (LDAP) Modify-Increment Extension. K. Zeilenga. June 2006. (Format: TXT=11251 bytes) (Status: INFORMATIONAL)
RFC 4526	Lightweight Directory Access Protocol (LDAP) Absolute True and False Filters. K. Zeilenga. June 2006. (Format: TXT=10097 bytes) (Status: PROPOSED STANDARD)

RFC 4527	Lightweight Directory Access Protocol (LDAP) Read Entry Controls. K. Zeilenga. June 2006. (Format: TXT=15987 bytes) (Status: PROPOSED STANDARD)
RFC 4528	Lightweight Directory Access Protocol (LDAP) Assertion Control. K. Zeilenga. June 2006. (Format: TXT=12539 bytes) (Status: PROPOSED STANDARD)
RFC 4529	Requesting Attributes by Object Class in the Lightweight Directory Access Protocol. K. Zeilenga. June 2006. (Format: TXT=11927 bytes) (Status: INFORMATIONAL)
RFC 4530	Lightweight Directory Access Protocol (LDAP) entryUUID Operational Attribute. K. Zeilenga. June 2006. (Format: TXT=15191 bytes) (Status: PROPOSED STANDARD)
RFC 4531	Lightweight Directory Access Protocol (LDAP) Turn Operation. K. Zeilenga. June 2006. (Format: TXT=18986 bytes) (Status: EXPERIMENTAL)
RFC 4533	Lightweight Directory Access Protocol (LDAP) "Who am I?" Operation. K. Zeilenga. June 2006. (Format: TXT=14247 bytes) (Status: PROPOSED STANDARD)
RFC 4533	The Lightweight Directory Access Protocol (LDAP) Content Synchronization Operation. K. Zeilenga, J.H. Choi. June 2006. (Format: TXT=73895 bytes) (Status: EXPERIMENTAL)

Commonly used attributes

Abbrev.	Name	objectClass	Description	Schema
c	countryName	country	2 character country code defined in ISO 3166	core.schema
cn	commonName	person organizationalPerson organizationalRole groupOfNames applicationProcess applicationEntity posixAccount device		core.schema
dc	domainComponent	dcObject	any part of a domain name e.g. domain.com, domain or com	core.schema
-	facsimileTelephoneNumber	residentialPerson organizationalRole organizationalPerson		core.schema

co	friendlyCountryName	friendlyCountry	full name of country	cosine.schema
gn	givenName	inetOrgPerson	First or given name	core.schema
homePhone	homeTelephoneNumber	inetOrgPerson		cosine.schema
-	jpegPhoto	inetOrgPerson	jpg format photo	inetorgperson.schema
l	localityName	locality organizationalPerson		core.schema
mail	rfc822Mailbox	inetOrgPerson	email address e.g. joe@smokeyjoe.com	core.schema
mobile	mobileTelephoneNumber	inetOrgPerson	mobile or cellular phone number	cosine.schema
o	organizationName	organization	Organization name or even organisational name	core.schema
ou	organisationalUnitName	organizationUnit	Usually department or any sub entity of larger entity	core.schema
-	owner	groupOfNames device groupOfUniqueNames		core.schema
pager	pagerTelephoneNumber	inetOrgPerson		cosine.schema
-	postalAddress	organizationalPerson		core.schema
postalCode	postalCode	organizationalPerson	Post Code or ZIP	core.schema
sn	surname	person	surname or family name	core.schema
st	stateOrProvinceName	organizationalPerson		core.schema
street	streetAddress	organizationalPerson		core.schema
-	telephoneNumber	organizationalPerson		core.schema
userPassword	-	organization organizationalUnit person dmd simpleSecurityObject domain posixAccount	User password for some form of access control	core.schema
uid	userid	account inetOrgPerson posixAccount	various - mostly username or other unique value	core.schema

Object classes

Not an exhaustive list but shows the mandatory (**MUST**) and optional (**MAY**) attributes in some commonly used objectclasses.

Name	MUST	MAY	Schema
account	userid	description $ seeAlso $ localityName $ organizationName $ organizationalUnitName $ host	cosine.schema
country	c	searchGuide $ description	core.schema
dcObject	dc	-	core.schema
device	cn	serialNumber $ seeAlso $ owner $ ou $ o $ l $ description	core.schema
friendlyCountry [->country]	friendlyCountyName	-	cosine.schema
groupOfNames	member $ cn	businessCategory $ seeAlso $ owner $ ou $ o $ description	core.schema
groupOfUniqueNames	uniqueMember $ cn	businessCategory $ seeAlso $ owner $ ou $ o $ description	core.schema
inetOrgPerson [->person]	-	audio $ businessCategory $ carLicense $ departmentNumber $ displayName $ employeeNumber $ employeeType $ givenName $ homePhone $ homePostalAddress $ initials $ jpegPhoto $ labeledURI $ mail $ manager $ mobile $ o $ pager $ photo $ roomNumber $ secretary $ uid $ userCertificate $ x500uniqueIdentifier $ preferredLanguage $ userSMIMECertificate $ userPKCS12	inetorgperson.schema
locality	-	street $ seeAlso $ searchGuide $ st $ l $ description	core.schema
organizationalPerson [->person]	-	title $ x121Address $ registeredAddress $ destinationIndicator $ preferredDeliveryMethod $ telexNumber $ teletexTerminalIdentifier $ telephoneNumber $ internationaliSDNNumber $ facsimileTelephoneNumber $ street $ postOfficeBox $ postalCode $ postalAddress $	core.schema

		physicalDeliveryOfficeName $ ou $ st $ l	
organization	o	userPassword $ searchGuide $ seeAlso $ businessCategory $ x121Address $ registeredAddress $ destinationIndicator $ preferredDeliveryMethod $ telexNumber $ teletexTerminalIdentifier $ telephoneNumber $ internationaliSDNNumber $ facsimileTelephoneNumber $ street $ postOfficeBox $ postalCode $ postalAddress $ physicalDeliveryOfficeName $ st $ l $ description	core.schema
organizationalRole	cn	x121Address $ registeredAddress $ destinationIndicator $ preferredDeliveryMethod $ telexNumber $ teletexTerminalIdentifier $ telephoneNumber $ internationaliSDNNumber $ facsimileTelephoneNumber $ seeAlso $ roleOccupant $ preferredDeliveryMethod $ street $ postOfficeBox $ postalCode $ postalAddress $ physicalDeliveryOfficeName $ ou $ st $ l $ description	core.schema
organizationalUnit	ou	userPassword $ searchGuide $ seeAlso $ businessCategory $ x121Address $ registeredAddress $ destinationIndicator $ preferredDeliveryMethod $ telexNumber $ teletexTerminalIdentifier $ telephoneNumber $ internationaliSDNNumber $ facsimileTelephoneNumber $ street $ postOfficeBox $ postalCode $ postalAddress $ physicalDeliveryOfficeName $ st $ l $ description	core.schema
person	sn $ cn	userPassword $ telephoneNumber $ seeAlso $ description	core.schema
posixAccount	cn $ uid $ uidNumber $ gidNumber $ homeDirectory	userPassword $ loginShell $ gecos $ description	nis.schema

residentialPerson [->person]	l	businessCategory $ x121Address $ registeredAddress $ destinationIndicator $ preferredDeliveryMethod $ telexNumber $ teletexTerminalIdentifier $ telephoneNumber $ internationaliSDNNumber $ facsimileTelephoneNumber $ preferredDeliveryMethod $ street $ postOfficeBox $ postalCode $ postalAddress $ physicalDeliveryOfficeName $ st $ l	core.schema

Sample: Schema entry

```
objectclass ( 2.5.6.4 NAME 'organization' SUP top STRUCTURAL
  MUST o
  MAY ( userPassword $ searchGuide $ seeAlso $ businessCategory $
   x121Address $ registeredAddress $ destinationIndicator $
   preferredDeliveryMethod $ telexNumber $ teletexTerminalIdentifier $
   telephoneNumber $ internationaliSDNNumber $
   facsimileTelephoneNumber $ street $ postOfficeBox $ postalCode $
   postalAddress $ physicalDeliveryOfficeName $ st $ l $ description ) )
```

Related Information

Troubleshooting ActiveDirectory
LDAP Login Process
LDAP-Platform Synchronization Scenarios
Importing LDAP Data Using ImpEx
Installing OpenLDAP

Importing LDAP Data Using ImpEx

During the import process, the LDAP data (LDIF or Search Results) are internally converted to an ImpEx script, which is imported, finally. This transformation process heavily depends on the configured LDAP to SAP Commerce data mapping. The mapping describes which LDAP `objectClasses` are imported at all and of which SAP Commerce type these entries are.

The mapping of the LDAP and SAP Commerce system attributes is defined furthermore and there is a chance to define an additional ImpEx expression for every single ImpEx column, too. There exists also a mapping which allows the replacement of LDAP-based values. You can find a sample file (`configuration.xml`) in the resources directory of the `ldap` extension.

XML Schema of the Mapping Definition

/ext/ldap/resources/configuration.xsd

```
<?xml version="1.0" encoding="UTF-8" ?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="attribute">
    <xs:complexType mixed="true">
```

```
        <xs:choice>
          <xs:element ref="hybris" />
          <xs:element ref="impex" />
          <xs:element ref="ldap" />
        </xs:choice>
      </xs:complexType>
    </xs:element>

    <xs:element name="attributes">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="attribute" maxOccurs="unbounded" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>

    <xs:element name="code">
      <xs:complexType mixed="true" />
    </xs:element>

    <xs:element name="config">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="types" />
          <xs:element ref="mappings" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>

    <xs:element name="defaultimpexheaderentry">
      <xs:complexType mixed="true" />
    </xs:element>

    <xs:element name="hybris">
      <xs:complexType mixed="true" />
    </xs:element>

    <xs:element name="impex">
      <xs:complexType mixed="true" />
    </xs:element>

    <xs:element name="ldap">
      <xs:complexType mixed="true" />
    </xs:element>

    <xs:element name="mapping">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="attributes" />
          <xs:element ref="values" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>

    <xs:element name="mappings">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="mapping" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>

    <xs:element name="objectclass">
      <xs:complexType mixed="true" />
    </xs:element>

    <xs:element name="objectclasses">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="objectclass" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>
```

```
    <xs:element name="type">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="code" />
          <xs:element ref="objectclasses" />
          <xs:element ref="attributes" />
          <xs:element ref="defaultimpexheaderentry" minOccurs="0" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>

    <xs:element name="types">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="type" maxOccurs="unbounded" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>

    <xs:element name="value">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="ldap" />
          <xs:element ref="hybris" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>

    <xs:element name="values">
      <xs:complexType>
        <xs:sequence>
          <xs:element ref="value" />
        </xs:sequence>
      </xs:complexType>
    </xs:element>

  </xs:schema>
```

Examples

Definition of an LDAP to SAP Commerce objectClass/type and Value Mapping

In the example below the LDAP **objectClass organizationalPerson** is imported as an SAP Commerce **Employee**. The LDAP attribute **sn** of the **objectClass organizationalPerson**, is represented by the **Employee** attribute **uid** and the additional ImpEx expression **unique=true** is added.

So the generated ImpEx header entry looks like this:

```
INSERT_UPDATE Employee; ...; uid[unique=true];
```

Adding a Default ImpEx Header Entry

The definition:

```
<defaultimpexheaderentry>
  ldapaccount[virtual=true, default=true]
</defaultimpexheaderentry>
```

The above definition adds the column entry **[virtual=true, default=true]** at the end of each generated ImpEx header.

So every generated ImpEx header entry for the **current type** (here: Employee) looks like:

```
INSERT_UPDATE Employee; ...; ldapaccount[virtual=true, default=true]
```

Replacing of LDAP-Based Values

In the sample below the value of the LDAP attribute **ou** is replaced from **RD** to **Development**.

This replacement is executed before the LDAP to SAP Commerce mapping is applied!

```
<mappings>
  <mapping>
    <attributes>
      <attribute>ou</attribute>
    </attributes>
    <values>
      <value>
       <ldap>RD</ldap>
       <hybris>Development</hybris>
      </value>
    </values>
  </mapping>
</mappings>
```

Sample Transformation: LDAP to ImpEx

Applying the mapping definition below on the content:

LDIF

```
dn: CN=Hendrik Reh,OU=development,OU=hybris users,OU=hybris,dc=hybris,dc=de
objectClass: top
objectClass: person
objectClass: organizationalPerson
cn: Hendrik Reh
displayName: Hendrik Reh
memberOf: CN=Alle,OU=Shares,OU=hybris groups,OU=hybris,DC=hybris,DC=de
memberOf: CN=hybris_muc,OU=hybris Verteiler,OU=hybris,DC=hybris,DC=de
memberOf: CN=TemplatesReadonly,OU=Shares,OU=hybris groups,OU=hybris,DC=hybris,DC=de
memberOf: CN=DevelopmentFull,OU=Shares,OU=hybris groups,OU=hybris,DC=hybris,DC=de
memberOf: CN=ProductmarketingReadonly,OU=Shares,OU=hybris groups,OU=hybris,DC=hybris,DC=de
memberOf: CN=all_users,OU=hybris groups,OU=hybris,DC=hybris,DC=de
memberOf: CN=development,OU=hybris groups,OU=hybris,DC=hybris,DC=de
memberOf: CN=Install,OU=hybris groups,OU=hybris,DC=hybris,DC=de
memberOf: CN=everyone,OU=hybris groups,OU=hybris,DC=hybris,DC=de
memberOf: CN=backup,OU=hybris groups,OU=hybris,DC=hybris,DC=de
memberOf: CN=Entwickler,OU=hybris Verteiler,OU=hybris,DC=hybris,DC=de
sn: Reh
```

is transformed to:

ImpEx

```
#
# CN=Hendrik Reh,OU=development,OU=hybris users,OU=hybris,DC=hybris,DC=de
#
INSERT_UPDATE Employee; name; cn; dn; ldaplogin; uid[unique=true]; groups(dn)[translator=de.hybris.
Employee; Hendrik Reh; Hendrik Reh; CN=Hendrik Reh,OU=development,OU=hybris users,OU=hybris,dc=hybr
```

Sample Mapping Configuration

/ext/ldap/resources/configuration.xml

```
<config>
  <types>
    <type>
 <code>Employee</code>
      <objectclasses>
        <objectclass>organizationalPerson</objectclass>
      </objectclasses>
      <attributes>
        <attribute>
          <ldap>displayName</ldap>
          <hybris>name</hybris>
        </attribute>
        <attribute>
          <ldap>cn</ldap>
          <hybris>cn</hybris>
        </attribute>
        <attribute>
          <ldap>dn</ldap>
          <hybris>dn</hybris>
        </attribute>
        <attribute>
          <ldap>dn</ldap>
          <hybris>ldaplogin</hybris>
        </attribute>
        <attribute>
          <ldap>sn</ldap>
          <hybris>uid</hybris>
          <impex>[unique=true]</impex>
        </attribute>
        <attribute>
          <!--
          <ldap>dn</ldap>
          <hybris>groups</hybris>
          <impex>(dn)[translator=de.hybris.platform.ldap.impex.ExtractUserGroupTranslator, groupid=
          -->
          <ldap>memberOf</ldap>
          <hybris>groups</hybris>
          <impex>(dn)[translator=de.hybris.platform.ldap.impex.ActiveDirectoryGroupCollectionTransl
        </attribute>
      </attributes>
      <defaultimpexheaderentry>
        ldapaccount[virtual=true, default=true]
      </defaultimpexheaderentry>
    </type>
    <type>
 <code>UserGroup</code>
      <objectclasses>
        <objectclass>organizationalUnit</objectclass>
      </objectclasses>
      <attributes>
        <attribute>
          <ldap>ou</ldap>
          <hybris>uid</hybris>
          <impex>[unique=true]</impex>
        </attribute>
        <attribute>
          <ldap>dn</ldap>
          <hybris>cn</hybris>
        </attribute>
        <attribute>
          <ldap>dn</ldap>
          <hybris>dn</hybris>
        </attribute>
      </attributes>
    </type>
  </types>
  <mappings>
    <mapping>
```

```
        <attributes>
          <attribute>ou</attribute>
        </attributes>
        <values>
          <value>
            <ldap>RD</ldap>
            <hybris>Development</hybris>
          </value>
        </values>
      </mapping>
    </mappings>
  </config>
```

ldap Extension - Connection Configuration

In most cases only authenticated users are allowed to query a LDAP server, so you have to configure a user with the needed privileges in your **`project.properties`** file. This document explains the LDAP specific **`project.properties`** settings.

Debugging

Property Name	Description
ldap.connection.enable.tracing	Enable/disable LDAP ASN.1 BER & SSL tracing

Server Settings

Property Name	Description/Remarks
ldap.server.type	Definition of the LDAP server type. Supported types are: ActiveDirectory (Microsoft) and LDAP (OpenLDAP, iPlanet, and so on).
ldap.server.url	The URL's/IP's of the LDAP servers. Pattern: ***<IP|NAME>***: ***<PORT>***, default port is 389 (and 636 for SSL respectively). **Note** **Fail-Over** The SAP Commerce **ldap** extension supports fail-over to multiple servers. The ordered list of servers in **ldap.server.url** is used. **Ordered** means that all requests go to the first server in the configured list as long as it is available. If that server fails it goes to the next in the list until it finds an available server. Afterwords it periodically retries the servers at the top of the list to see if they have returned to life. If you want to specify more than one server, use ";" as the delimiter.
ldap.minimum.fail.backtime	Failback Time to the Primary Server. Handles fail back to the primary server. If the current server is the primary server, this method does nothing. Otherwise, if it has been greater then the minimum fail back time, the current server is reset to the primary server.

Property Name	Description/Remarks
ldap.server.root.dn	Root DN

LDAP Accout Settings

Property Name	Description/Remarks
ldap.jndi.principals	LDAP User Login
ldap.jndi.credentials	LDAP User Password

SSL Settings

Property Name	Description/Remarks
ldap.ssl.caKeystoreFile	Keystore File Name of Public Certificates (trusted CA signs)
ldap.ssl.caPassphrase	Password for the caKeystoreFile certificate. May be null if only simple, 'server authenticated' ssl is being used, and keystore type is **JKS**.
ldap.ssl.caKeystoreType	The type of cakeystore file. The keystore type defaults to Sun's JKS (at time of writting, the only keystore typ that the default Sun security provider will handle).
ldap.ssl.clientKeystoreFile	Keystore file name of the client's certificates, containing private keys. May be null if only simple, 'server authenticated' SSL is being used.
ldap.ssl.clientPassphrase	Password for the clientKeystoreFile certificate. May be null if only simple, 'server authenticated' ssl is being used.
ldap.ssl.clientKeystoreType	The type of clientkeystore file. The keystore type defaults to Sun's JKS (at time of writting, the only keystore type that the default Sun security provider will handle).
ldap.jndi.socket.factory	The used SSLSocketFactory implementation. Sample: `ldap.jndi.socket.factory=de.hybris.platform.ldap.connection.ssl.JNDISocketFactory`
ldap.security.protocol	The used security protocol. Actually only SSL is supported.

Connection/Provider Settings

Property Name	Description/Remarks
ldap.local.accounts.only	Definition of **local only** accounts. For the specified SAP Commerce users, no LDAP authentication will be initiated. **⚠ Caution** Change this entries only if you know exactly what you do!

JNDI Settings

Property Name	Description/Remarks
ldap.pool.enabled	Enable/disable JNDI connection pooling
ldap.pool.init.size	com.sun.jndi.ldap.connect.pool.initsize: The number of connections per connection identity to create when initially cre connection for the identity.
ldap.pool.pref.size	com.sun.jndi.ldap.connect.pool.prefsize: The preferred number of connections per connection identity that should be concurrently.
ldap.pool.maxsize	com.sun.jndi.ldap.connect.pool.maxsize: The maximum number of connections per connection identity that can be m concurrently.

ldap.pool.timeout	com.sun.jndi.ldap.connect.pool.timeout: The number of milliseconds that an idle connection may remain in the pool w closed and removed from the pool.
	
	`# JNDI provider settings (see http://java.sun.com/j2se/1.4.2/docs/guide/jndi/spec/jndi/jr`
	
	`ldap.jndi.factory=com.sun.jndi.ldap.LdapCtxFactory`
	`ldap.jndi.version=v3`
	`ldap.jndi.authentication=simple`

Examples

Microsoft Active Directory

Below you see the configuration for the following environment:

Active Directory Server:	ldap.foo.com
User:	herbert.mustermann
Domain:	foo

ℹ Note

Be Careful

The login

```
foo\herbert.mustermann
```

has to be written as

```
foo\\herbert.mustermann
```

project.properties

```
ldap.server.type=ActiveDirectory
ldap.server.url=ldap.foo.com
```

```
ldap.server.root.dn=dc=foo,dc=com

ldap.jndi.principals=foo\\herbert.mustermann
# OR:
# ldap.jndi.principals=CN=Herbert Mustermann,OU=development,OU=foo users,OU=foo,DC=foo,DC=com
ldap.jndi.credentials=mysecret

ldap.activedirectory.pagedresultscontrol.pagesize=100
ldap.activedirectory.fastbind.enable=true
```

→ Tip

Enabling Fastbind Recommended

To improve performance, SAP Commerce recommends enabling fastbind. As a positive side effect, fastbind also avoids re-connection issues if the connection to the ActiveDirectory server gets lost for some reason.

OpenLDAP (SSL)

Below you see the configuration for the following environment:

OpenLDAP Server:	192.168.108.131
User:	admin
SSL:	active

project.properties

```
ldap.server.type=LDAP
ldap.server.url=192.168.108.131:636
ldap.server.root.dn=dc=foo,dc=com

ldap.jndi.principals=cn=admin,dc=foo,dc=com
ldap.jndi.credentials=mysecret

ldap.ssl.caKeystoreFile=../ext/ldap/resources/security/cacerts
ldap.ssl.caPassphrase=mycapassphrase
ldap.ssl.caKeystoreType=JKS
ldap.ssl.clientKeystoreFile=../ext/ldap/resources/security/clientcerts
ldap.ssl.clientPassphrase=myclienpassphrase
ldap.ssl.clientKeystoreType=JKS

ldap.jndi.socket.factory=de.hybris.platform.ldap.connection.ssl.JNDISocketFactory

ldap.security.protocol=ssl
```

→ Tip

In case of problems with the authentication like **javax.net.ssl.SSLHandshakeException: sun.security.validator.ValidatorException: PKIX path building failed: sun.security.provider.certpath.SunCertPathBuilderException: unable to find valid certification path to requested target**

If OpenLDAP server uses an SSL certificate signed by an internally generated root certificate (not trusted by default), you need to:

1. Add the necessary root certificate to a keystore
2. Specify this using the ldap.ssl.caKeystoreFile property
3. Add your internal root certificate into the main JVM cacerts file on the servers

Related Information

Configuring the Behavior of SAP Commerce

ldap Extension - Sample Configuration

Below is a sample configuration for LDAP.

→ Tip

The hybris **ldap** extension contains a sample file with mappings from LDAP fields to types in SAP Commerce: **`${hybrishome}/bin/platform/ext/ldap/resources/configuration.xml`**.

Features

1. Authentification
2. LDIF Import (Wizard)
3. Import of a result set of a LDAP query (Wizard)
4. Group/User Import (Wizard)

Configuration

project.properties File

```
#################### LDAP SETTINGS ################################
                                    #
                                    # LDAP connection settings
                                    #
                                    ###########################################################

                                    # values: LDAP || ActiveDirectory
                                    ldap.server.type=

                                    # ldap.server.url <IP|NAME>:<PORT> // default port is 389
                                    ldap.server.url=

                                    #
                                    # typical settings for OpenLDAP, IBM Tivoli, SUN iPlanet &
                                    #

                                    # ldap.server.type=LDAP
                                    # ldap.server.url=192.168.108.131:636

                                    #
                                    # typical settings for Microsoft ActiveDirectory (used by A
                                    #

                                    # CAUTION: hymail.hybris.de will disable the configured acc

                                    # ldap.server.type=ActiveDirectory
                                    # ldap.server.url=hymail.hybris.de
                                    # PagedResultsControl (1.2.840.113556.1.4.802) pagesize set
                                    # ldap.activedirectory.pagedresultscontrol.pagesize=100
                                    # FastBindConnectionControl (1.2.840.113556.1.4.1781) activ
                                    # ldap.activedirectory.fastbind.enable=true
                                    # ldap.jndi.principals=ms-domain\\herbert.mustermann

                                    #
                                    # server independent user settings
                                    #
```

```
ldap.jndi.principals=
ldap.jndi.credentials=
ldap.server.root.dn=

#
# SSL certicate settings
#

# cacerts - the trusted certificates of directories and cer
# used by the LDAP extension to establish server-authentica

ldap.ssl.caKeystoreFile=../ext/ldap/resources/security/cace
ldap.ssl.caPassphrase=

# the keystore type defaults to Sun's JKS (at time of writt
# keystore type that the default Sun security provider will

ldap.ssl.caKeystoreType=JKS

# clientcerts - the clients own certificates and private ke
# refer to the sun docu (you can try importing pkcs#8, but

ldap.ssl.clientKeystoreFile=../ext/ldap/resources/security/
ldap.ssl.clientPassphrase=

# the keystore type defaults to Sun's JKS (at time of writt
# type that the default Sun security provider will handle

ldap.ssl.clientKeystoreType=JKS

#
# HINT: You should define the preceding settings in your lo
#

#
# Connection/Provider settings
#

# for the following hybris users, no ldap authentification
# change only if you know exactly what you do !!!
ldap.local.accounts.only=admin; anonymous

# connection pool settings (see http://java.sun.com/product

ldap.minimum.fail.backtime=90000
ldap.pool.init.size=1
ldap.pool.pref.size=10
ldap.pool.maxsize=50
ldap.pool.timeout=5000
ldap.pool.enabled=false

# JNDI provider settings (see http://java.sun.com/j2se/1.4.

ldap.jndi.factory=com.sun.jndi.ldap.LdapCtxFactory
ldap.jndi.version=v3
ldap.security.protocol=
ldap.jndi.authentication=simple
ldap.jndi.socket.factory=de.hybris.platform.ldap.connectior

# deprecated (only used by the deprectaed LDAPConfigProxyIt

ldap.login.field=cn
```

local.properties File

Below, you see the mandatory values for establishing a LDAP/SSL connection.

```
ldap.server.type=LDAP
ldap.server.url=192.168.108.131:636
ldap.security.protocol=ssl
ldap.jndi.principals=cn=Manager,dc=my-domain,dc=com
```

```
ldap.jndi.credentials=secret
ldap.server.root.dn=dc=my-domain,dc=com

#
# SSL certicate settings
#

# cacerts - the trusted certificates of directories and certificate
# used by the LDAP extension to establish server-authenticated SSL
ldap.ssl.caKeystoreFile=/home/mustermann/hybrisplatform/ext/ldap/re
ldap.ssl.caPassphrase=

# clientcerts - the clients own certificates and private keys.  Imp
# refer to the sun docu (you can try importing pkcs#8, but it doesn
ldap.ssl.clientKeystoreFile=/home/mustermann/hybrisplatform/ext/lda
ldap.ssl.clientPassphrase=
```

Troubleshooting ActiveDirectory

There is a description of how to troubleshoot ActiveDirectory below.

Reported Problem

I am using LDAP Active Directory configuration in my SAP Commerce application. When I enter username and password for the first time it is giving exception. And for the second time, its working correctly.

Typical exceptions/log entries

CommunicationException

```
javax.naming.CommunicationException: Connection reset [Root exception is java.net.SocketException:
        at com.sun.jndi.ldap.LdapCtx.doSearch(LdapCtx.java:1965)
        at com.sun.jndi.ldap.LdapCtx.doSearchOnce(LdapCtx.java:1897)
        at com.sun.jndi.ldap.LdapCtx.c_lookup(LdapCtx.java:991)
        at com.sun.jndi.toolkit.ctx.ComponentContext.p_lookup(ComponentContext.java:526)
        at com.sun.jndi.toolkit.ctx.PartialCompositeContext.lookup(PartialCompositeContext.java:159
        at com.sun.jndi.toolkit.ctx.PartialCompositeContext.lookup(PartialCompositeContext.java:148
        at javax.naming.InitialContext.lookup(InitialContext.java:392)
        at de.hybris.platform.ldap.connection.JNDIConnectionManager.getDirContext(JNDIConnectionMar
        at de.hybris.platform.ldap.connection.JNDIConnectionImpl.<init>(JNDIConnectionImpl.java:65)
        at de.hybris.platform.ldap.connection.LDAPConnectionFactory.getLDAPConnection(LDAPConnectio
        at de.hybris.platform.ldap.jalo.LDAPManager.checkPassword(LDAPManager.java:215)
        at de.hybris.platform.ldap.jalo.LDAPManager.checkPassword(LDAPManager.java:201)
        at de.hybris.platform.ldap.jaloimpl.LDAPUserEJBImpl.checkPassword(LDAPUserEJBImpl.java:95)
        at de.hybris.platform.jalo.user.User.checkPassword(User.java:409)
        at de.hybris.platform.jalo.JaloSession.performLogin(JaloSession.java:314)
        at de.hybris.platform.jalo.JaloSession.transfer(JaloSession.java:642)
```

LDAP: error code 49 - 80090308

```
DEBUG [ActiveDirectoryConnectionImpl] rawSearchSubTree( cn=development,ou=hybris groups,ou=hybris,
DEBUG [ActiveDirectoryConnectionImpl] used pagesize: 100
ERROR [ActiveDirectoryConnectionImpl] javax.naming.AuthenticationException: [LDAP: error code 49 -
WARN  [ActiveDirectoryConnectionImpl] Operation failed: javax.naming.AuthenticationException: [LDAP
de.hybris.platform.ldap.exception.LDAPOperationException: [LDAP: error code 49 - 80090308: LdapErr
```

Note: ***data 525 (user not found)*** represents an ActiveDirectory Error Code (see 8. ActiveDirectory Error Codes)

Testscenarios (working fine)

The extension was tested against a ActiveDirectory in the following scenarios:

- Client (Platform) is a member of the AD domain
- Client (Platform) is NOT a member of the AD domain
- Unplugging the ethernet cable during a running LDAP access and replugging after some time (testing of the **reconnect** mechanism)
- Configuring "invalid" and "valid" LDAP servers in ***local.porperties*** (testing of the **failover** mechanism)

  local.properties

  ```
  ldap.server.url=invalid.hybris.de; hymail.hybris.de
  ```

  and

  local.properties

  ```
  ldap.server.url=hymail.hybris.de; invalid.hybris.de
  ```

Testcases

- User/Group synchronization
- LDAP querying ***(admin_ldap_search.jsp)***
- **Stress testing** platform and AD Server by calling LDAP testpage ***(isAlive check, login process)*** **10.000 times**. Response time between 88 and 1327ms.

Questions for our partners

1. Which AD version/OS version are you running?

2. Your LDAP related ***project.properties***, ***local.properties*** and ***advanced.properties*** settings?

3. Installed LDAP (AD) related extensions?

Some AD related extensions

```
OID: 1.2.840.113556.1.4.319 (pagedResultsControl)
OID: 1.2.840.113556.1.4.801 (LDAP_SERVER_SD_FLAGS_OID)
OID: 1.2.840.113556.1.4.473 (sortKeyList)
OID: 1.2.840.113556.1.4.528 (LDAP_SERVER_NOTIFICATION_OID)
OID: 1.2.840.113556.1.4.417 (LDAP_SERVER_SHOW_DELETED_OID)
OID: 1.2.840.113556.1.4.619 (LDAP_SERVER_LAZY_COMMIT_OID)
OID: 1.2.840.113556.1.4.841 (replControlValue)
OID: 1.2.840.113556.1.4.529 (LDAP_SERVER_EXTENDED_DN_OID)
OID: 1.2.840.113556.1.4.805 (LDAP_SERVER_TREE_DELETE_OID)
OID: 1.2.840.113556.1.4.521 (LDAP_SERVER_CROSSDOM_MOVE_TARGET_OID)
OID: 1.2.840.113556.1.4.1338 (LDAP_SERVER_VERIFY_NAME_OID)
OID: 1.2.840.113556.1.4.474 (sortResult)
OID: 1.2.840.113556.1.4.1339 (LDAP_SERVER_DOMAIN_SCOPE_OID)
OID: 1.2.840.113556.1.4.1340 (LDAP_SERVER_SEARCH_OPTIONS_OID)
OID: 1.2.840.113556.1.4.1413 (LDAP_SERVER_PERMISSIVE_MODIFY_OID)
OID: 2.16.840.1.113730.3.4.9 (LDAP_CONTROL_VLVREQUEST VLV)
OID: 2.16.840.1.113730.3.4.10 (LDAP_CONTROL_VLVRESPONSE VLV)
OID: 1.2.840.113556.1.4.1504 (LDAP_SERVER_ASQ_OID)
...
```

4. Is there any available network traffic log (Wireshark etc.)? **(... don't forget to remove the credentials)**

5. Are there any SAP Commerce related Cookies 'active'? ***(LOGINTOKEN; JSESSION)***

6. Occurs the problem only for some specific accounts or is it a general problem?

7. Value of ***'ldap.activedirectory.fastbind.enable'***? ... should be ***true***.

8. SSL enabled?

9. Any Loadbalancer and/or Firewall involved?

Recommended *local.properties* settings for debugging

DEBUG mode

```
log4j.logger.de.hybris.platform.ldap=debug
log4j.logger.de.hybris.platform.ldap.connection=debug
log4j.logger.de.hybris.platform.ldap.jalo=debug
log4j.logger.de.hybris.platform.ldap.jaloimpl=debug
```

Increasing the Network Connection Timeout

It is a well know fact that not all JNDI based connection creations are successful.

If the LDAP provider cannot establish a connection within a certain timeout period, it aborts the connection attempt. By default, this timeout period is the network (TCP) timeout value, which is in the order of a few minutes.

You can do this by setting the **ldap.jndi.connection.timeout** property according to your needs.

advanced.properties

```
ldap.jndi.connection.timeout=60000
```

ActiveDirectory Error Codes

Common Active Directory LDAP bind errors:

80090308: LdapErr: DSID-0C09030B, comment: AcceptSecurityContext error, data 525, v893

HEX: 0x525 - user not found

DEC: 1317 - ERROR_NO_SUCH_USER (The specified account does not exist.)

NOTE: Returns when username is invalid.

80090308: LdapErr: DSID-0C09030B, comment: AcceptSecurityContext error, data 52e, v893

HEX: 0x52e - invalid credentials

DEC: 1326 - ERROR_LOGON_FAILURE (Logon failure: unknown user name or bad password.)

NOTE: Returns when username is valid but password/credential is invalid. Will prevent most other errors from being displayed as noted.

80090308: LdapErr: DSID-0C09030B, comment: AcceptSecurityContext error, data 530, v893

HEX: 0x530 - not permitted to logon at this time

DEC: 1328 - ERROR_INVALID_LOGON_HOURS (Logon failure: account logon time restriction violation.)

NOTE: Returns only when presented with valid username and password/credential.

80090308: LdapErr: DSID-0C09030B, comment: AcceptSecurityContext error, data 531, v893

HEX: 0x531 - not permitted to logon from this workstation

DEC: 1329 - ERROR_INVALID_WORKSTATION (Logon failure: user not allowed to log on to this computer.)

LDAP

NOTE: Returns only when presented with valid username and password/credential.

80090308: LdapErr: DSID-0C09030B, comment: AcceptSecurityContext error, data 532, v893

HEX: 0x532 - password expired

DEC: 1330 - ERROR_PASSWORD_EXPIRED (Logon failure: the specified account password has expired.)

LDAP - PASSWORDEXPIRED

NOTE: Returns only when presented with valid username and password/credential.

80090308: LdapErr: DSID-0C09030B, comment: AcceptSecurityContext error, data 533, v893

HEX: 0x533 - account disabled

DEC: 1331 - ERROR_ACCOUNT_DISABLED (Logon failure: account currently disabled.)

LDAP - ACCOUNTDISABLE

NOTE: Returns only when presented with valid username and password/credential.

80090308: LdapErr: DSID-0C09030B, comment: AcceptSecurityContext error, data 701, v893

HEX: 0x701 - account expired

DEC: 1793 - ERROR_ACCOUNT_EXPIRED (The user's account has expired.)

LDAP - ACCOUNTEXPIRED

NOTE: Returns only when presented with valid username and password/credential.

80090308: LdapErr: DSID-0C09030B, comment: AcceptSecurityContext error, data 773, v893

HEX: 0x773 - user must reset password

DEC: 1907 - ERROR_PASSWORD_MUST_CHANGE (The user's password must be changed before logging on the first time.)

LDAP - MUST_CHANGE_PASSWD

NOTE: Returns only when presented with valid username and password/credential.

80090308: LdapErr: DSID-0C09030B, comment: AcceptSecurityContext error, data 775, v893

HEX: 0x775 - account locked out

DEC: 1909 - ERROR_ACCOUNT_LOCKED_OUT (The referenced account is currently locked out and may not be logged on to.)

LDAP - LOCKOUT

NOTE: Returns even if invalid password is presented.

LDAP Login Process

The LDAP login process is described here.

1. During the startup process of SAP Commerce the `class de.hybris.platform.persistence.user.UserEJBImpl`, which implements the method `#checkPassword( ...)`, will be replaced by the `de.hybris.platform.ldap.jaloimpl.LDAPUserEJBImpl`.

   Due to this, every login process will now be handled by `LDAPUserEJBImpl` instead of `UserEJBImpl`.

2. The `LDAPUserEJBImpl` checks if the user, which login should be verified, is marked as an `LDAP Account` or not, and ...
3. ... calls the corresponding `#checkPassword( ... )` implementation.
4. ... which verifies the submitted password against the underlying LDAP or SAP Commerce system.

Related Information

LDAP Guide
LDAP-Platform Synchronization Scenarios
Configuration of the LDAP to HYBRIS Data & Value Mapping
OpenLDAP Install Guide

LDAP-Platform Synchronization Scenarios

Described here are several scenarios for LDAP to Platform synchronization.

Abstract

For some reasons (Order and History handling, etc.) every LDAP account needs a corresponding Platform user entry.

So in most cases, you will use the data, which are stored in the LDAP backend, for creating such an entry.

Solutions

This "user creation" can be realized in the following ways:

- 1. The LDAP server generates the LDIF file on demand (for example in case of a modified or a new user entry) and an SAP Commerce-based ***CronJob*** is responsible for importing this file.
- 2. Platform executes a LDAP search for getting the needed data and initiates an import process on the fly.

ℹ Note

The LDAP system is the leading system!

This means, that there will be no synchronization of modifed data from SAP Commerce to the LDAP system!

ℹ Note

Handling of removed LDAP users

You should never delete user accounts in your SAP Commerce!

Disable them instead!!! (see: User#setLoginDisabled(boolean))

Related Information

LDAP Guide
LDAP Login Process
Configuration of the LDAP to HYBRIS Data & Value Mapping
OpenLDAP Install Guide

Installing OpenLDAP

Here you can learn how to install OpenLDAP.

Prerequisite software

OpenLDAP Software relies upon a number of third-party software packages. Depending on the features you intend to use, you may have to download and install a number of additional software packages.

Transport Layer Security

OpenLDAP clients and servers require installation of OpenSSL TLS libraries to provide Transport Layer Security services. Though some operating systems may provide these libraries as part of the base system or as an optional software component, OpenSSL often requires separate installation.

OpenLDAP Software will not be fully LDAPv3 compliant unless OpenLDAP's configure detects a usable OpenSSL installation.

Kerberos Authentication Services

OpenLDAP clients and servers support Kerberos-based authentication services. In particular, OpenLDAP supports the SASL/GSSAPI authentication mechanism using either Heimdal or MIT Kerberos V packages. If you desire to use Kerberos-based SASL/GSSAPI authentication, you should install either Heimdal or MIT Kerberos V.

Simple Authentication and Security Layer

OpenLDAP clients and servers require installation of Cyrus's SASL libraries to provide Simple Authentication and Security Layer services. Though some operating systems may provide this library as part of the base system or as an optional software component, Cyrus SASL often requires separate installation.

Cyrus SASL will make use of OpenSSL and Kerberos/GSSAPI libraries if preinstalled.

OpenLDAP Software will not be fully LDAPv3 compliant unless OpenLDAP's configure detects a usable Cyrus SASL installation.

Database Software

OpenLDAP's slapd BDB and HDB primary database backends require ***Berkeley DB***. If not available at configure time, you will not be able build slapd with these primary database backends. ***Berkeley DB*** is available from Oracle's download page. There are several versions available. Generally, the most recent release (with published patches) is recommended. This package is required if you wish to use the BDB or HDB database backends.

Obtaining and Extracting the Software

You can obtain OpenLDAP Software from the project's download page or directly from the project's FTP service.

Common problems

Running ./configure fails

If the configuration process fails with the following message:

```
configure: error: BDB/HDB: BerkeleyDB not available
```

You have to add the following settings to your environment:

```
CPPFLAGS="-I/usr/local/BerkeleyDB.4.5/include"
export CPPFLAGS
LDFLAGS="-L/usr/local/lib -L/usr/local/BerkeleyDB.4.5/lib -R/usr/local/BerkeleyDB.4.5/lib"
export LDFLAGS
LD_LIBRARY_PATH=/usr/local/BerkeleyDB.4.5/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

ldap_bind: Invalid credentials (49)

On my system (SLES 9) it looks like it's not possible to do a simple bind against the rootdn if the password is specified in cleartext.

Especially if you have built the version of OpenLDAP with --disable-cleartext

The workaround would be to use slappasswd like this:

```
% slappasswd -s secret
\{SSHA\}ioGadl0574KxRPecJ7Pb5q33j2x/Fi3w
```

Then you can paste in the resulting hash into your config file (/usr/local/etc/openldap/slapd.conf) as your rootpw directive.

Related Information

[About the ldap Extension]

[LDAP Login Process]

[LDAP-Platform Synchronization Scenarios]

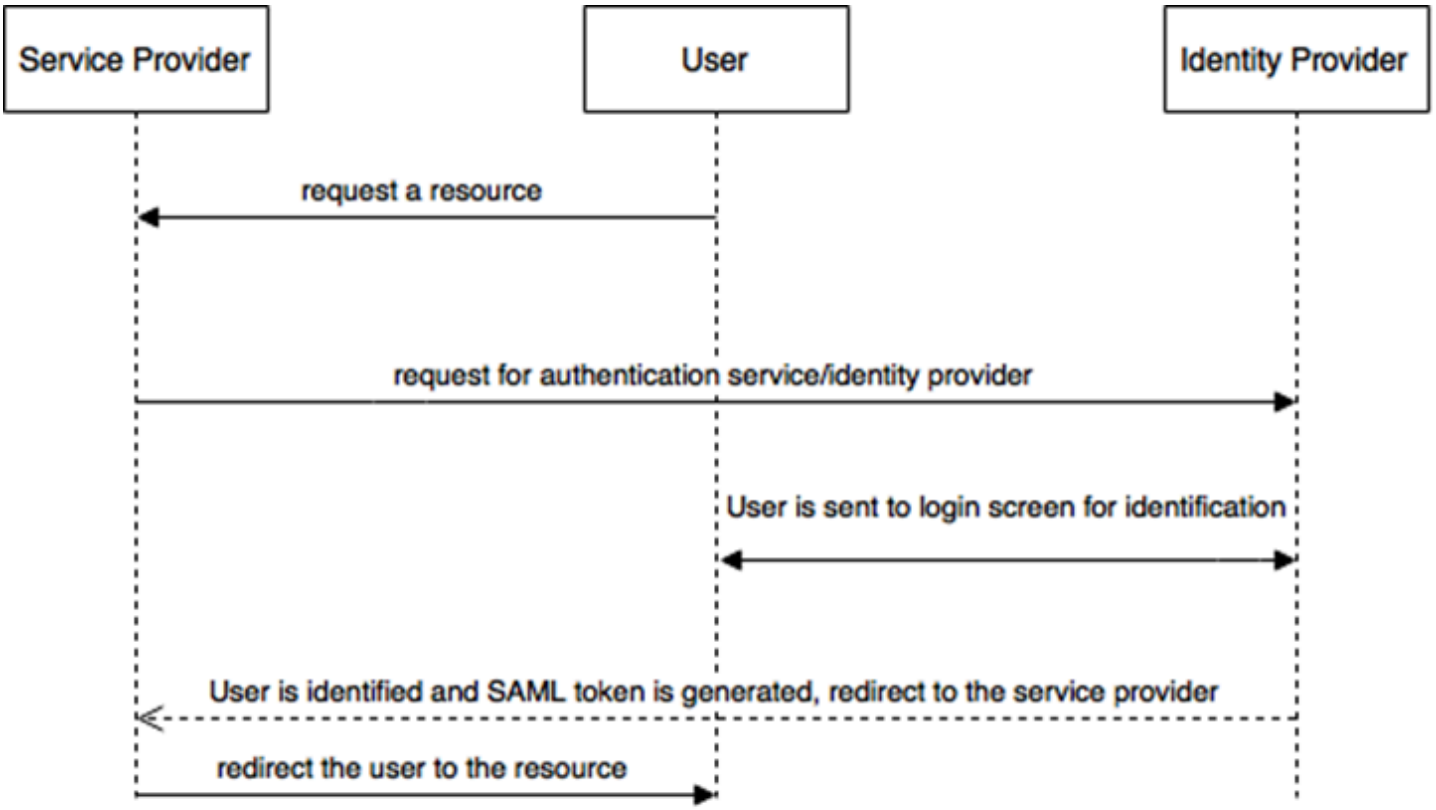
Single Sign-On in SAP Commerce

Single Sign-On (SSO) shares login authentication across multiple applications. It allows users to sign into one system and access multiple associated systems without having to sign into these again.

Security Assertion Markup Language (SAML) 2.0 is an XML-based protocol that uses security tokens containing assertions to pass information about a principal (usually an end user) between an identity provider, and a SAML consumer, that is, a service provider. SAML 2.0 enables web-based authentication and authorization scenarios including cross-domain SSO. The `samlsinglesignon` extension uses the Spring Security library and the SAML 2.0 protocol to authenticate and authorize data across different security domains.

SSO authenticates users for all applications they have been authorised to use and eliminates the need to log in when they switch applications during a particular session. SSO reduces the administrative overhead of distributing multiple authentication tokens to users.

In a typical SSO flow, users have to establish an identity with the identity provider. Then, when they attempt to access the desired application, the identity provider sends a SAML assertion to the service provider, verifying their identity and privileges. The service provider accepts the assertion and users gain access to the application.



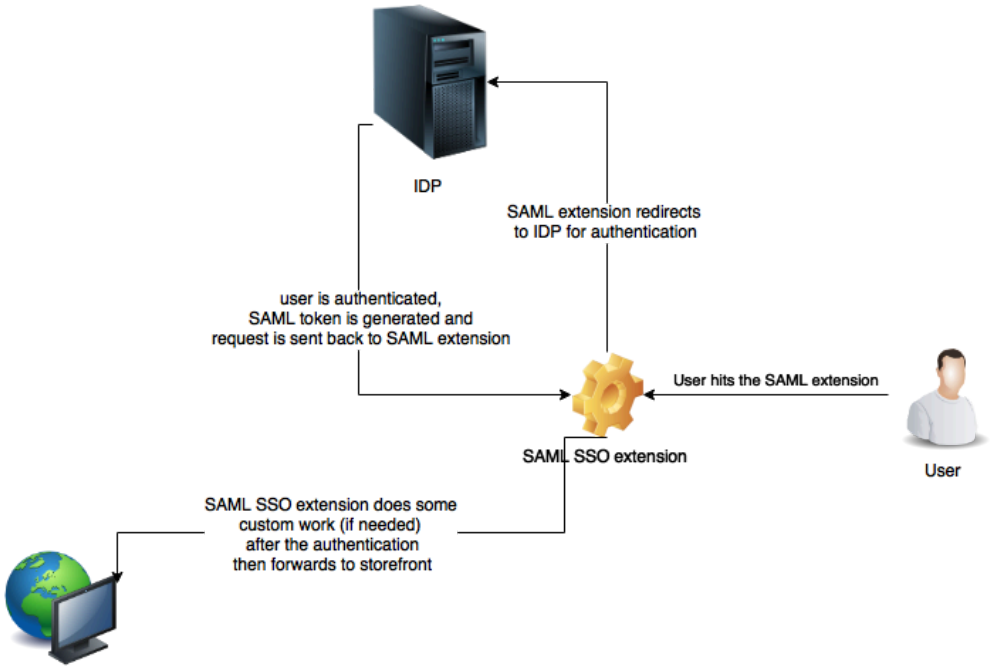
ℹ Note

External dependencies are placed in the `/WEB-INF/lib` folder of given web applications, which means that the `lib` folder in the module directory doesn't contain any dependencies.

Request Flow

When a user tries to access an SAP Commerce storefront, the request flow looks as follows:

1. Instead of going to the storefront directly, the user's request goes to the `samlsinglesignon` extension.
2. The `samlsinglesignon` extension listens for incoming requests and verifies if the user has a SAML assertion. If the user doesn't have a SAML assertion, the extension redirects the user to the identity provider and they're asked to log in.
3. After making sure the user is logged in and a SAML assertion exists for this particular user, the extension creates a secure cookie with the user's details and redirects them to the storefront. You can inject your custom logic for user after authentication.
4. The storefront consumes the cookie and takes any further actions required.



Single Sign-On Configuration

Configure SSO in SAP Commerce.

To use SSO with SAP Commerce, configure the following:

1. metadata files
2. certificates
3. Group level authorization

Metadata Files

SAML metadata is an XML document that contains information necessary for interaction with SAML-enabled identity or service providers.

Metadata files contain URLs of endpoints and information about supported bindings, identifiers, and public keys. Metadata files need to be exchanged between your service and identity provider.

ℹ Note

Metadata and keystore files are loaded relative to the `WEB-INF/security` folder and not from a classpath. You can override them, if needed.

Identity Provider Metadata

The `metadata.xml` file contains information about the identity provider. The content of this file is different for every identity provider, so this file isn't shipped with the `samlsinglesignon` extension.

Use the `sso.metadata.location` property to point out to the location of your identity provider's metadata file. If you don't configure it, the system uses the default location of `./WEB-INF/security/metadata.xml`.

Service Provider Metadata

Service provider metadata contains keys, services, and URLs defining SAML endpoints of your application. To ensure that your identity provider recognizes your service provider, provide a unique value of the `entityId` parameter using the `sso.entity.id` property. The default value is `urn:ssoextension:hybris:de`.

To obtain the metadata for the service provider, go to: `http://localhost:9002/samlsinglesignon/saml/metadata`.

ℹ Note

If you have set up SAP Commerce with the `samlsinglesignon` extension on a different machine, make sure to change the host and port in the URL.

SAML SSO Signing Certificate

The SAML SSO signing certificate is used to sign SAML requests, responses, and assertions.

Metadata files don't need to be digitally signed. However, when they're signed with a signing certificate, they require verification of signature's validity and trust. The signature is verified with a PKIX algorithm and uses all public keys that are present in the configured keystore.

ℹ Note

You can disable the verification process by using the `sso.metadata.verification.enabled` property with the `false` value. It is, however, not recommended due to security reasons.

In order for the signed metadata to work, you need to import the certificate from the identity provider into your keystore so that the signature can be verified. The `samlkeystore.jks` file is your default keystore. You can import certificates into your keystore by invoking the following command:

```
keytool -importcert -alias some-alias -file key.cer -keystore samlKeystore.jks
```

The name of the alias is not important as the signature is verified against all public keys in your keystore.

ℹ Note

In your production environment, don't use the `samlKeystore.jks` file that comes bundled with SAP Commerce. Its only purpose is to support local development and testing. Every environment requires a unique private key and certificate.

See the list of properties that you need to configure in your `local.properties` file for your keystore:

Property	Description	Default Value for samlKeystore.jks
`sso.keystore.location`	The location of your keystore file	`./WEB-INF/security/samlKeystore.jks`

Property	Description	Default Value for samlKeystore.jks
`sso.keystore.password`	The password to your keystore file	`changeit`
`sso.keystore.privatekey.password`	The password to your private key	`changeit`
`sso.keystore.privatekey.alias`	The private key alias	`hybris`
`sso.keystore.default.certificate.alias`	The default certificate alias	`hybris`

Group Level Authorization

The `samlsinglesignon` extension allows you to map a user group from the identity provider to SAP Commerce by defining a group mapping in the `local.properties` file or in the database through Backoffice.

To define a user group mapping in the `local.properties` file, use the following properties:

```
sso.mapping.<sso usergroup>.usertype=Employee
sso.mapping.<sso usergroup>.groups=employeegroup
```

To define a user group in your database at run time, use ImpEx or log in to Backoffice and navigate to System > Saml Mappings .

Note

The Backoffice configuration for the `SamlUserGroup` mapping is only available when the `samlssobackoffice` extension is active.

If you have defined group mappings in both the `local.properties` file and Backoffice, use the `sso.database.usergroup.mapping` property to switch between the mappings that you want to use. Use the value `false` for the `local.properties` file or `true` for Backoffice or ImpEx. By default, the property is set to `true`.

If an IDP user group isn't mapped to a relevant group on the SAP Commerce side in the `local.properties` file or the database, or the usergroup SAML custom assertion attribute isn't defined, users are not authenticated. As a result, users trying to log in are refused access to a given resource or service and the system displays an error page.

Related Information

Single Sign-On in Backoffice

Single Logout in SAP Commerce

Single Logout (SLO) allows users to log out from a single application and be automatically logged out from other applications.

Single Logout

The `samlsinglesignon` extension supports the following SLO scenarios:

- service provider initiated - when you log out of your application, the service provider sends the `saml2:LogoutRequest` message to the SLO service endpoint that is exposed by your identity provider. The identity provider then completes the logout and sends the `saml2:LogoutResponse` message back to the service provider.

- identity provider initiated - when you log out from your identity provider, the identity provider sends the `saml2:LogoutRequest` message to the SLO service endpoint that is exposed by your application. The service provider then completes the logout and sends the `saml2:LogoutResponse` message back to the identity provider.

i Note

Before using SLO, ensure that your identity provider supports it.

Single Logout Configuration

1. Make sure that you have correctly configured bindings for SLO services in your identity provider and in the metadata file that is delivered by the identity provider.
2. For the service provider initiated SLO - the `samlsinglesignon` extension exposes the logout filter endpoint under `saml/logout/**`. To enable SLO, ensure that any requests that are sent to this endpoint contain the `local` request parameter set to `false`, for example `/../samlsinglesignon/saml/logout?local=false`. To disable SLO, change this parameter to `true`.
3. For the identity provider initiated SLO - no additional configuration is required. The `samlsinglesignon` extension exposes the SLO service endpoint by default. The endpoint is configured to handle `saml2:LogoutRequest` requests.

For more information on bindings and endpoints, see Endpoints Configuration and Bindings.

Endpoints Configuration

SAML endpoints expose essential SAML services.

By default, the `samlsinglesignon` extension exposes the following legacy endpoints:

- `https://localhost:9002/samlsinglesignon/saml/SSO` - the assertion consumer service endpoint
- `https://localhost:9002/samlsinglesignon/saml/SingleLogout` - the SLO endpoint
- `https://localhost:9002/samlsinglesignon/saml/metadata` - the metadata generation endpoint

If you want to use the default endpoints shipped with the `spring-security-saml2-service-provider` library, set the `sso.legacy.endpoints.enabled` property to `false`. The default endpoints are:

1. `https://localhost:9002/samlsinglesignon/login/saml2/sso/{registrationId}` - the assertion consumer service endpoint
2. `https://localhost:9002/samlsinglesignon/logout/saml2/slo/` - the SLO endpoint
3. `https://localhost:9002/samlsinglesignon/saml2/service-provider-metadata/{registrationId}` - the metadata generation endpoint

Set the `{registrationId}` parameter using the `sso.relyingPartyRegistration.registrationId` property. The system uses the value of this property to create the `RelyingPartyRegistration` object. The value of this property can't be empty. Its default value is `registrationId`.

Signing Algorithms

Signing algorithms ensure that your service and identity providers exchange information in a secure manner.

Before sending the `saml2:LogoutRequest` message back to the identity provider, the service provider signs the `saml2:LogoutRequest` request with the configured private key and a signing algorithm. By default, SAP Commerce uses the following signing algorithms:

- `http://www.w3.org/2001/04/xmldsig-more#rsa-sha256`
- `http://www.w3.org/2009/xmldsig11#dsa-sha256`
- `http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256`
- `http://www.w3.org/2001/04/xmldsig-more#hmac-sha256`

To sign requests, the service provider uses the same algorithm as the one that has been used to encrypt the configured private key.

If your identity provider requires a different signing algorithm, use the `sso.signing.algorithms` property to configure it, for example:

```
sso.signing.algorithms=http://www.w3.org/2000/09/xmldsig\\#rsa-sha1
```

To set a value with the hash (#) symbol, precede the hash with a double backlash (\\). When configuring more than one algorithm, use a comma as a delimiter, for example:

```
sso.signing.algorithms=http://www.w3.org/2001/04/xmldsig-more\\#rsa-sha256,http://www.w3.org/2000/0
```

To use legacy algorithms with `sha1` as a hash algorithm, set the `sso.signing.algorithms` property as follows:

```
sso.signing.algorithms=http://www.w3.org/2000/09/xmldsig\\#rsa-sha1,http://www.w3.org/2000/09/xmlds
```

Assertion Attribute Mapping

A SAML Assertion is an XML file that contains attributes with information about users logging through SSO.

The identity provider sends an SAML Assertion file to the service provider that uses this file to confirm the identity of the user who tries to log in using SSO. The `samlsinglesignon` extension allows you to use the following properties to map SAML Assertion attributes:

- `sso.usergroup.attribute.key` - the user name attribute in SAML Assertion
- `sso.firstname.attribute.key` - the first name attribute in SAML Assertion
- `sso.lastname.attribute.key` - the last name attribute in SAML Assertion
- `sso.userid.attribute.key` - a user identifier, for example an e-mail address in SAML Assertion
- `sso.language.attribute.key` - the language attribute that can be set to determine the application language in SAML Assertion

Bindings

Bindings are mechanisms that transport SAML messages between the service provider and the identity provider.

Bindings are responsible for:

- sending the `saml2:AuthnRequest` message to the identity provider and receiving the `saml2:Response` message from the identity provider in the SSO process
- sending the `saml2:LogoutRequest` message to the identity provider and receiving the `saml2:LogoutResponse` from the identity provider in the service provider initiated SLO process
- receiving the `saml2:LogoutRequest` from the identity provider and sending the `saml2:LogoutResponse` message to the identity provider in the identity provider initiated SLO process

In your identity provider configuration, you can determine which bindings are used to send SAML messages.

The `samlsinglesignon` extension supports the following bindings:

- `POST` and `Redirect` - used for sending the `saml2:AuthnRequest` message in the SSO process. When deciding whether the `POST` or the `Redirect` binding should be used, the system checks the order of the `SingleSignOnService` elements in the metadata file and selects the first one defined there.
- `POST` - used for receiving the `saml2:Response` message in the SSO process.
- `POST` and `Redirect` - used for sending the `saml2:LogoutRequest` message in the service provider initiated SLO process and the `saml2:LogoutResponse` message in the identity provider initiated SLO process. When deciding whether the `POST` or the `Redirect` binding should be used, the system checks the order of the `SingleSignOnService` elements in the metadata file and selects the first one defined there.
- `POST` and `Redirect` - used for receiving the `saml2:LogoutRequest` message in the identity provider initiated SLO process and the `saml2:LogoutResponse` in the service provider initiated SLO process. The `sso.single.logout.binding` property with the default value of `urn:oasis:names:tc:SAML:2.0:bindings:HTTP-POST` determines which binding is used. If you want to use the `Redirect` binding, use the `urn:oasis:names:tc:SAML:2.0:bindings:HTTP-Redirect` value.

Azure Active Directory as Identity Provider

When using Azure Active Directory as your identity provider, make sure that you have configured your service provider metadata file.

To ensure that there are no errors related to reading XML files containing the same namespaces with different prefixes, follow one of the steps when uploading your service provider metadata file to the Azure Active Directory identity provider:

- Add the `md` prefixes to the `EntityDescriptor` elements in your metadata file, for example:

```
<md:EntityDescriptor xmlns:md="urn:oasis:names:tc:SAML:2.0:metadata" entityID="urn:ssoextensi
...
</md:EntityDescriptor>
```

- Remove the `md` prefixes from all elements in your metadata file
- Add the following values from the service provider metadata file to the Azure Active Directory identity provider:
  - for Identifier (Entity ID) - the value of the `entityId` attribute from the `<EntityDescriptor.../>` element
  - for Reply URL (Assertion Consumer Service URL) - the value of the `Location` attribute from the `<AssertionConsumerService.../>` element
  - for Logout URL - the `Location` value from the `<SingleLogoutService.../>` element

Single Sign-On for Back-End Applications

Platform provides an SSO support for back-end applications and implements it for SAP Commerce Administration Console.

Single sign-on for back-end applications depends on the `samlsinglesignon` extension and the Spring Security Remember Me functionality. To benefit from Spring Security Remember Me, make sure that the properties defining cookie names for SSO and login token have the same value:

```
login.token.name=LoginToken
sso.cookie.name=LoginToken
```

When the extension is properly configured, it is required to set the login redirection URI property:

```
hac.login.singlesignon.redirect=/samlsinglesignon/saml/
```

In this case `hac` is placed under the `/` webroot.

If you place `hac` under `/hac` webroot, the property value would be:

```
hac.login.singlesignon.redirect=/samlsinglesignon/saml/hac
```

The value of this property along with the value of the `sso.redirect.url` property points to the URL where the user is redirected after successful authentication with the SSO identity provider.