



Platform

Generated on: 2021-10-25 17:10:52 GMT+0000

SAP Commerce | 1811

PUBLIC

Original content: <https://help.sap.com/viewer/d0224eca81e249cb821f2cdf45a82ace/1811/en-US>

Warning

This document has been generated from the SAP Help Portal and is an incomplete version of the official SAP product documentation. The information included in custom documentation may not reflect the arrangement of topics in the SAP Help Portal, and may be missing important aspects and/or correlations to other topics. For this reason, it is not for productive use.

For more information, please visit the <https://help.sap.com/viewer/disclaimer>.

Working with Catalogs

Platform catalogs hold, structure, and manage products and product information.

Catalogs offer the following features and functionality:

- **Structuring Your Collection with Products and Categories:** Products are the basic elements of each catalog. By grouping them, you can arrange your collection into categories. In order to build a hierarchical product structure, products can be kept in categories. For more information, see [Structuring Your Collection with Products and Categories](#).
- **Catalogs and Catalog Versions:** Categories are held in catalog versions. A catalog version can represent your collection of products at a certain point in time. To enable you to maintain your collection within a basic process, you usually contain two or more catalog versions in an object called a **catalog**. For example, you can have one catalog version for editing the content (the **Staged** version) and you can have another catalog version for propagation as a web shop (the **Online** version). For more information, see [Catalogs and Catalog Versions](#).
- **Managing Multiple Product Catalogs in a Catalog System:** A catalog system is a group of SAP Commerce catalogs. It represents a framework for managing multiple output catalogs enabling you to maintain the varying content of several product catalogs. A catalog system can also include multiple input catalogs. Typically, a catalog system is set up around a master catalog that contains the leading versions of product data. The different catalog contents are synchronized among defined catalog versions, thus reflecting your catalog maintenance process. For more information, see [Managing Multiple Product Catalogs in a Catalog System](#).
- **Synchronizing Catalog Versions:** Catalog versions can be synchronized in a one-directional way in order to update a catalog version based on another one. This enables you to transfer catalog content between the catalog versions that represent steps in your catalog maintenance process. You can synchronize catalog versions, even if they are from the same or different catalogs within your SAP Commerce. For more information, see [Synchronizing Catalog Versions](#).
- **Defining Product Attributes:** Product attributes specify properties of products. They are used to describe a product and to hold related information. A product attribute has a name and a value. For example, the value of the attribute `color` is `blue`; SAP Commerce provides different modeling methods for defining product attributes: typing and classification. For more information, see [Defining Product Attributes](#).
- **Classification Guide:** The classification functionality enables you to define product attributes in a way different to the typing method. Classification-based attributes are called `category features`; they can also sometimes be referred to as `classification attributes`. Through classification, you can flexibly allocate category features that may change frequently. You can easily define and modify them because you manage them independently of the product type.

Available category features can be organized independently from product catalog structures in separate `classification systems`. Here they can be structured into `classifying categories`. Through such classifying categories, you can assign a category feature to one or multiple product categories used in catalogs. That means that all category features of the classifying categories are available within all products included in the assigned product categories. Therefore, each category feature assigned to a category of a catalog structure is inherited by all subcategories. For more information, see [Classification](#).

Synchronizing Catalogs

Catalog versions can be synchronized in a one-directional way for updating a catalog version according to another. This way you can transfer catalog content between the catalog versions which represent steps in your catalog maintenance process. You can synchronize catalog versions, even if they are of the same or different catalogs within SAP Commerce.

The catalog that is online, and therefore visible to customers, is often not the only catalog that your platform contains; usually, you have several non-visible catalogs as well that you can work with. This has the advantage in that you can assemble several catalogs in advance so you can use them when needed, for example, when seasons change and your product portfolio needs to match that.

For transferring the differing content between two catalogs, you use synchronizing operations among subordinate catalogs versions.

i Note

From a technical perspective, there is no difference between catalogs or catalog versions. For example, the online catalog is different from the catalog that is hidden only because of access rights.

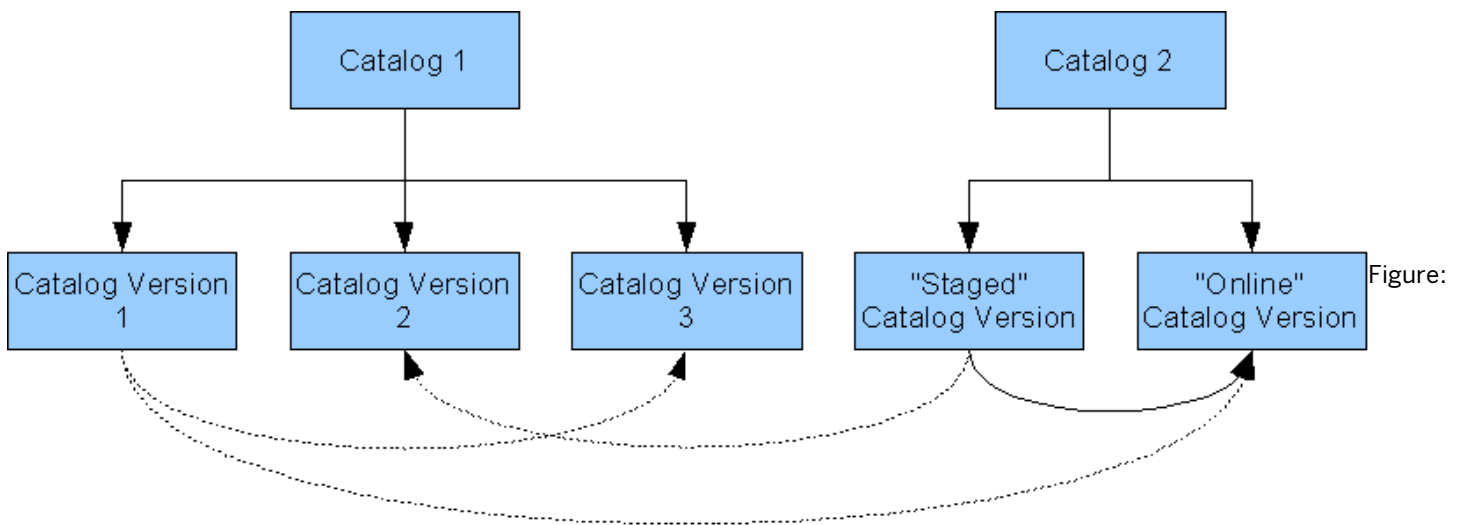
How Does Synchronizing Work?

The single catalog that is visible to your customers is called **online**, and the catalogs that are not visible to customers are **staged**. A standard synchronizing operation is the act of updating the online catalog with at least one of the staged catalogs. If you synchronize a staged catalog, the online catalog will then contain all values from that staged catalog.

A synchronizing operation can be applied to an entire catalog version, but it can also be applied to selected categories or products. Synchronization works in a unidirectional manner only, from staged to online. However, you can apply two synchronizing operations to establish bi-directional update procedures.

i Note

You can also change access rights of the staged catalog to be the online catalog. However, SAP Commerce does not recommend this solution because you will not have a backup of your online catalog. Synchronizing catalogs is easy and can be fully automated.






This illustration shows a sample catalog system consisting of **Catalog 1** with three versions, and **Catalog 2** with two versions. You may want such a setup if, for example, you have several suppliers who have product catalogs of their own. The dotted lines represent customized synchronizations, and the solid line represents the default synchronization:

Default:Staged Default:Online .

Synchronization Statuses

When you search for items in Backoffice, for example products, they display with a graphic symbol next to each product name. Each such symbol indicates a current product synchronization status.

- : Indicates that a source item is synchronized with the target item.
- : Indicates that a source item has been changed and needs synchronization with the target item.
- : Indicates that an initial synchronization is needed. The source catalog version has never been synchronized with the target catalog version. Since synchronizing single items is disabled, you need to synchronize an entire catalog version.

10/25/2021

Each synchronization execution produces synchronization timestamps. The logic evaluating the **Initial synchronization needed** status checks to see if any timestamps exist for the source catalog version. In other words, it checks if the catalog has ever been synchronized.

It is possible to change the logic by introducing the following configuration entry:

```
catalog.synchronization.initialinit.check.timestamps=false
```

In this case, the status will be evaluated based on the executions of synchronizations as Cronjobs.

i Note

If you set the **catalog.synchronization.initialinit.check.timestamps=false** and clear the historical synchronization Cronjob execution items from your system, you will see the 🚧 status for your catalog version.

See the following topic for more information: *No Changes in a Synchronized Catalog Version After the Synchronization Was Modified?*.

About Synchronizations

A synchronizing operation is held in a `synchronization` object referencing the source and the target object. Thus, synchronizations are definitions of how products and their data such as names, descriptions, and prices will be copied into the target catalog.

You can create, edit, and delete a synchronization using Backoffice. Each synchronization is bound to the catalog version that is used as the synchronization source. For more information, see [Editing Synchronization Rules](#).

i Note

To support systems with a large number of products, SAP Commerce supports multithreaded catalog synchronization operations. For details, see [Performing Multithreaded Catalog Synchronization in Backoffice](#).

i Note

The synchronization logic is designed to manage deadlocks resulting from synchronization being a highly-concurrent process.

Performing Synchronization

Backoffice enables you to:

- edit synchronization rules
- synchronize complete catalog versions or chosen items
- check the impact of synchronization

You can find more explanation in separate synchronization topics.

No Changes in a Synchronized Catalog Version After the Synchronization Was Modified?

After you have become familiar with how synchronization works, you will probably want to experiment with synchronizations. If so, you may run into the following situation: You have synchronized a catalog version, modified the synchronizations, and after launching the synchronizations again, the target catalog version is the same as before running the edited synchronizations. For

example, the original synchronizations updated the product codes, whereas the new synchronizations updated their prices. However, the target catalog version still only contains the codes. This seems to be a bug, but, in fact, it is not. It has to do with the way the catalog selects the products to synchronize. Whether a product is considered to be new and therefore, synchronized or not, depends on the last point of time when the product was changed. The platform itself calls this the modification timestamp.

Once a product is synchronized with a certain synchronization, that timestamp is stored with that rule. Consequently, the product is considered up-to-date in the target catalog. Unless the product's last change date changes, that product will not be synchronized any more with that rule, even if the rule was edited. The idea behind this is that the platform can avoid synchronizing up-to-date products, which speeds up the synchronizing operation tremendously. In the case that the synchronization has changed, however, this commonly useful mechanism fails. It is technically possible to find out if a synchronization has changed since its last run, but due to all of the checks and database accesses that are necessary, performance would be greatly impacted.

If you want to try an edited synchronization effect with a previously synchronized product, you need to manually edit that product. For example, change the product description, and then save the changed product. The platform will then set the modification timestamp to the current date and synchronize this product again on the synchronization's next run. After this edit and change, you can directly re-edit your product to its previous state. For example, add a letter to the description, save it and then remove that letter from the description again and save. That way, the timestamp will be updated, while your product stays the same. You might think that simply opening the product's editor and saving it will change the timestamp, but it does not because the system recognizes that the product has not changed. It does not save the product and therefore, the timestamp does not change either.

Synchronization of Translated Collection Elements

A warning message appears if an element cannot be translated, but the synchronization continues. Additionally, there is a new flag to control the default behavior `synchronization.allow.partial.collection.translation`.

This property defines what happens with the synchronization of a collection of references if the sync process cannot translate all the references in the source collection.

`synchronization.allow.partial.collection.translation=true` (default behavior) means that if the synchronization fails to translate any of the source references, it skips translation of that reference with a warning message and continues translation of the root item. The result is the target collection only contains those target references that were successfully translated.

`synchronization.allow.partial.collection.translation=false` means that if the synchronization fails to translate any of the source references, it aborts synchronization of the root item with an error message and any pending changes to the root item are rolled back.

The `partiallyTranslatable` in `SyncAttributeDescriptorConfig` allows you to set up a synchronization job in ImpEx and override the global behavior for a given synchronization job.

Configuring Synchronization as CronJobs

A CronJob is a task that runs automatically at a certain point of time for a given number of times. For example, CronJobs may create backups every day and store them on remote drives on a certain day of the week, or rebuild indexes for the search engine every other day at a certain time.

Synchronizations are CronJobs as well, so you may have the respective catalog versions being synchronized automatically at a certain time. The idea behind this is that these synchronizations may cause heavy load on your servers and you should run them at times when the system load is usually low, such as early morning or on Sundays. SAP Commerce enables running synchronization processes independently, without a related CronJob. For details on CronJobs, refer to [The Cronjob Service](#).

Configuring Synchronization for Catalog Systems

A catalog system is group of SAP Commerce catalogs connected by synchronizing operations that are realized between the contained catalog versions. You organize the involved catalogs in a way to establish a synchronization process from input to output. Thus, a catalog system represents a framework for managing multiple output catalogs enabling you to maintain the varying content of several product catalogs. In addition, a catalog system can handle multiple input catalogs. Typically, a catalog system is set up around a master catalog holding the leading versions of product data.

See also: [Managing Multiple Product Catalogs in a Catalog System](#)

Example of a catalog system:

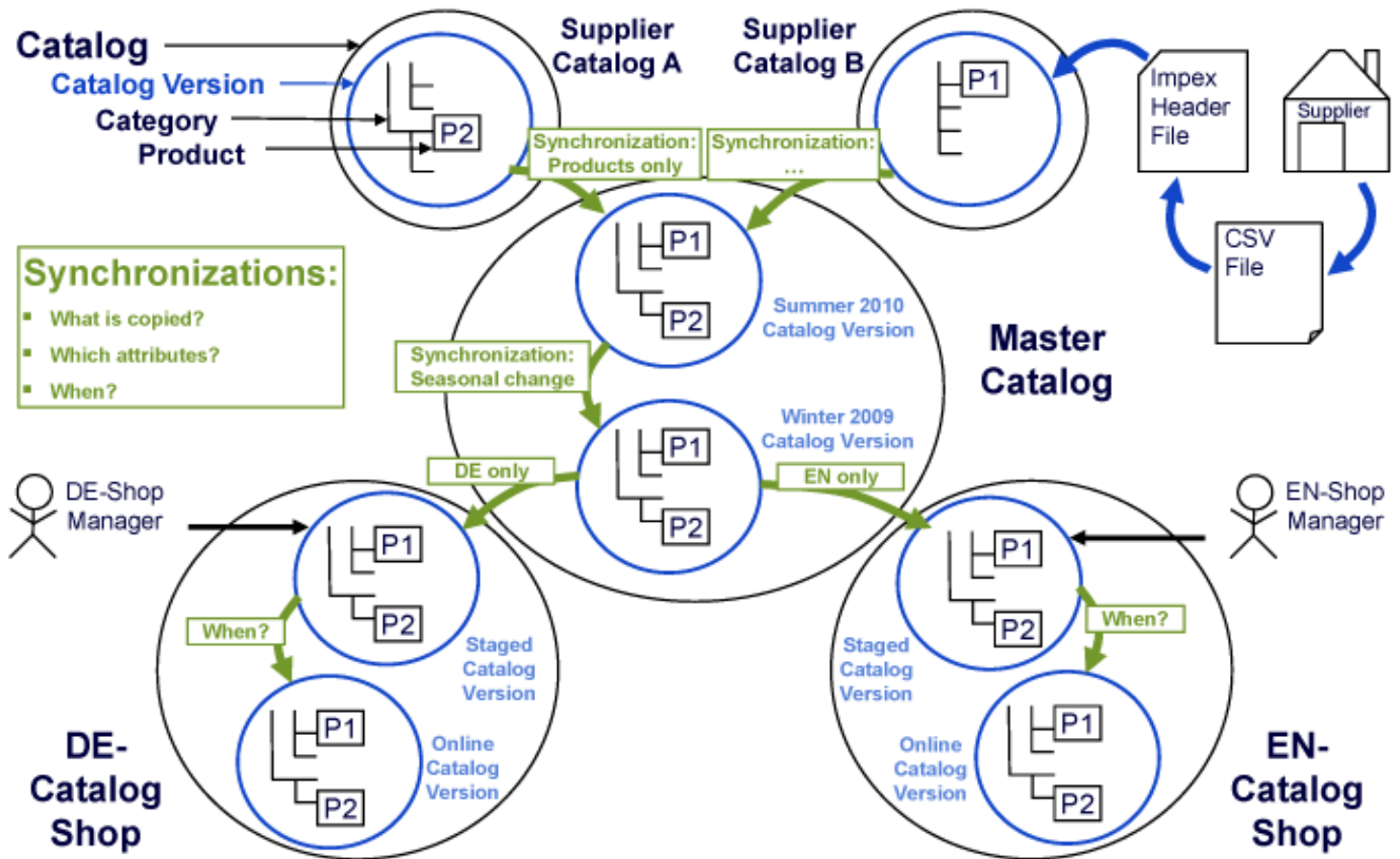


Figure: This sample catalog system consists of two supplier catalogs (A and B), a **Master Catalog**, and two language-specific output catalogs; **DE-Catalog Shop** and **EN-Catalog Shop**. The master catalog and the output catalogs each consist of two catalog versions to distinguish between editing stages and final stages. While the current catalog version is Winter 2009, the next issue, Summer 2010 is prepared in its own catalog version. In the output catalogs, the contained catalog versions are used to separate country-specific editing and approval activities from the propagated final **Online** catalog version. The catalog versions in this catalog system are connected by synchronizations, indicated by green arrows that configure the content transfer between catalog versions.

Synchronization with ServiceLayer

You can use the ServiceLayer adapter for synchronization jobs. The immediate benefit is that you can use the ServiceLayer infrastructure such as interceptors and validators. To enable the ServiceLayer mode, place the following property in your `local.properties` file:

```
synchronization.legacy.mode=false
```

You can also change this dynamically in SAP Commerce Administration Console. Go to the **Platform tab Configuration**.

For more information, see the following:

- [ServiceLayer](#)
- [Configuring the Behavior of SAP Commerce](#)
- [Administration Console](#)

Editing Synchronization Rules

Backoffice allows you to edit synchronization rules by using synchronization editor.

Context

To edit synchronization:

Procedure

1. Log into Backoffice and navigate to **System Multithreaded Synchronization**.

The result view displays available synchronization jobs.

2. Choose the synchronization job you wish to modify to expand a synchronization editor.

The screenshot shows the SAP Backoffice interface for editing synchronization rules. On the left is a dark sidebar with a 'Filter Tree entries' box and a navigation menu. The main area has a search bar at the top and a table of synchronization jobs. One job, 'Sync Default:Staged -> Default:Online', is selected and highlighted. Below the table, the 'CONFIGURATION' tab is active, showing the 'ESSENTIAL' section with input fields for the code, source, and target.

Filter Tree entries

- System
 - OAuth
 - CORS Filter
 - Advanced Configuration
 - Tools
 - Output Documents
 - Workflow Administration
 - Validation
 - Scripting
 - Business Processes
 - Background Processes
- Types
 - Saved Queries
 - Backoffice Saved Queries
- Report Definitions
- Personalization
 - System Setup Audit
- Audit Report Config
- Multithreaded Synchronization

Search

<input checked="" type="checkbox"/> Code	Synchronization source	Synchronization target
<input checked="" type="checkbox"/> Sync Default:Staged -> Default:Online	default catalog : Staged	default catalog : Online

1 ITEMS SELECTED

Sync Default:Staged -> Default:Online

CONFIGURATION CRONJOBS RESTRICTIONS ADVANCED ATTRIBUTES ADMINISTRATION

ESSENTIAL

Code	Synchronization source	Synchronization target
Sync Default:Staged -> Default:Online	default catalog : Staged	default catalog : Online

3. On the **CONFIGURATION** tab in the editor area, click **Edit Configuration** button.

Sync Default:Staged -> Default:Online

CONFIGURATION CRONJOBS RESTRICTIONS ADVANCED ATTRIBUTES ADMINISTRATION

ESSENTIAL

Code	Synchronization source	Synchronization target
Sync Default:Staged -> Default:Online	default catalog : Staged	default catalog : Online

SYNCHRONIZATION ATTRIBUTES CONFIGURATION

Synchronization attributes configuration

Included Properties 31 Types 224 Attributes	Excluded Properties 0 Types 0 Attributes
--	---

[Edit Configuration](#)

Browse the tree to expand available properties. Any selected property is updated in the target catalog, and any unselected properties keep their values. This enables you to merge content from several catalog versions into a single one, and split them into individual catalogs again.

Edit Synchronization Attributes Configuration

Search

☒ Item [Item]

☒ Comments [comments]

☒ Documents [allDocuments]

☒ Owner [owner]

☒ AbstractMedia [AbstractMedia]

☒ Category [Category]

☒ Category Feature [ClassAttributeAssignment]

☒ Configured product [AbstractConfiguratorSetting]

☒ [configurationCategoryPOS]

☒ Configurable category [configurationCategory]

☒ Configurator type [configuratorType]

☒ Identifier [id]

☒ Qualifier [qualifier]

☒ Feature Descriptor [ClassificationAttribute]

☐ Synchronize
☐ Copy by value
☐ Untranslatable value
☐ Partially translatable

☐ Attribute
 ☒ Completely Included
 ☐ Partially Included
 ☒ Not Included

CANCEL SAVE

4. On the **ADMINISTRATION** tab, set the synchronization rules of how product data is copied to the target catalog version:

- Basic Setting:** You can decide whether missing objects in the target catalog version should be created. Furthermore, you can decide that existing products in the target catalog version that do not exist in the source catalog version should be removed. Alternatively, only products that exist in both the source and the target catalog versions are synchronized.
- Language Settings:** Define the languages of products attributed to be synchronized.
- Synchronization User/Groups:** Define users and groups that are permitted to initiate such a process although they don't have such write permissions.

If the option, **Respect sync. permissions only** is checked, only the defined sync. users are allowed to synchronize. If it is not checked, then a user with granted write permissions for the target catalog version is allowed to do so, too.

- **Type Settings:** Set the root types that are analyzed in case of complete synchronizations of catalog versions. Such synchronizing procedures searches for new, changed, updated, and removed items of the listed types.

Checking the Impact of Synchronization

Backoffice allows you to look up differences in the content of your catalog versions. The **Catalog Version Diff** action displays items that need to be synchronized. You may want to use this function to see the differences before synchronization and a synchronization result afterwards.

Context

A list of differences between two versions of a catalog is generated based on a synchronization job that binds the two versions of the catalog. The synchronization job configuration determines the items that are going to be synchronized. As a result, only the items provided in the configuration are taken into consideration when you generate the list of differences between your catalog versions.

In case given items have been already synchronized, generating a list of differences returns no result.

To better understand the context of this topic, get familiar with the synchronization example and its prerequisites described in [Synchronizing Catalog Versions in Backoffice](#).

To display a list of differences between catalog versions, log in to Backoffice and follow these steps:

Procedure

1. In the Backoffice navigation tree, click **Catalog Catalog Versions**.

A tab with a list of available catalog versions shows up.

2. Click a catalog version.

In our case, it is the Staged version.

Information about the catalog version displays, with action icons available for that catalog version, including the **Catalog Version Diff** action icon.

SEARCH

2 items

<input type="checkbox"/> Catalog	Catalog Version	is active catalog version
<input checked="" type="checkbox"/> default catalog	Staged	false
<input checked="" type="checkbox"/> default catalog	Online	true

0 ITEMS SELECTED

default catalog : Staged

REFRESH

SAVE

CATALOG VERSIONCONTENTCATALOG VERSIONSBMECATPERMISSIONSADMINISTRATION

ESSENTIAL

Catalog

Catalog Version

default catalog

Staged

3. Click the **Catalog Version Diff** icon.
- The **Catalog Version Diff** window opens.
- Notice that for the Staged catalog version, a synchronization job is already chosen. It is Sync Default:Staged -> Default:Online.
4. **Optional:** Choose a particular synchronization job.
5. Click **Generate**.
- A list displays with items that haven't been synchronized.

Catalog Version Diff

Select synchronization job: Sync Default:Staged -> Defa

GENERATE

Type	Source	Last modification of source	Last synchronization
Product	001 [product001] - default catalog : Staged	January 3, 2018 3:08:12 PM CET	
Product	002 [product002] - default catalog : Staged	January 3, 2018 3:08:36 PM CET	
Product	003 [product003] - default catalog : Staged	January 3, 2018 3:09:00 PM CET	

Results

You have displayed a list of items that require to be synchronized.

Synchronizing Catalog Versions in Backoffice

Backoffice allows you to synchronize entire catalog versions. To synchronize your catalogs, use the **Catalog synchronization** action.

Context

The example synchronization procedure we perform below is based on a few prerequisites:

- We perform a full synchronization between the Staged and Online versions of the default catalog. The catalog is provided with SAP Commerce by default.
- Since both versions of the catalog are empty, we created a few products (product001, product002, product003) that are the items that we synchronize. The products are created for the Staged version of the catalog.
- Our synchronization example uses the Sync Default:Staged -> Default:Online job that binds both versions of the catalog. The job defines the Staged version as the source catalog, and the Online version as the dependent (target) catalog. This relation determines the direction of synchronization - items are synchronized from the source to the target. The job is provided with SAP Commerce by default..

Before you start synchronization, you may want to see the difference in the content of the catalogs you want to synchronize. For more information, see [Checking the Impact of Synchronization](#).

To synchronize catalogs, log in to Backoffice and follow these steps:

Procedure

1. In the Backoffice navigation tree, click **Catalog Catalog Versions**.

A tab with a list of available catalog versions shows up.

2. Click the source version of the catalog you want to synchronize.

In our case, it is the Staged version of default catalog.

A tab shows up with information about the catalog as well as available action icons, including the **Catalog synchronization** icon.

2 items

<input type="checkbox"/> Catalog	Catalog Version	is active catalog version
<input checked="" type="checkbox"/> default catalog	Staged	false
<input checked="" type="checkbox"/> default catalog	Online	true

0 ITEMS SELECTED

default catalog : Staged

[CATALOG VERSION](#)
[CONTENT](#)
[CATALOG VERSIONS](#)
[BMECAT](#)
[PERMISSIONS](#)
[ADMINISTRATION](#)

ESSENTIAL

Catalog

default catalog

Catalog Version

Staged

CATALOG VERSION

Generation date

is active catalog version

☐ True
☒ False

Languages

German [de]

English [en]

Territories

3. Click the **Catalog synchronization** icon.

The **Catalog synchronization** wizard opens up.

4. In the **Sync job selection** tab of the wizard, choose a synchronization job .

In our case, it is Sync Default:Staged -> Default:Online.

Catalog synchronization



CATALOG SYNCHRONIZATION
Sync Job selection

CATALOG SYNCHRONIZATION
Advanced configuration

CATALOG SYNCHRONIZATION
Results

Synchronization Job:

Sync Default:Staged -> Default:Online

CANCEL

NEXT

- Click **Next** to move to the **Advanced configuration** tab and adjust your configuration if required.

We don't change anything at this step.

- Click **Start** in the **Advanced configuration** tab to perform the synchronization.

Synchronization results display when the process has finished.

- Click **Done** to close the wizard window.

Results

You have performed synchronization.

Now you can check the synchronization result. For more information, see [Checking the Impact of Synchronization](#). As another option, you can check the status of the items you synchronized. For more information, see the Synchronization Statuses topic in [Synchronizing Catalogs](#).

Synchronizing Items

Backoffice enables you to synchronize selected items. To synchronize your items, use **Synchronize Action** icon.

Context

The following example synchronization procedure is based on a few prerequisites:

- We perform a synchronization of items from the Staged to the Online version of the default catalog.
- We created three products (product001, product002, product003) and we synchronize two of them (product001, product002). The products are assigned to the Staged version of the catalog.

For more information about creating a product, see section Creating a Product in [Editing Tax and Discount Values of an Order in Backoffice](#).

- Our synchronization example uses the Sync Default:Staged -> Default:Online job that binds both versions of the catalog. The job defines the Staged version as the source catalog, and the Online version as the dependent

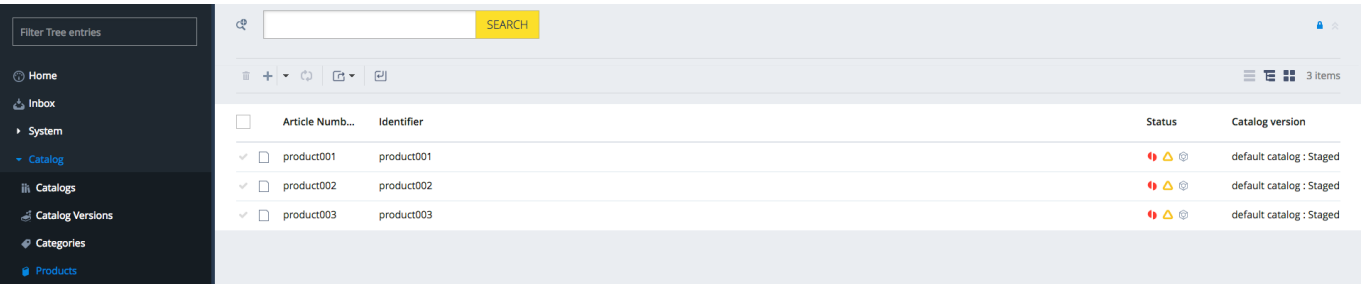
(target) catalog. This relation determines the direction of synchronization - items are synchronized from the source to the target. The job is provided with SAP Commerce by default.

To synchronize items, log in to Backoffice and follow these steps:

Procedure

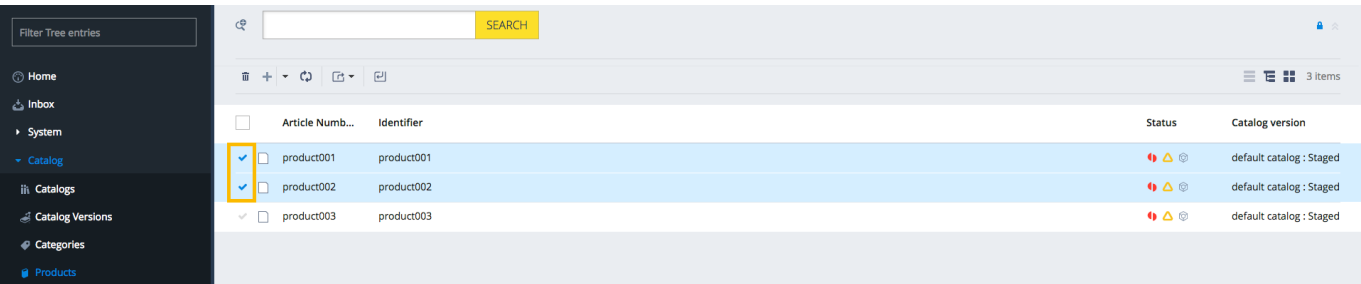
- 1. In the Backoffice navigation tree, click **Catalog Products**.

A tab with a list of available products shows up.

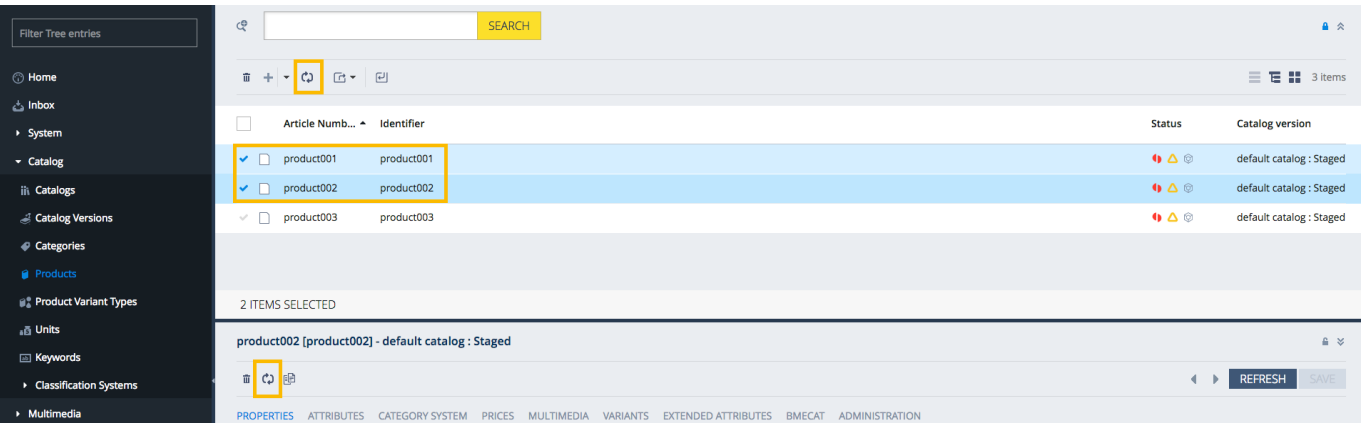


- 2. Choose the products you want to synchronize.

In our case, they are product001 and product002.



A tab shows up with information about the product as well as available action icons, including the **Synchronize Action** icon.

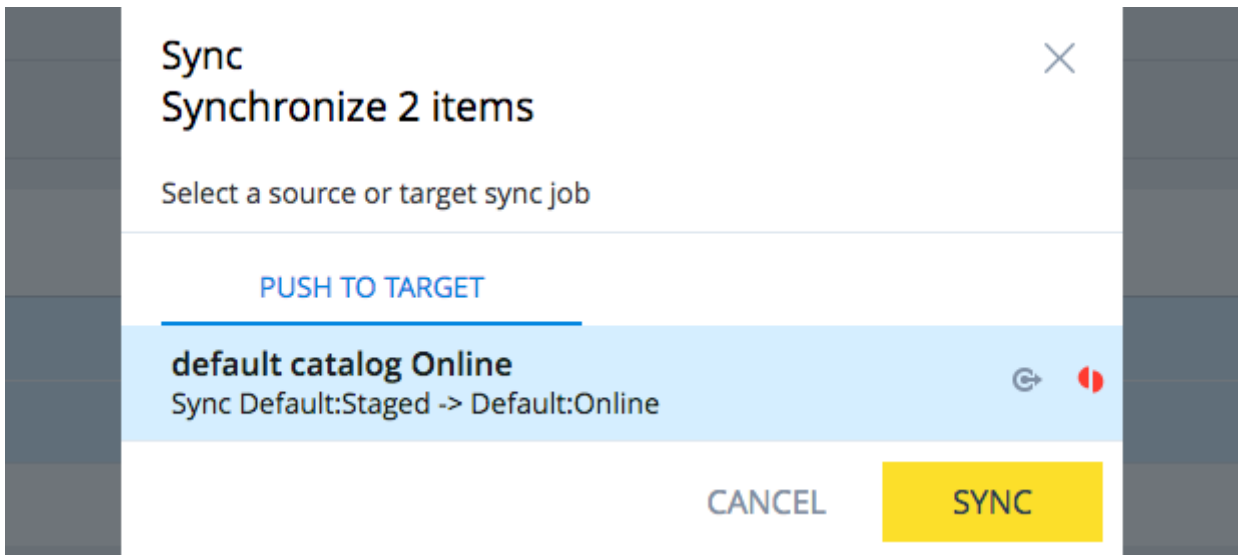


- 3. Click the **Synchronize Action** icon.

The **Synchronize Action** wizard opens up.

- 4. In the **Synchronize Action** wizard, choose a synchronization job and click **SYNC**.

In our case, it is Sync Default:Staged -> Default:Online.



The synchronization process is completed. You can look up the items in the collection browser and see their synchronization statuses.

Filter Tree entries

Home

Inbox

System

Catalog

Catalogs

Catalog Versions

Categories

Products

🔍

SEARCH

🔍

5 Items

<input type="checkbox"/>	Article Numb... ^	Identifier	Status	Catalog version
✓ <input type="checkbox"/>	product001	product001	<div><div>🟢</div><div>⚠️</div><div>🔄</div></div>	default catalog : Staged
✓ <input type="checkbox"/>	product001	product001	<div><div>🟢</div><div>⚠️</div><div>🔄</div></div>	default catalog : Online
✓ <input type="checkbox"/>	product002	product002	<div><div>🟢</div><div>⚠️</div><div>🔄</div></div>	default catalog : Staged
✓ <input type="checkbox"/>	product002	product002	<div><div>🟢</div><div>⚠️</div><div>🔄</div></div>	default catalog : Online
✓ <input type="checkbox"/>	product003	product003	<div><div>🔴</div><div>⚠️</div><div>🔄</div></div>	default catalog : Staged

Results

You have performed the synchronization of your items.

For more information, see [Checking the Impact of Synchronization](#). As another option, you can check the status of the items you synchronized. For more information, see the Synchronization Statuses section in [Synchronizing Catalogs](#).

Dependent Synchronization

Dependent synchronization avoids creation of cross-references between items from **Staged** and **Online** catalog versions that are a part of different catalogs.

Overview of Dependent Synchronization

It is recommended to use dependent synchronization in the following situations:

- You have multiple catalogs with Staged and Online catalog versions
- These catalogs have items that refer to each other
- These catalogs have synchronization jobs to synchronize their catalog versions

The following diagram illustrates an example with two different catalogs:

- **Categories Catalog:** It contains the category structure for the Web shop. There are two catalog versions:
 - **Categories Catalog Staged Version:** Here you create the structure of catalogs. Then you decide which product is shown in which category, that is to say, you match a category to a product. In your catalogs structure products are held in the Product Catalog. These changes need to be synchronized with the **Categories Catalog Online Version** to make the changes visible in the Web shop.

- **Categories Catalog Online Version:** After synchronization this catalog version should reflect data from the **Categories Catalog Staged Version**.
- **Products Catalog:** It is fed by the SAP system. There are two catalog versions:
 - **Products Catalog Staged Version:** It contains all available products with all updated attributes, for example price. You decide which catalog should be available online and for how long. From this point you can also refer a product to a category. All changes are synchronized with the **Products Catalog Online Version**.
 - **Products Catalog Online Version:**

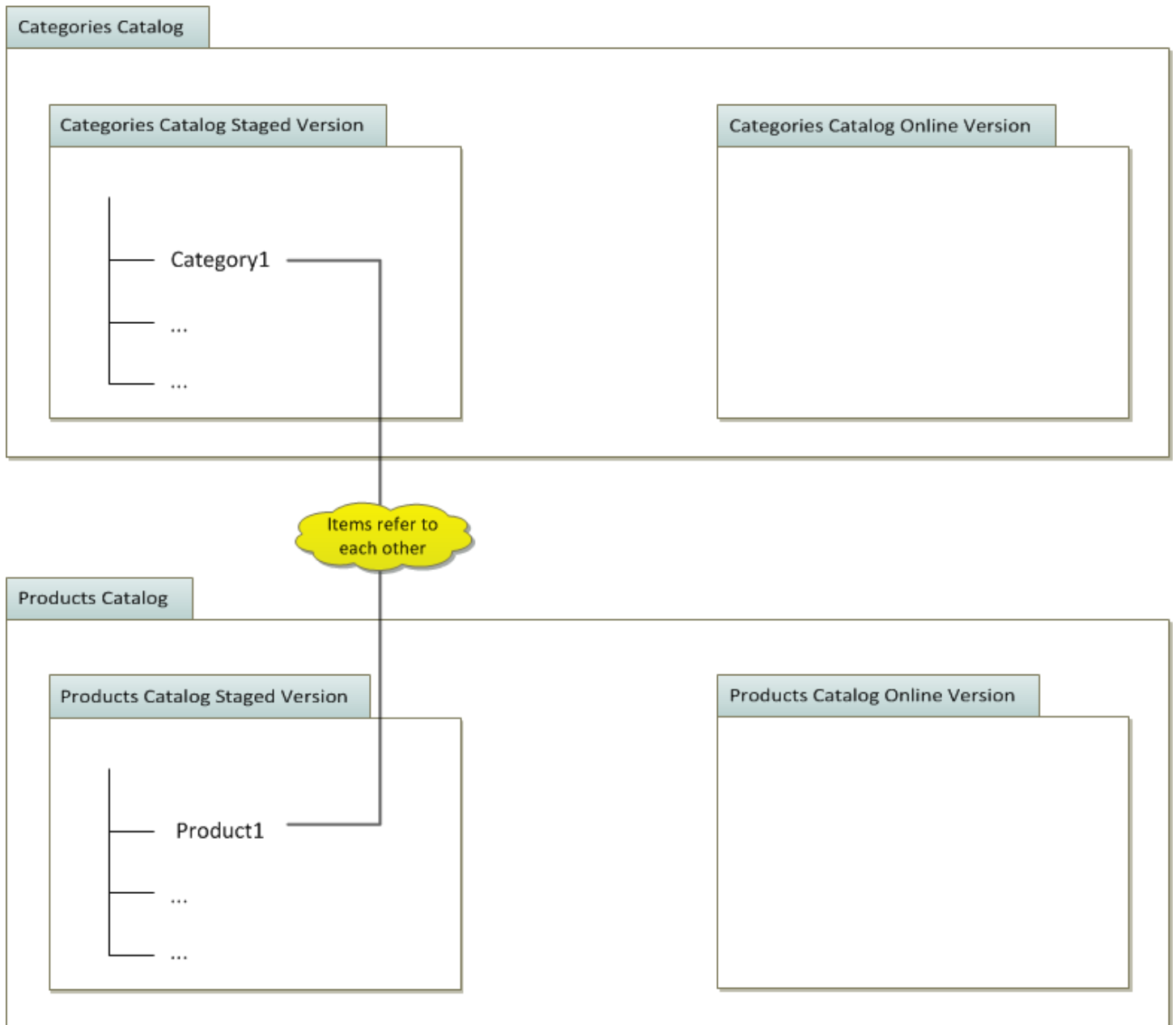


Figure: Catalogs before synchronization.

Synchronization without Dependency

If synchronization jobs are not aware of each other, then cross-references between items from different catalogs are created.

The following diagram provides an example with the same catalogs shown in *About Dependent Synchronization* section after running an independent synchronization:

1. **Categories Catalog Synchronization:** Category1, from the Categories Catalog Staged Version, is copied within the same Categories Catalog to the Categories Catalog Online Version. Category1 contains a reference to Product1 in the Products Catalog Staged Version, which is a part of the Product Catalog. Because of the relation, this reference is also copied. As a result, after the synchronization, Product1 refers to two categories: stages and online.

2. **Products Catalog Synchronization:** Product1 from the Products Catalog Staged Version is copied within the same Products Catalog to the Products Catalog Online Version. As Product1 already had two references to two different categories (staged and online), those references are also copied.

After both synchronization jobs are finished, items from online versions of catalogs do not only hold references to each other, but also hold cross-references to the staged catalog versions. The situation would look exactly the same if synchronization jobs ran in a different order.

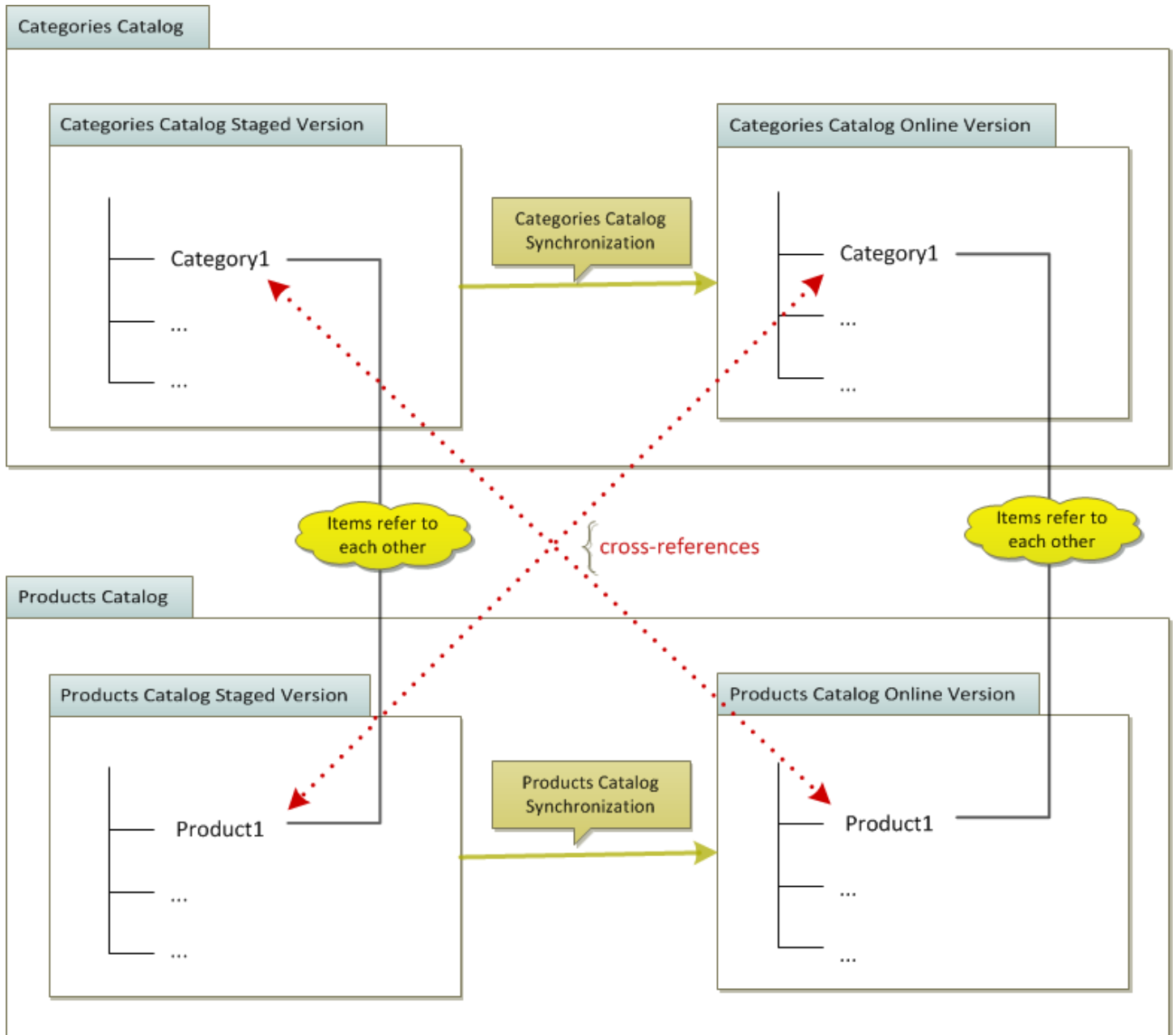


Figure: Catalogs after independent synchronization.

Synchronization with Dependency

If synchronization jobs are dependent on each other, cross-references between items from different catalogs are not created. Setting a dependency between synchronization jobs does not mean that these synchronizations jobs trigger each other.

The diagram below illustrates an example with the same catalogs shown in the Dependend Synchronization section, after running the dependent synchronization:

1. Categories Catalog Synchronization:

- Category1, from the Categories Catalog Staged Version, is copied within the same Categories Catalog to the Categories Catalog Online Version.

- b. As Category1 contains a reference to Product1, from the Products Catalog Staged Version, which is a part of the Product Catalog, it is checked if this synchronization job is dependent or depends on synchronization job from the Product Catalog.
- c. The dependency is configured, the Products Catalog Online Version is checked for the online version of Product1. As the Products Catalog Synchronization was not triggered yet, then the online version of Product1 does not exist. Cross-reference to staged version is not created.

2. Products Catalog Synchronization:

- a. Product1 from the Products Catalog Staged Version is copied within the same Products Catalog to the Products Catalog Online Version.
- b. As Product1 contains a reference to Category1 from the Categories Catalog Staged Version, which is a part of the Category Catalog, it is checked if this synchronization job is dependent or depends on synchronization job from the Category Catalog.
- c. The dependency is configured, the Categories Catalog Online Version is checked for the online version of Category1. As the Categories Catalog Synchronization has already been triggered, the online version of Category1 exists. The reference between online version of the items from different catalogs is created.

After both synchronization jobs are finished, items from online versions of catalogs hold only the references to each other. The situation would look exactly the same if synchronization jobs ran in a different order.

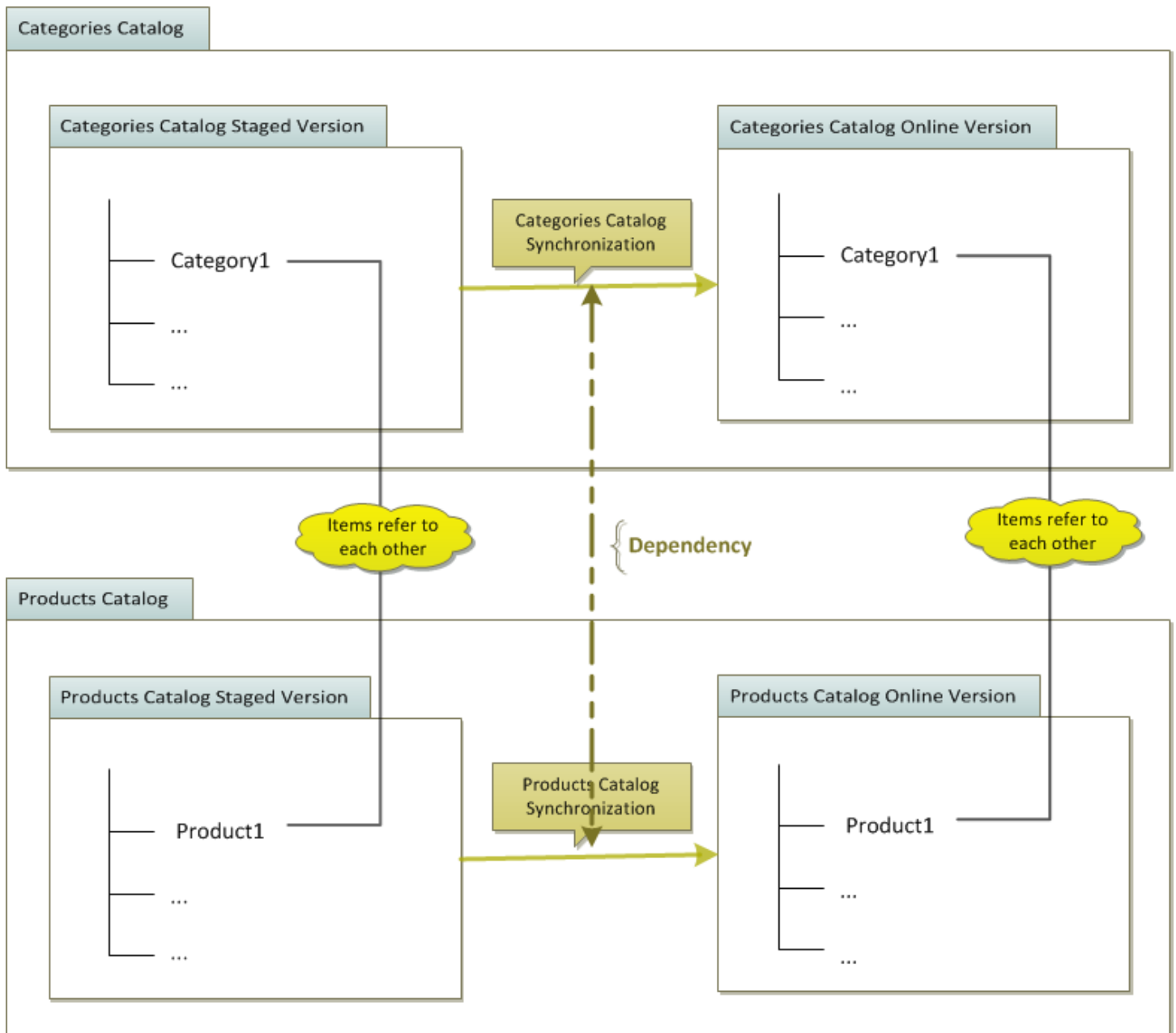


Figure: Catalogs after dependent synchronization.

Configure Dependency in Code

To set a dependency, you need to use one of these attributes: `dependsOnSyncJob` and `dependentSyncJob`. From the code perspective it does not matter which attribute is used because they use many-to-many relations, so dependency works in the same way.

Configure CatalogVersionSyncJobs with Dependency

1. Create synchronization jobs. Creates synchronization jobs for the given catalogs from the staged catalog version to the online catalog version.

```
final CatalogVersionSyncJob category_job = createSyncJob("categoryCatalog", "category_staged")
final CatalogVersionSyncJob product_job = createSyncJob("productCatalog", "product_staged", "I
```

2. Set a dependency. Make sure that a rerun of the synchronization always uses a new cron job.

i Note

Every execution of the **CatalogVersionSyncJob** requires a new cron job.

```
//Method that creates for a given synchronization job a new CronJob and configures it automat
private CatalogVersionSyncCronJob createFullSyncCronJob(final CatalogVersionSyncJob job)
{
    final CatalogVersionSyncCronJob cj = (CatalogVersionSyncCronJob) job.newExecution();
    job.configureFullVersionSync(cj);
    return cj;
}
```

```
//Now you get your CronJobs for the given jobs
CatalogVersionSyncCronJob category_cj = createFullSyncCronJob(category_job);
CatalogVersionSyncCronJob product_cj = createFullSyncCronJob(product_job);
```

```
//Set the dependencies between jobs
final Set<CatalogVersionSyncJob> dependantJobSet = new HashSet<CatalogVersionSyncJob>();
dependantJobSet.add(product_job);
```

```
category_job.setDependentSyncJobs(dependantJobSet);
//Alternative use: product_job.setDependsOnSyncJob(dependantJobSet);
```

3. Execute cron jobs asynchronously.

```
category_job.perform(category_cj, false);
product_job.perform(product_cj, false);
```

It is worth noting that after the execution of cron jobs, synchronization jobs of one or both cron jobs may still be running or has not even started yet. Therefore, you need to check if they have finished. See details in *Waiting for Synchronization Jobs* section.

Checking Results of Synchronization and Rerun

It is possible that one of dependent synchronization jobs ends with an error. This is normal, because one job relies on an item which is created by the other job. If this item is not there, this job reports an error. In the second run, this job will be successfully finished, because the item exists. Therefore, after both synchronizations are finished, check the result and do another rerun if needed.

```
EnumerationValue fail = category_cj.getFailureResult();
if (category_cj.getResult().equals(fail) || product_cj.getResult().equals(fail) ) //one of them fa
{
    //You need to create a new CronJob for the rerun
    CatalogVersionSyncCronJob category_cj = createFullSyncCronJob(category_job);
    CatalogVersionSyncCronJob product_cj = createFullSyncCronJob(product_job);

    //Execute them asynchronously
```

10/25/2021

```
category_job.perform(category_cj, false);
product_job.perform(product_cj, false);
}
```

If, after the second run, the synchronization job still finishes with an error, then it is a real error and it should be investigated.

Waiting for Synchronization Jobs

1. The waiting is divided into two parts:

- Waiting if both synchronization jobs started in a given timeout
- Waiting if both synchronization jobs finished in a given timeout

```
private void checkExecutionOfAsynchronousJobs(final long startTimeOutInMillis, final long
{
    assertAllStarted(startTimeOutInMillis, cronjobs);
    assertAllFinished(executionTimeOutInMillis, cronjobs);
}
```

2. Set waiting if both synchronization jobs started in a given timeout.

```
private void assertAllStarted(final long startTimeOutInMillis, final CatalogVersionSyncCronJob
    throws InterruptedException
{
    final long timeoutTime = System.currentTimeMillis() + startTimeOutInMillis;
    //check if the cronjob had started
    while (!allStartedOrFinished(cronjobs) && System.currentTimeMillis() < timeoutTime)
    {
        Thread.sleep(500);
    }
    if(!allStartedOrFinished(cronjobs)) throw new RuntimeException("At least one of the given J
}

private boolean allStartedOrFinished(final CatalogVersionSyncCronJob... cronjobs)
{
    for (final CatalogVersionSyncCronJob cronjob : cronjobs)
    {
        if (!(cronjob.isRunning() || cronjob.isFinished()))
        {
            return false;
        }
    }
    return true;
}
```

3. Set waiting if both synchronization jobs finished in a given timeout.

```
private void assertAllFinished(final long endTimeOutInMillis, final CatalogVersionSyncCronJob.
{
    final long timeoutTime = System.currentTimeMillis() + endTimeOutInMillis;
    //check if the cronjob had started
    while (!areAllOfThemFinished(cronjobs) && System.currentTimeMillis() < timeoutTime)
    {
        Thread.sleep(500);
    }
    if(!areAllOfThemFinished(cronjobs)) throw new RuntimeException("At least one of the Jobs di
}

private boolean areAllOfThemFinished(final CatalogVersionSyncCronJob... cronjobs)
{
    for (final CatalogVersionSyncCronJob cronjob : cronjobs)
    {
        if (!cronjob.isFinished())
        {
            return false;
        }
    }
}
```

```
    return true;
}
```

4. To wait for the above synchronization jobs, they need to be executed in the following manner:

```
//Execute them asynchronously
category_job.perform(category_cj, false);
product_job.perform(product_cj, false);
checkExecutionOfAsynchronousJobs(2000, 5000, category_job, product_job); //waiting at least 2
```

Running CatalogVersionSyncJobs Synchronously

If synchronization jobs are started asynchronously, this means that both jobs run in the same thread, one after one.

To run jobs synchronously do the following:

```
category_job.perform(category_cj, true); //code execution stops here for some seconds till job is f
product_job.perform(product_cj, true); //code execution stops here for some seconds till job is fin

//here, the jobs could still have an error result, so another single rerun is necessary
//But you do not need to poll the jobs if they are already finished or not.
```


Performing Multithreaded Catalog Synchronization in Backoffice

To support systems with large numbers of products in a multi-catalog scenario, SAP Commerce offers multithreaded catalog synchronization. You can perform it in Backoffice.

Context

Multithreaded catalog synchronization enables you to use as many cores as you like to speed up catalog synchronization operations. You can configure the number of cores for each synchronization job separately. Our tests on 16 core machines have demonstrated an almost linear scaling if the database can handle the load. Follow the steps to configure the multithreaded synchronization.

Procedure

1. To run a multithreaded catalog synchronization, log into Backoffice and navigate to **System Multithreaded Synchronization**.
2. Use the  button to add a new synchronization configuration.

Filter Tree entries

- Business Processes
- Background Processes
- Types
- Saved Queries
- Backoffice Saved Queries
- Report Definitions
- Personalization
- System Setup Audit
- Audit Report Config
- Multithreaded Synchronization**
 - Retention Rule
 - Search and Navigation
- Catalog
- Multimedia
- User
- Order
- Price Settings

SAVED QUERIES

No queries

SEARCH

Code	Synchronization source
✓ sync apparel-deContentCatalog:Staged->Online	Apparel DE Content Catalog : Staged
✓ sync apparel-ukContentCatalog:Staged->Online	Apparel UK Content Catalog : Staged
✓ sync apparelProductCatalog:Staged->Online	Apparel Product Catalog : Staged
✓ sync electronics-deContentCatalog:Staged->Online	Electronics Content Catalog DE : Staged
✓ sync electronics-ukContentCatalog:Staged->Online	Electronics Content Catalog UK : Staged
✓ sync electronics-usContentCatalog:Staged->Online	Electronics Content Catalog US : Staged

0 ITEMS SELECTED

Sync Default:Staged -> Default:Online

CONFIGURATION CRONJOBS RESTRICTIONS ADVANCED ATTRIBUTES ADMINISTRATION

ESSENTIAL

Code	Synchronization source	Synchronization target
Sync Default:Staged -> Default:Online	default catalog : Staged	default catalog : Online

hmc.essential

SYNCHRONIZATION ATTRIBUTES CONFIGURATION

3. Configure the multithreaded synchronization as regular synchronizations.
4. Once the configuration is ready, navigate to **Administration Unbound** to adjust the advanced settings.

Sync Default:Staged -> Default:Online

REFRESH SAVE

CONFIGURATION CRONJOBS RESTRICTIONS ADVANCED ATTRIBUTES **ADMINISTRATION**

UNBOUND

Cache size 5000	Dependent Sync-Jobs ...	Depends on Sync-Jobs ...	Use transactions <input type="radio"/> True <input checked="" type="radio"/> False
Number of scheduler threads 1	Number of used threads 1	Documents + Create new Output Document	Assigned Cockpit Item Templates ...
Comments ...	Create new elements <input checked="" type="radio"/> True <input type="radio"/> False	Effective synchronization languages German [de]	Exclusive mode <input type="radio"/> True <input checked="" type="radio"/> False

For synchronization, you can set the following:

- **Cache size** in entries: Keep the default.
- **Usage of transactions**: By default set to false.
- **Number of scheduler threads**: How many threads are used for the scheduling if the medias. Factory default is 1, meaning no multithreading.
- **The Number of used threads**: Enter the number of the used threads. Factory default is 1, meaning no multithreading.

5. Save and execute the synchronization as usual.

Synchronization of Custom Item Types

SAP Commerce allows you to synchronize catalog versions into another catalog version by defining synchronizations.

[Item Type Requirements](#)

This is custom documentation. For more information, please visit the [SAP Help Portal](#)

Item types which should be included in the synchronizing process need to meet some basic requirements before it is possible to add them to a synchronization item.

[Marking a Type as Synchronizing Capable](#)

If all requirements are met, the item type can be **marked** as synchronizable.

[Synchronization Strategies](#)

Once our item type has been marked as synchronizable, it will be automatically recognized as such by any synchronizing operation without a need to configure anything else. This means that source version items of this type are translated into their existing target version counterpart items.

[Synchronizing a Single Item](#)

Apart from synchronizing the entire catalog, you can also choose to synchronize single items.

[Troubleshooting](#)

Even though it's much easier to include own item types in the synchronization operation there are still some points to be remembered in case of any issues.

Item Type Requirements

Item types which should be included in the synchronizing process need to meet some basic requirements before it is possible to add them to a synchronization item.

The required settings can be done either manually in Backoffice or declared inside the `items.xml` file.

Catalog Version Home Attribute

First of all, item types must provide a catalog version home attribute typed to `CatalogVersion`. This way, each item of this type knows where it belongs to. This attribute should be initial, mandatory, and read-only if possible. For details on setting `CatalogVersion` as the home attribute, see [Setting CatalogVersion as the Home Attribute for a Type](#).

Presence of a Unique Key Attribute

An item must have one or more unique key attributes. A synchronizing capable item has to be distinguishable within its enclosing catalog version. For example, for the type **Unit** the unique key attribute is **code**.

Setting CatalogVersion as the Home Attribute for a Type

Follow these steps to learn how to set `CatalogVersion` as a home attribute for a type.

Context

In our example, we set `CatalogVersion` as a home attribute for the `Unit` type. If possible, the attribute must be set to initial, mandatory, or read-only.

Once you followed the steps, you will end up with the following configuration:

```
<attribute qualifier="catalogVersion" type="CatalogVersion">
  <modifiers read="true" write="false" search="true" initial="true" optional="false"/>
  <persistence type="property"/>
</attribute>
```

You may decide to make those changes directly in `items.xml` if you don't want to use Backoffice.

Procedure

1. In the Backoffice tree, go to **System Types** and select a type, for example Unit.
2. On the **Properties** tab, click **Create new Attribute**

Unit [Unit]

REFRESHSAVE

COMMONPROPERTIESXML REPRESENTATIONEXTENDEDADMINISTRATION

code	java.lang.String [java.lang.true	true	false	true
conversion	java.lang.Double [java.lang.true	true	true	true
name	[localized:java.lang.String]true	true	true	true
unitType	java.lang.String [java.lang.true	true	false	true

+ Create new Attribute

3. Choose **CatalogVersion** as **Feature** type and edit a **Qualifier** by entering for example **catalogVersion**.
4. Save and the attribute is listed as property of the type.

Marking a Type as Synchronizing Capable

If all requirements are met, the item type can be **marked** as synchronizable.

Context

You can do it manually in Backoffice or this can be declared inside the `items.xml` file.

Procedure

1. In Backoffice go to the **Extended** tab of the type editor.
2. In the **Catalog** group, set the **Is repository capable** to **True**.
3. In the **Repository attribute** drop-down list box, select the catalog version home attribute, for example **catalogVersion**.
4. To the **Repository unique key attributes** list, add the qualifier code of the attribute, for example code for **Unit**

Unit

SEARCH

Unit [Unit]

COMMONPROPERTIESXML REPRESENTATIONEXTENDEDADMINISTRATION

IdentifierExtensionname

Unitcore

EXTENDED

PERSISTENCE

CATALOG

is repository capable

True

False

Repository attribute

Unit [Unit] -> [CatalogVersion]

Repository unique key attributes

Unit [Unit] -> Identifier [code]

...

This is custom documentation. For more information, please visit the [SAP Help Portal](#)

24

Alternatively, this can also be done by editing the `items.xml` model file. The following example refers to the default catalog item **Keyword**. It shows the shortened **Keyword** item type declaration, which illustrates how to use the `<custom-properties>` tag to mark the type, define the catalog version, and unique key attributes:

```
<itemtype code="Keyword" ...>
  <custom-properties>
    <!-- marking the type as synchronizing capable here: -->
    <property name="catalogItemType">
      <value>java.lang.Boolean.TRUE</value>
    </property>

    <!-- define catalog version attribute here: -->
    <property name="catalogVersionAttributeQualifier">
      <value>"catalogVersion"</value>
    </property>

    <!-- define unique key attributes here; separate multiple attribute qualifiers by comma:
    <property name="uniqueKeyAttributeQualifier">
      <value>"keyword"</value>
    </property>
  </custom-properties>
  <attributes>
    <attribute qualifier="keyword" type="java.lang.String">
      <modifiers read="true" write="true" search="true" optional="false"/>
      <persistence type="property"/>
    </attribute>
    ...
    <attribute qualifier="catalogVersion" type="CatalogVersion">
      <modifiers read="true" write="true" search="true" optional="false"/>
      <persistence type="property"/>
    </attribute>
  </attributes>
  ...
</itemtype>
```

Synchronization Strategies

Once our item type has been marked as synchronizable, it will be automatically recognized as such by any synchronizing operation without a need to configure anything else. This means that source version items of this type are translated into their existing target version counterpart items.

Often this is not enough, because a item is not automatically created inside the target version, unless it's part of another item which is copied to the target version.

There are two ways to ensure that items are created inside the target version:

- Using **part-of mechanism** to create/copy them together with enclosing or referring items
- Making it the **root type** of the synchronization to create/copy them without any enclosing or referring item required

Synchronizing with Enclosing or Referring Items Using Part-of Logic

To use the part-of mechanism means that each occurrence of the item type inside an attribute of other synchronized item types (like Product) must be marked as part-of within the synchronization.

Context

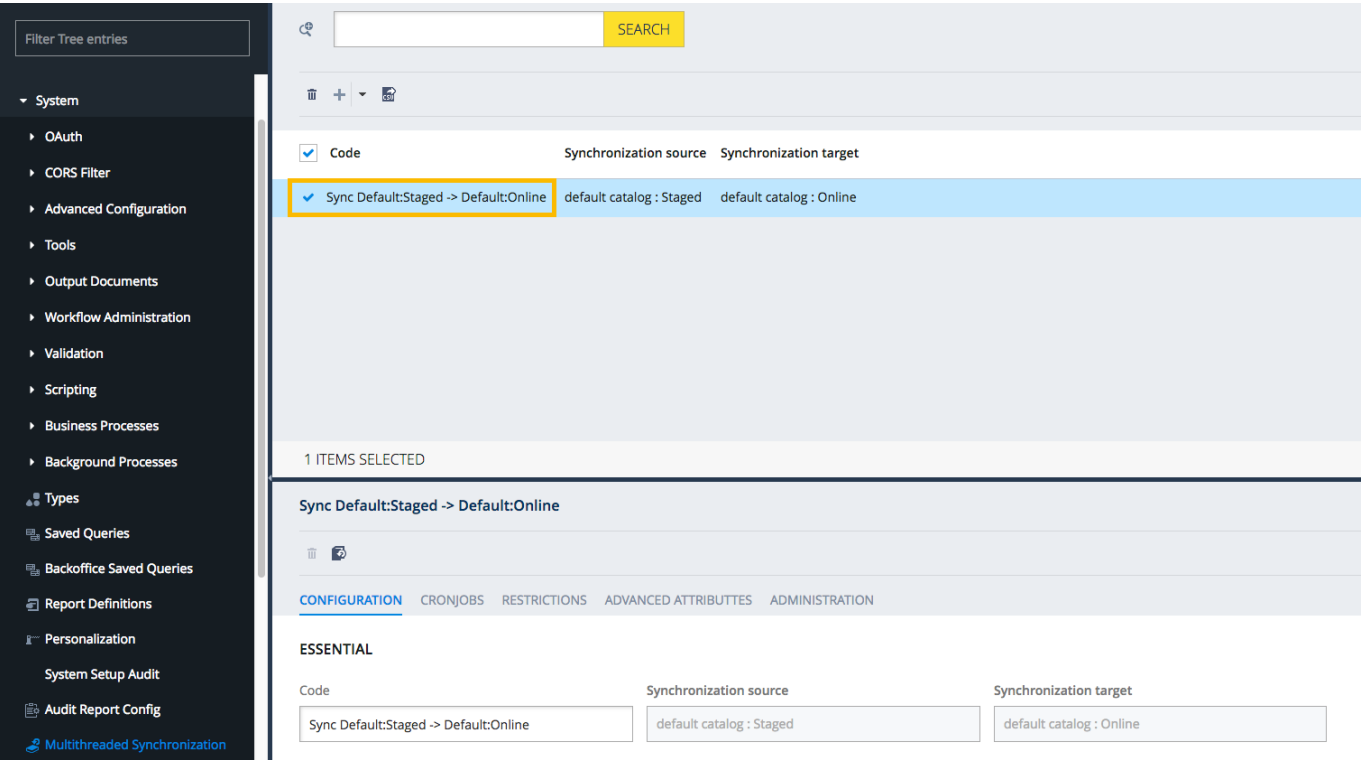
Now, items of our type are always treated like part-of items which means they are always created if they do not exist within the target catalog version. To achieve this, use Backoffice to configure synchronization attributes configuration for Multithreaded Synchronization as described in the following steps:

Procedure

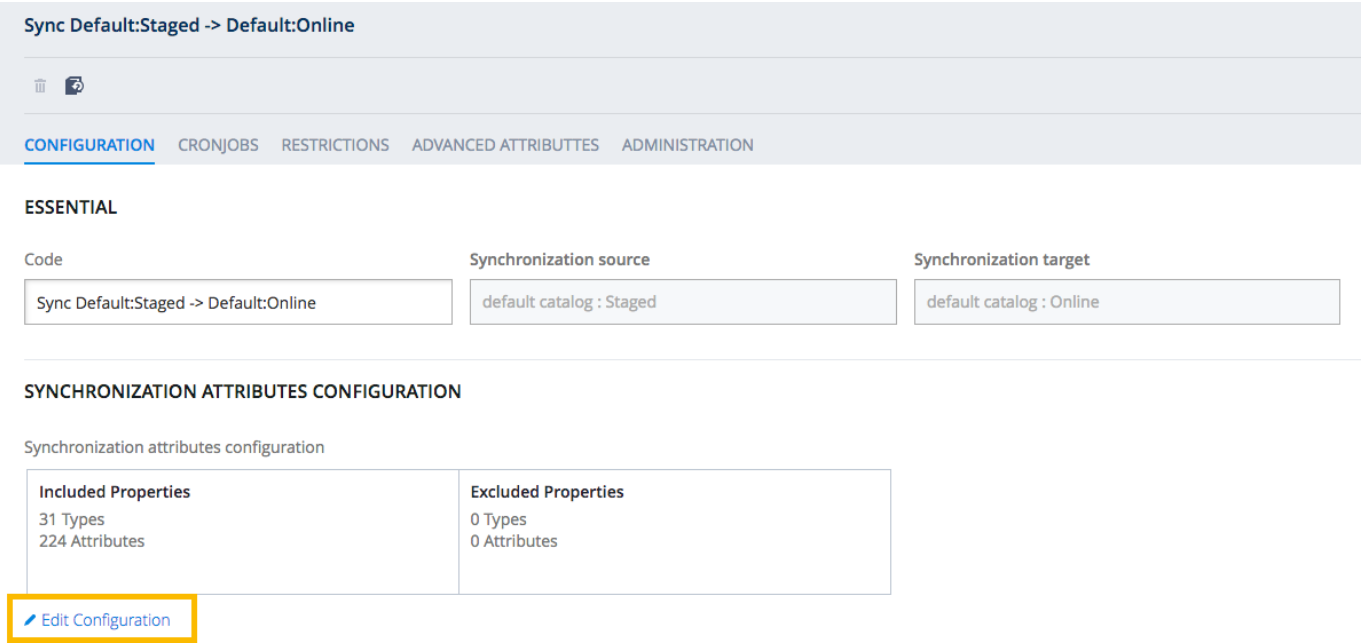
1. Log into Backoffice and navigate to **System Multithreaded Synchronization** .

The result view displays available synchronization jobs.

2. Choose the synchronization job you wish to modify to expand a synchronization editor.



3. On the **CONFIGURATION** tab in the editor area, click **Edit Configuration** button.



4. Expand the tree to find all attributes containing our item type. You can also use the search field to find a specific attribute.

Edit Synchronization Attributes Configuration

Search

Item [Item]

Assigned Cockpit Item Templates [assignedCockpitItemTemplates]

Comments [comments]

Documents [allDocuments]

Owner [owner]

Abstract Configuration [AbstractAsConfiguration]

Catalog Version [catalogVersion]

Unique Identifier [uid]

Boost Item Configuration [AbstractAsBoostItemConfiguration]

Boost Rule [AsBoostRule]

Facet Configuration [AbstractAsFacetConfiguration]

Facet Value Configuration [AbstractAsFacetValueConfiguration]

Synchronize

Copy by value

Untranslatable value

Partially translatable

Attribute

Completely Included

Partially Included

Not Included

CANCEL

SAVE

5. Enable **Copy by value**.

Because these settings cannot be declared inside items.xml, you should put some code inside the `createEssentialData` method of your extension manager, which performs these changes.

```

public void createEssentialData(Map params, JspContext jspc) throws Exception
{
    ...

    CatalogManager cm = CatalogManager.getInstance();

    // get source and target version of synchronization
    CatalogVersion src = cm.getCatalog( "hwcatalog" ).getCatalogVersion( "staged" );
    CatalogVersion tgt = cm.getCatalog( "hwcatalog" ).getCatalogVersion( "online" );

    // find rule by source and target (provided there is just one!)
    SyncItemJob syncJob = cm.getSyncJob( src, tgt );

    // get attribute which should be treated as part-of
    AttributeDescriptor ad = TypeManager.getInstance().getComposedType(
        Product.class
    ).getDeclaredAttributeDescriptor( "keywords" );

    // get or create attribute synchronization config item
    SyncAttributeDescriptorConfig cfg = syncJob.getConfigFor(
        ad,
        true /* create on demand */
    );

    // set part-of behavior to true
    cfg.setCopyByValue( true );

    ...
}

```

Synchronizing as Root Type

If our item type describes rather central or standalone items without much connection to other synchronized item types you should consider to add it to the root types of the synchronization instead of tweaking attributes as described before.

Context

All items of synchronization root types are evaluated at the beginning of a synchronizing operation and scheduled for synchronization if:

- No counterpart item exists within the target catalog version, unless this has been disabled.
- The source item modification time stamp is newer than last synchronizing time.
- An item exists within the target catalog version but no longer in the source catalog version (by default this is disabled).

You can set add the new item to synchronization root types in the Multithreaded Synchronization in Backoffice.

Procedure

1. Navigate to **System Multithreaded Synchronization**.
2. Select the synchronization from the list.
3. Go to **Administration Unbound**
4. Add the new type to the Root types list.

5. Of course this may also be put in code, preferably inside the `createEssentialData` method of your extension manager.

```
public void createEssentialData(Map params, JspContext jspc) throws Exception
{
    ...

    CatalogManager cm = CatalogManager.getInstance();

    // find source and target version
    CatalogVersion src = cm.getCatalog( "hwcatalog" ).getCatalogVersion( "staged" );
    CatalogVersion tgt = cm.getCatalog( "hwcatalog" ).getCatalogVersion( "online" );

    // find synchronization by source and target version
    SyncItemJob syncJob = cm.getSyncJob( src, tgt );

    // get modifieable list of root types
    List<ComposedType> rootTypes = new ArrayList<ComposedType>( syncJob.getRootTypes() );

    // insert our type at first position
    rootTypes.add(
        0,
```

```

        TypeManager.getInstance().getComposedType( "Keyword" )
    );

    // dont forget to save them again
    syncJob.setRootTypes( rootTypes );

    ...
}

```

Synchronizing a Single Item

Apart from synchronizing the entire catalog, you can also choose to synchronize single items.

It is always possible to trigger synchronizations for single items if:

- A synchronization exists for the selected item's catalog version.
- The current user has got the permission to change the target catalog version.

For details on synchronizing single items, see [Synchronizing Items](#).

For details on synchronizing catalogs, see [Synchronizing Catalogs](#).

Troubleshooting

Even though it's much easier to include own item types in the synchronization operation there are still some points to be remembered in case of any issues.

Bear in mind the following notes while working with synchronization of custom items:

- Dependencies: Real lifetime dependencies (for example between `Order` and `OrderEntry`) must be obeyed when configuring synchronization root types: the parent item type must be placed before its part-of item type.
- Unique key behavior differences: Sometimes unique key constraints are put down into code and cannot be changed (for example for `Language.isoCode`), no matter what we're trying to configure at its composed type; synchronization just checks configured constraints but item creation will most likely throw errors.
- Cyclic part-of configuration: When using the synchronization attribute configuration to mark some attributes as part-of this may collide with other (real or configured) part-of attributes; synchronization will fail whenever a part-of item tries to copy its parent (or any of its parents) as part-of too.
- Root type list contains super and subtypes: Make sure each new synchronization root type is not subtype of any existing root type; otherwise synchronization will at least perform slow or may fail completely.
- Modification time issues:
 - Keep in mind that changing a part-of item does not mark its owning item modified automatically. To allow synchronization to schedule the owning item correctly you have to mark it modified manually, preferably inside the part-of item's business code.
 - On the other hand relation attributes and plain property backed attributes do mark the enclosing item modified.
 - Since there is just one modification time for each item the synchronizing operation may only notice that something has changed, but not which attributes. So even if the synchronization omits several attributes a modified item will always be synchronized regardless, which attributes are actually changed.
- Root types affect full synchronization only: When triggering synchronization for single items (via search result and so on) the synchronizing operation will publish these items only. Of course all synchronization attribute settings are obeyed here as well.

Catalog Synchronization API

You can start catalog synchronization processes and get their status via API.

i Note

The actual setup of a `CatalogSyncJob` is not in scope here.

Starting Full Catalog Synchronization

`CatalogSynchronizationService` offers multiple methods for starting the synchronization process.

Ad-Hoc Full Synchronization

There are methods for creating and starting sync jobs ad hoc. You can use them when there are no pre-existing `CatalogVersionSyncJob` items set up for a given pair of catalog versions:

```
CatalogSynchronizationService css = ...
CatalogVersionModel source = ...
CatalogVersionModel target = ...

// create a job and a cron job and execute synchronously using the default amount of worker threads
css.synchronizeFully( source, target )

// create a job and a cron job and execute synchronously using 16 worker threads
css.synchronizeFully( source, target, 16 )

// create a job and a cron job and execute in background using the default amount of worker threads
css.synchronizeFullyInBackground(source, target)
```

! Restriction

This is recommended for testing only as sync jobs are heavy weight and shouldn't be created more than once per catalog version pair.

Full Synchronization for an Existing Sync Job

The regular use case of starting synchronization is based on an existing `CatalogVersionSyncJob`, containing all relevant settings (for example number of workers, root types, attributes).

You can pass additional settings via a `SyncConfig` parameter:

```
CatalogSynchronizationService css = ...
CatalogVersionSyncJobModel syncJob = ...

// configuring
SyncConfig cfg = new SyncConfig();
cfg.setSynchronous( false ); // background
cfg.setForceUpdate( true ); // update all even if timestamps are the same

css.synchronize( syncJob, cfg );
```

Starting Partial Synchronization

Simplified Partial Synchronization API

The service offers methods for use cases where only a subset of source catalog items is supposed to be processed:

```
CatalogSynchronizationService css = ...
CatalogVersionSyncJobModel syncJob = ...
SyncConfig cfg = ...

// start partial sync for given items and specific job
List<ItemModel> itemsToSync = ...
css.performSynchronization( itemsToSync, syncJob, cfg );

// start multiple partial sync processes for given items from multiple catalog
List<ItemModel> itemsFromMultipleCatalogs = ...
List<CatalogVersionSyncJobModel> multipleSyncJobs = ...
css.performSynchronization( itemsFromMultipleCatalogs, multipleSyncJobs, cfg );
```

i Note

Filtering of Items

The given items are always **filtered** to match the specified sync jobs. Items that don't match their source or targets catalog versions are silently ignored! This also means that when multiple sync jobs are specified, items may be accepted by one job but ignored by another.

i Note

Scheduling Removal

In case of items **removed** from their source catalog versions, schedule their **target catalog version counterparts** in order for the sync process to correctly remove them. Since they're recognized as target items automatically, you can simply add them to the item list.

Partial Synchronization via SyncConfig API

Partial synchronization can also be started using SyncConfig:

```
CatalogSynchronizationService css = ...
CatalogVersionSyncJobModel syncJob = ...

List<ItemModel> sourceItemsToSync = ...
List<ItemModel> targetItemsToRemove = ...

SyncConfig cfg = new SyncConfig();
cfg.setSynchronous( false ); // background

// add source items for insert or update
for( ItemModel item : sourceItemsToSync )
{
    cfg.addItemToSync( item.getPk() );
}
// add (orphan) target items for removal
for( ItemModel item : targetItemsToRemove )
{
    cfg.addItemToDelete( item.getPk() );
}
```

```

}

// start it
css.synchronize( syncJob, cfg );

```

Getting Process Information

Most methods actually return `SyncResult` objects, which allows you to inspect the status of a started sync process:

```

CatalogSynchronizationService css = ...
CatalogVersionSyncJobModel syncJob = ...
SyncConfig cfg = ...

// start it
SyncResult sr = css.synchronize( syncJob, cfg );

// get sync cron job
CatalogVersionSyncCronJob syncCronJob = sr.getCronJob();

// if started synchronous: check result
if( sr.isFinished() && sr.isSuccessful() )
{
    // great
}

// if started in background: poll for status - note that models need to be refreshed to reflect cha
ModelService modelService = ...

CatalogVersionSyncCronJob syncCronJob = sr.getCronJob();
while( !sr.isFinished() )
{
    Thread.sleep( 1000 );
    modelService.refresh( cronJob() );
}

```

Catalog Item Synchronization Status API

You can look up synchronization statuses for specific items contained within catalog versions.

A `CatalogVersionSyncJob` specifies a pair of `CatalogVersion` items as **source** and **target** catalog versions. You can only obtain synchronization statuses for items contained in either of them.

Such a sync job also defines a list of `ComposedTypes` (the so called **root types**) that determine which type of catalog-version-contained items are actually subject to being synchronized by this job. Therefore, only items that are of one of the root types of a given sync job can have synchronization statuses.

Synchronization Statuses

The statuses that items within the source and target versions (using `SyncItemStatus` codes) can have, include:

Source catalog version	Target catalog version	Status	Comment
------------------------	------------------------	--------	---------

Source catalog version	Target catalog version	Status	Comment
Item X exists.	No copy of item X exists.	COUNTERPART_MISSING	The next sync run will create a copy of X within the target catalog version.
Item X exists.	A copy of item X exists with the same content as X.	IN_SYNC	Items are in sync - nothing to do.
Item X exists.	A copy of item X exists with content older than X.	NOT_SYNC	The source item has been changed since last sync - next sync will update the copy of X.
Item X was removed.	A copy of item X still exists	COUNTERPART_MISSING	The next sync run will apply the source item removal and will remove the copy of X as well.

Looking up Statuses for Items

To look up an individual status information for a given item and a sync job, use `SynchronizationStatusService`:

```
// given
SynchronizationStatusService sss = ...
ItemModel myItem = ...
CatalogVersionSyncJobModel syncJob = ...

// get info
SyncItemInfo info = sss.getSyncInfo( myItem, syncJob);

switch( info.setSyncStatus() )
{
    case COUNTERPART_MISSING:
        if( info.getFromSource() ){
            // new item to be copied
        }else{
            // leftover target item without source
        }
        break;
    case NOT_SYNC:
        // item exists on both sides but need update
}
}
```

For any pair of items being synchronized, an `ItemSyncTimestamp` item is created for tracking their status. You can look up that item using `SyncItemStatus`:

```
// given
SyncItemInfo info = sss.getSyncInfo( myItem, syncJob);
ModelService modelService = ...

// get last sync pending attributes
PK tsPK = info.getSyncTimestampPk();
if( tsPK != null )
{
    ItemSyncTimestampModel ts = modelService.get(tsPK);
    for( AttributeDescriptorModel pending : ts.getPendingAttributes() )
    {

```

10/25/2021

```
{  
    // got it  
}
```

To look up multiple statuses for lists of items or for multiple sync jobs, use bulk methods:

```
// given  
SynchronizationStatusService sss = ...  
  
// bulk get for one item with multiple sync jobs set up  
ItemModel item = ...  
List<ItemSyncJobModel> syncJobs = ...  
List<SyncItemStatus> states = sss.getSyncInfo( item, syncJobs );  
  
// bulk get for multiple items with multiple sync jobs set up  
ItemSyncJobModel syncJob = ...  
List<ItemModel> items = ...  
List<SyncItemStatus> states = sss.getSyncInfo( items, syncJob );
```

i Note

Note that the bulk methods check whether the given items are actually applicable to the given sync job or not. If they're not, information with `SyncItemStatus.NOT_APPLICABLE` is returned.

Looking up Sync Jobs for Items

Sometimes you may want to find out which sync jobs are actually responsible for a given item, either using it as source or target. For that purpose, the service offers these methods:

```
// given  
SynchronizationStatusService sss = ...  
ItemModel myItem = ...  
  
// where do we sync this to?  
List<SyncItemJobModel> syncFromHereJobs = sss.getOutboundSynchronizations();  
  
// where do we get data synced from?  
List<SyncItemJobModel> syncToHereJobs = sss.getInboundSynchronizations();
```

Parallel Catalog Synchronization

Full catalog synchronizations cannot run in parallel. It is possible, however, to start multiple partial sync CronJobs on condition the items you want to synchronize don't overlap.

You can pick one or more products from a catalog and synchronize them using a chosen catalog version sync job. The `CatalogVersionSyncCronJob` gets the `fullSync` attribute. This attribute is set by service during its creation, and contains information whether a given cron job performs a partial or a full synchronization.

There are some points to consider when starting an A catalog synchronization job:

- if another B. `fullSync == true` sync job is running --> failure ("Cannot perform partial sync cron job when there is other full sync cron job running")

- if `A.fullSync == true`, and any other sync is running --> failure ("Cannot perform full sync cron job when others are running")
- if `A.fullSync == false`, and no other sync is running --> success
- if `A.fullSync == false`, and there is another sync B with `B.fullSync == false` is running -> compare schedules:
 - overlap --> failure ("Cannot perform partial sync when there are items overlapping from other running cron job")
 - The number of items exceed the limit (`catalog.sync.partial.max.items` default value set to 500) --> failure ("Cannot perform partial sync when number of items to sync exceed the limit")
 - no overlap --> success

If you set the `abortOnCollidingSync` attribute to `true`, it sets a cron job status to `ABORTED` when the given cron job fails to pass the mentioned validation, otherwise the status stays as `UNKNOWN` for backward compatibility.

When checking for overlapping items with other running cron jobs, the items are taken into account as of the beginning of the cron job. So even if a cron job did synchronize most of its items, the overlap will be checked for all of them.

Synchronization Properties

These are some of the properties that you can use to configure your synchronization, or that are related to the synchronization process.

Property Name	Default Value	Definition
<code>synchronization.dumpfile.tempdir</code>	<code><\${HYBRIS_TEMP_DIR}>/sync</code>	<p>Defines a directory to store dump files for synchronization.</p> <p>If the property value is empty, the sync dump files are stored in your workspace directory.</p>