

apparel-ukIndex

Index

Hot Update Index

PROPERTIES

INDEXED TYPES

KEYWORD REDIRECTS

SYNONYMS

STOPWORDS

CRON JOBS

ADMINISTRATION

SYNONYMS

EXPORT TO SOLR

Synonyms configuration

hats -> caps

shoes,sneaker,sneakers,trainer -> shoe

shades,glasses -> sunglasses

Videokamera -> Camcorder

af -> Autofokus

deals -> special offers

small,mini -> compact

video camera -> camcorder

+ Create new Synonyms configuration

8. In the new window fill in the fields.

Create New Synonyms configuration

ESSENTIALS

Provide all mandatory fields

From:

Hats

To:

Caps

Language:

English [en]

CANCEL

DONE

Field Name	Description
From	The word you use when searching for a particular item, such as "hat".

Field Name	Description
To	The synonym for the searched term, such as "cap".
Language	<p>The language of the synonym.</p> <p>i Note</p> <p>The wizard allows you to select one language at the time so in order to create synonyms in more languages start the wizard once again.</p>

9. Click **Done** to finish.

The synonym is visible on the list.

10. Remember click **Save**, otherwise you will lose the changes introduced.

11. Export the synonyms to solr server using the **Export to Solr** button.

12. If the export was performed successfully, a confirmation will appear at the top of the page.

13. Re-index the items using the FULL indexing operation type.


14. If you want to remove the synonym from the list, hover over it and use the **X** button to remove it.


The synonym is no longer present on the list. Click **Save** to save your changes.


[PROPERTIES](#)
[INDEXED TYPES](#)
[KEYWORD REDIRECTS](#)
[SYNONYMS](#)
[STOPWORDS](#)
[CRON JOBS](#)
[ADMINISTRATION](#)

SYNONYMS

EXPORT TO SOLR

Synonyms configuration 


hats -> caps	
shoes,sneaker,sneakers,trainer -> shoe	
shades,glasses -> sunglasses	
Hats -> Caps	
	...



15. If you want to delete the synonym permanently, click the entry.

a. An editor appears. Use the **bin** icon in the left top corner to delete the synonym.

Edit item hats -> caps



REFRESH SAVE

SYNONYM ADMINISTRATION

SYNONYM

From

hats

To

caps

Language

English [en]

b. You are asked to confirm your decision. Click [Yes](#) to confirm.

16. Remember to export the list again to the Solr server, to make it aware of the change.

Configure Stopwords

The Solr search server comes with a default stopwords list. However, you can use the Backoffice Administration Cockpit to create additional list of common words like **a**, **the**, and similar, that should not be considered by indexing mechanisms.

Procedure

1. Navigate to **System > Search and Navigation > Solr Facet Search Configuration > Facet Search Configurations** in the Explorer Tree.

A list of available configurations (if any) appears on the right.

2. Select a configuration and click it.

An editor appears.

3. Select the **Stopwords** tab.

apparel-ukIndex

Index Hot Update Index

PROPERTIES INDEXED TYPES KEYWORD REDIRECTS SYNONYMS **STOPWORDS** CRON JOBS ADMINISTRATION

STOPWORDS

EXPORT TO SOLR

Stopwords

a
the

...

4. At this point you can select the already created stopwords or add new ones.

5. In order to select an already existing stopword, use the magnifying glass icon to open the Reference Search window. You can also start typing the name of the stopword to find and add it.

6. To add a new stopword click the **Create new Stopword** button.

Index Hot Update Index

PROPERTIES INDEXED TYPES KEYWORD REDIRECTS SYNONYMS **STOPWORDS**

STOPWORDS

EXPORT TO SOLR

Stopwords

a
the
the
a
+ Create new Stopword

7. In the new window fill in the fields.

Create New Stopword ✕

ESSENTIALS
Provide all mandatory fields

Stopword:

Language:

CANCEL DONE

Field Name	Description
Stopword	A stopword of your choice.
Language	The language of the stopword. i Note The wizard allows you to select one language at the time so in order to create stopwords in more languages start the wizard once again.

8. Click **Done** to finish.

The stopword is visible on the list.

9. Remember click **Save**, otherwise you will lose the changes introduced.

10. Export the stopwords to the solr server using the **Export to Solr** button.

11. If the export was performed successfully, a confirmation will appear at the top of the page.

12. If you want to delete the stopword permanently, click the entry.

a. An editor appears. Use the **bin** icon in the left top corner to delete the stopword.

Edit item die



REFRESH SAVE

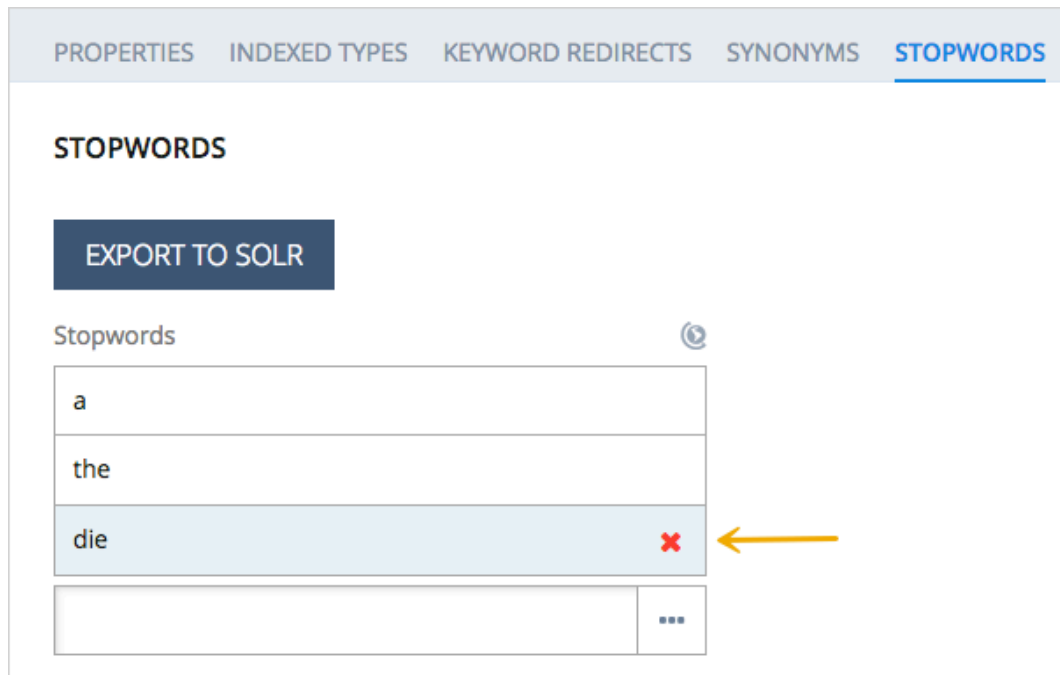
STOPWORD ADMINISTRATION

STOPWORD ⌵

Stopword Language

b. You will be asked to confirm your decision. Click **Yes** to confirm.

13. If you want to remove the stopword from the list, hover over it and use the **X** button to remove it.



The stopword is no longer present on the list. Click **Save** to save your changes.

14. Remember to export the list again to the Solr server, to make it aware of the change.

Update Solr Index

Instructions on how to update the Solr index using the Backoffice Administration Cockpit.

There are several ways to update the Solr index and you can use the visual interface of the Backoffice Administration Cockpit to perform all the necessary actions. You can create, update or remove Solr index.

[Indexer Operation Wizard](#)

You can create, update, and remove the Solr index. Steps presented below will show you how to update the Solr index using the Indexer Operation Wizard.

[Indexer Hot Update Wizard](#)

The main purpose of the Indexer Hot Update Wizard is to allow you performing quick, ad-hoc updates of the Solr index. It is useful if a need emerges of modifying attributes of only one or few indexed item types. You can also remove particular indexed items. The section guides you through the process.

[Update Solr Index with CronJob](#)

You can set up the cron job responsible for updating the Solr indexes. Follow the steps listed below to complete this task.

Indexer Operation Wizard

You can create, update, and remove the Solr index. Steps presented below will show you how to update the Solr index using the Indexer Operation Wizard.

Procedure

1. Navigate to **System > Search and Navigation > Solr Facet Search Configuration > Facet Search Configurations** in the Explorer Tree.

A list of available configurations (if any) appears on the right.

2. Select a configuration and click it.

An editor appears.

3. Click the **Index** button.

apparel-ukIndex	Apparel UK Solr Index
apparelUKConfig	
electronicsIndex	Electronics Solr Index

apparel-ukIndex

Index Hot Update Index

PROPERTIES INDEXED TYPES KEYWORD REDIRECTS SYNONYMS STOPWORDS CRON JOBS ADMINISTRATION

PROPERTIES

Name ⓘ	Description ⓘ	Index name prefix ⓘ
apparel-ukIndex	Apparel UK Solr Index	apparel-uk

4. In the wizard window select the operation value.

Create New SolrIndexerOperationWizard

SOLR INDEXER
Provide indexer configuration

Operation values

full

START

CANCEL

You have the following options to choose from:

- **FULL:** Re-creates the index, it means it drops the existing index and re-creates a new one by given properties. During the full update, index may not be available in full scale until the new index is fully created.
- **UPDATE:** Updates certain existing indexed items. Only the differences are updated and only updated items are affected. Other indexed items are available for browsing.
- **DELETE:** Rarely used. Deletes certain indexed items from the index, while other items are not affected and index remains available. Example of use could be if a certain item becomes unavailable and you are sure it will not be available in future, you can set up the **Delete** operation to remove this item from the Solr index.

5. Click **Start** to begin the indexing process.

6. Once the indexing is completed, a confirmation window is displayed. Click **Done** to close the wizard.

Indexer Hot Update Wizard

The main purpose of the Indexer Hot Update Wizard is to allow you performing quick, ad-hoc updates of the Solr index. It is useful if a need emerges of modifying attributes of only one or few indexed item types. You can also remove particular indexed items. The section guides you through the process.

Procedure

1. Navigate to **System > Search and Navigation > Solr Facet Search Configuration > Facet Search Configurations** in the Explorer Tree.
- A list of available configurations (if any) appears on the right.
2. Select a configuration and click it.
- An editor appears.
3. Click the **Hot Update Index** button to open the wizard.

apparel-ukIndex Apparel UK Solr Index

apparelUKConfig

electronicsIndex Electronics Solr Index

apparel-ukIndex

Index

Hot Update Index

PROPERTIES INDEXED TYPES KEYWORD REDIRECTS SYNONYMS STOPWORDS CRON JOBS ADMINISTRATION

PROPERTIES

Name ⓘ

apparel-ukIndex

Description ⓘ

Apparel UK Solr Index

Index name prefix ⓘ

apparel-uk

4. In the Type and Operation section provide the following values.

Create New SolrIndexerHotUpdateWizard

TYPE AND OPERATION

Set indexed type and indexer operation

ITEMS

Select items to index

Item type

Product

Operation values

update

BACK

CANCEL

Field Name	Description
Item type	The item type to be indexed. You can choose from the types listed under Indexed Types tab. More about defining Indexed Types: Create and Configure Indexed Types .
Operation values	Indexer operation value. You have two values to choose from: update or delete . Depending on the chosen operation, the items will be added or deleted from the index.

5. In the Items section, select the items to be indexed. You can choose from the items available from the list or create a new item.

6. If you want to create a new item, click the [Create New \(item name\)](#) button.

i Note

The button name and wizard fields differ depending on the item type.

Update Solr Index with CronJob

You can set up the cron job responsible for updating the Solr indexes. Follow the steps listed below to complete this task.

Context

Updating the entire document for large and complex catalogues in order to include just a few changes may be time consuming and is not efficient. A partial update is a way to update only a subset of the attributes of a Solr document, simultaneously preserving the value of the others.

A new task for a partial update can be created through the Backoffice Administration Cockpit. Like with any other cron job, you can either specify one or multiple time slots to run this task in, or run the task immediately.

Procedure

1. Navigate to **System > Background Processes > CronJobs** in the Explorer Tree.

A list of defined cronjobs appear on the right.

2. Click the  button at the top of the page.

A list of available cronjobs is displayed.

3. Select [Cronjob for solr Indexer \(external\)](#).
4. Use the wizard pop up window to configure the settings for your cronjob.

Create New Cronjob for solr indexer



PROVIDE ALL MANDATORY FIELDS

Language:

English [en]

Code:

Logs Operator: 

AND

Process on server node group: 

Log to file:

☒ True ☐ False

Allow system recovery after changes:

☒ True ☐ False

Configure the settings. The most important for you are the following:

- **Code:** a unique code for your cronjob, that makes it easy to distinguish. Example: `partialUpdate-electronicsIndex-cronJob`
- **Indexed Type:** a type to be indexed. Example: `Product`.
- **Indexer Operation:** the manner of indexing. To perform the partial update you need to select `partial_update`.
- **Query:** a query used for indexing. Example: `select {PK} from Product`.
- **Solr configuration:** facet search configuration the cronjob applies to.
- **Job definition:** a definition containing settings for a particular job. Select `solrExtIndexerJob`.

5. Click **Done** to finish.

Your cronjob is now present on the list.

6. Click the cronjob entry to see its details.

As you can see, the cronjob settings are grouped in tabs.

Each tab features the **Essential** section, where you can find the general settings for your cronjob. Below this section you can find information specific for a particular tab:

- **Log:** Information about the logs and job steps. More details about the logs:
- **Task:** Indexer specific information and notification settings.
- **Run as:** Options that allow you to run the task as a defined user or with defined session attributes, specify a dedicated server node to run the task (to optimize the load balancing) and choose the priority of the task.
- **Time Schedule:** Settings allowing you to specify one or multiple time slots to run this task in, or you can run the task immediately.
- **System Recovery:** Settings allowing you to specify whether the system should recover after the changes.
- **Administration:** Advanced configuration options.

Running a CronJob

Context

You have two options to run a cronjob: you can either start the cronjob right away or schedule it using a particular time slot.

Procedure

1. To schedule a cronjob go to the [Time Schedule](#) tab.

solrIndexerJob : full-apparel-deIndex-cronJob - FINISHED - SUCCESS

LOG TASK RUN AS **TIME SCHEDULE** SYSTEM RECOVERY ADMINISTRATION

ESSENTIAL

Code	Current status	Job definition	Last result
full-apparel-deIndex-cronJob	FINISHED	solrIndexerJob	SUCCESS

Timetable	Last start time	Enabled	Last end time
Daily at 03:05:00	Jan 27, 2017 9:04:36 AM	<input checked="" type="radio"/> True <input type="radio"/> False	Jan 27, 2017 9:05:23 AM

SCHEDULE

You can specify one or multiple time slots to run this task in, or you can run the task immediately.

Trigger

Daily at 03:05:00 - Sat Jan 28 03:0...

2. To schedule a cronjob go to the [Time Schedule](#) tab.

You can see the [Schedule](#) section where you can specify one or multiple slots to run this task.

solrIndexerJob : full-apparel-deIndex-cronJob - FINISHED - SUCCESS

LOG TASK RUN AS **TIME SCHEDULE** SYSTEM RECOVERY ADMINISTRATION

Timetable	Last start time	Enabled
Daily at 03:05:00	Jan 27, 2017 9:04:36 AM	<input checked="" type="radio"/> True <input type="radio"/> False

SCHEDULE

You can specify one or multiple time slots to run this task in, or you can run the task immediately.

Trigger

Daily at 03:05:00 - Sat Jan 28 03:0...

+ Create new Trigger

3. Click the [Create new Trigger](#) button.
4. A wizard window pops up.
5. Fill in the fields to provide mandatory information about the schedule. The name of the cronjob is already filled in for you.

Create New Trigger

×

ALL REQUIRED FIELDS

Cronjob:

solrIndexerJob : full-apparel-deIndex-cronJob - FINISHED - SUCCESS

Active: ?

☒ True
☐ False

Next Activation time: ?

Jan 27, 2017 2:45:33 PM

📅

Cron expression:

CANCEL

DONE

Field Name	Description
CronJob	Name of the cronjob. The field is filled in automatically.
Active	A flag specifying if the trigger is activated.
Next Activation Time	The time the trigger will be activated.
Cron Expression	Cron expression governing the time scheduling.

6. Click **Done** to finish scheduling.

You have successfully configured basic settings for a cronjob trigger. To add more specific settings, select the trigger from the list and double-click it to open the editor.

Create and Configure Indexed Types

Learn how to create and configure Indexed Type by defining its properties and queries.

Each item in Solr has a certain type. There is a specific number of types coming with the `solrfacetsearch` extension. However, you can also add new Solr item type. By adding the new types, you can define your own set of items that can be searched by the **SolrFacetSearch** service.

The Backoffice Administration Cockpit provides you with a wizard that helps you to easily create and configure each item.

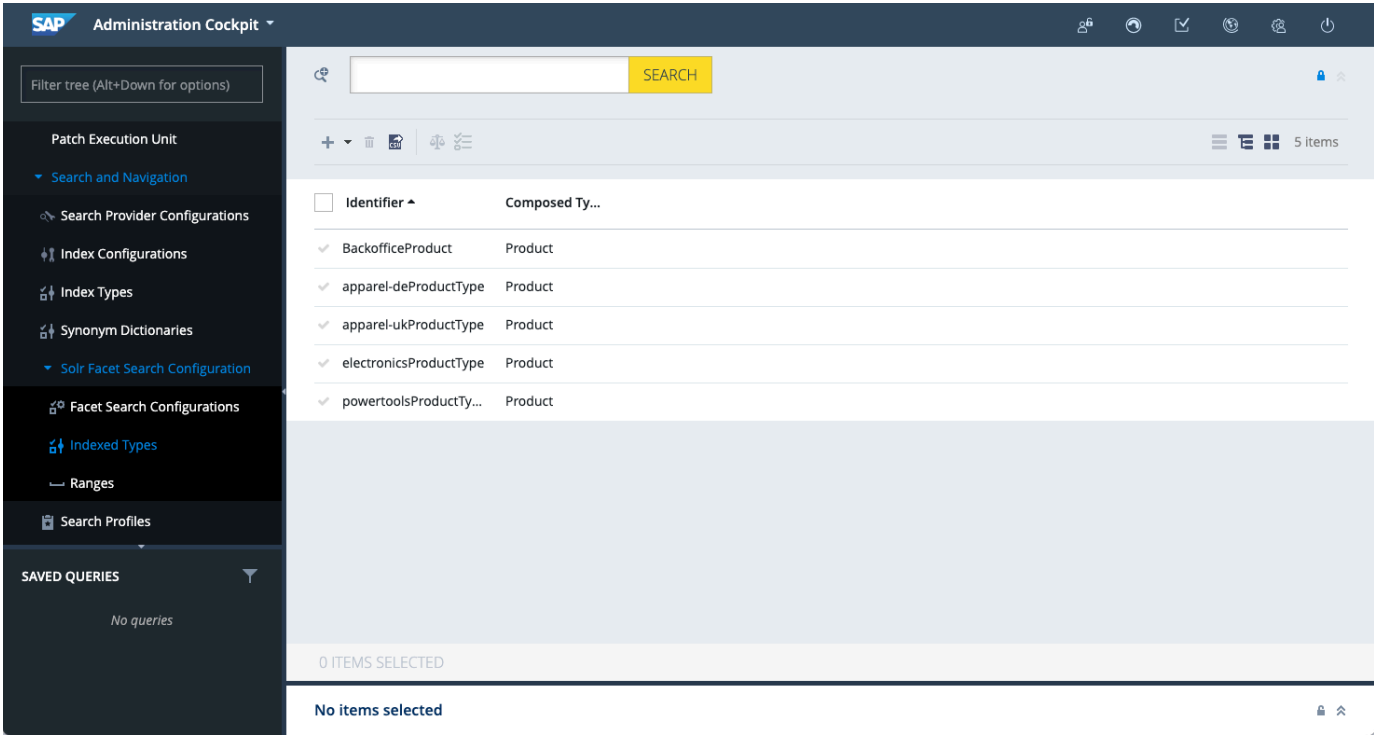
→ Tip

You don't have to fill all information and fields while creating a new item type. You can finish at any time by clicking **Done** and return to the main editor to add the missing information by editing your items.

Add Solr Indexed Types

Procedure

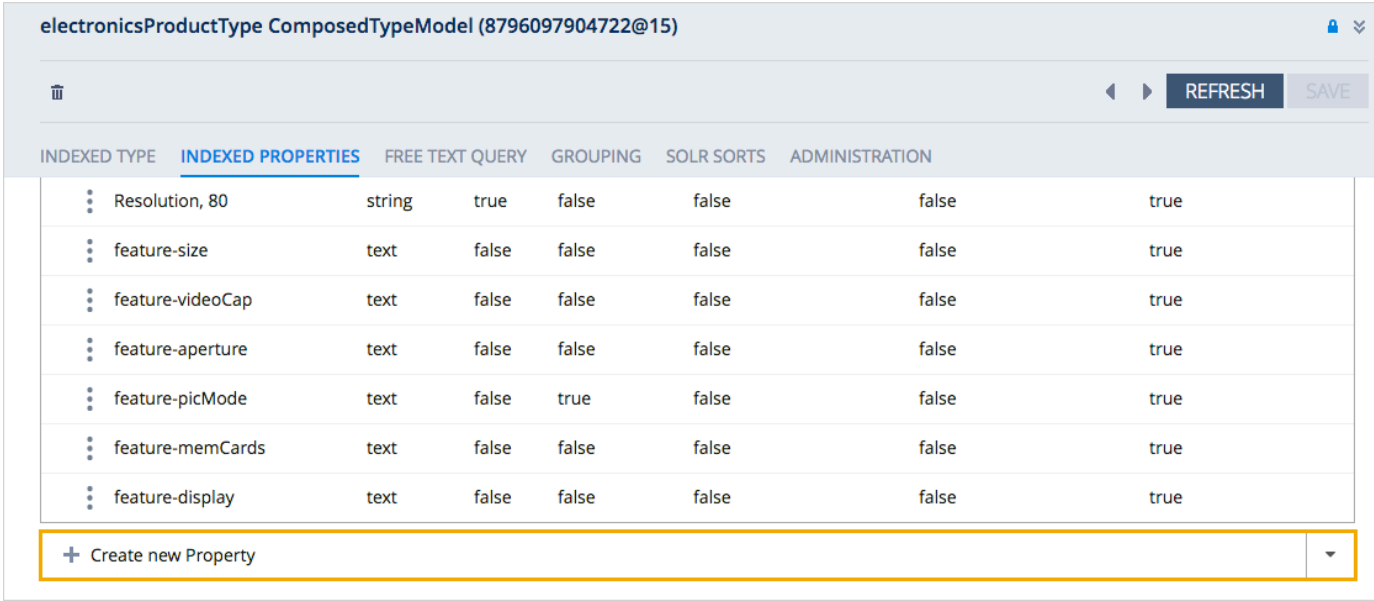
1. Navigate to **System > Search and Navigation > Solr Facet Search Configuration > Indexed Types**. On the right, you can see the list of available item types (if there are any defined).



2. Click the **+** button at the top of the editor.

A pop-up window appears.

3. In the **Type** section, you need to choose the types to be indexed. Provide a unique **Identifier** and select a **Composed Type** for your item.



i Note

You can finish creating a new type at any time by clicking **Done**. You can return to the editor later on to make changes.

To define the properties at this point click **Next**.

Defining Properties for Indexed Types

You can codify your indexed types using properties.

Context

You have already created an indexed type and you want to define it further by adding additional settings.

Procedure

1. Navigate to **System > Search and Navigation > Solr Facet Search Configuration > Indexed Types**. Select the configuration that you have created the indexed type for.
2. Select the **Indexed Properties** tab and click **Create new Property**.

electronicsProductType ComposedTypeModel (8796097904722@15)

REFRESH SAVE

INDEXED TYPE INDEXED PROPERTIES FREE TEXT QUERY GROUPING SOLR SORTS ADMINISTRATION

Resolution, 80	string	true	false	false	false	true
feature-size	text	false	false	false	false	true
feature-videoCap	text	false	false	false	false	true
feature-aperture	text	false	false	false	false	true
feature-picMode	text	false	true	false	false	true
feature-memCards	text	false	false	false	false	true
feature-display	text	false	false	false	false	true

+ Create new Property

3. In the **Essentials** section, define the following settings:

Field Name	Description
Name	A unique name of the property.
Type	Determines the custom type of the property.
Facet	Determines if the property should be classified as a facet.
Multi-value	Determines if the property can hold multiple values.
Currency	Determines if the property can hold currency.
Localized	Determines if the property can be localized.
Use for spell-checking	Determines if the property should be subject to spell-checking.
Use for auto-complete	Determines if the property should be subject to auto-complete.
Include in Response	Determines if the field is returned within a query response.
Use for highlighting	Determines if the field is used to highlight search terms from search results.

4. Click **Next** to proceed.
5. In the **Other** section, specify any additional values for your property.
6. Click **Done**.

Results

A new property is added to the list.

Define Queries for an Indexed Type

Context

The Indexer Configuration section of the Indexed Types tab provides you with a wizard to create the indexer queries.

Procedure

1. Navigate to **System > Search and Navigation > Solr Facet Search Configuration > Indexed Types**. Select the configuration you want to create a search query for and click it to access the editor.

The screenshot shows the SAP Solr Facet Search Configuration - Indexed Types editor. The 'electronicsProductType' configuration is selected. The 'SEARCH CONFIGURATION' tab is active, showing 'Search Query Templates' with a '+ Create new Search Query Template' button. The 'INDEXER CONFIGURATION' section is highlighted with a yellow border, showing a table of indexer queries.

Type	Query value
full	<pre>SELECT {PK} FROM {Product} WHERE ({varianttype} IS NULL OR {varianttype} NOT IN ({{ SELECT {PK} FROM {varianttype} WHERE {code} = 'Electronic AND {code} NOT IN({{ SELECT {code} FROM {GenericVariantProduct} }}} SELECT DISTINCT tbl.pk, tbl.code FROM ({{ SELECT DISTINCT {p:PK} AS pk, {p:code} AS code, {p:varianttype} AS varianttype FROM {Product} AS p LEFT JOIN CustomerReview AS cr ON {cr:product}={p:PK} } WHERE {p:modifiedtime} >= ?lastIndexTime OR {cr:modifiedtime} >= ?lastIndexTime }} UNION</pre>

2. Click the **Create new Indexer Query** button at the bottom of the section to create a new indexer query.
A new window pops up.
3. In the **Essentials** section, provide the values for the mandatory fields.

Field Name	Description
Identifier	A unique name of the query.
Type	The type of your query. You have the following values to choose from: <ul style="list-style-type: none"> o FULL: recreates the index o UPDATE: updates some documents in the index o PARTIAL_UPDATE: allows you to select the fields for the update o DELETE: deletes documents from the index
Query value	A flexible search query.
Inject current date	Checks if <code>currentDate</code> parameter should be injected to the query.
Inject current time	Checks if <code>currentTime</code> parameter should be injected to the query.
Inject last index time	Checks if <code>lastIndexTime</code> parameter should be injected to the query.

4. Proceed to **Others** section. Select the users, if queries are executed with the user's privileges. Click **Done** to finish.

5. You can now define the additional settings for the Indexed Type, using the main editor.

Additional Settings for Indexed Types

Search Query Templates

The search query templates introduce numerous customization options for getting improved search results for the customers. When creating the templates you can use various attributes, such as page size, grouping or sorting with respect to various channels, or even storefront parts. For details on creating and managing search query templates see [Search Query Templates](#).

Others

For each of your indexed types, you can also define the additional settings - you'll find under in the **Indexed Types** tab in the section **Others**. You can define the following items:

- **Identity provider:** Identity provider for the item type
- **Model loader**
- **Model fields values provider:** Sets fields values provider that depends on this type rather than on particular properties.
- **Default property value provider**
- **Result converter:** Result converter (spring bean id) needed to convert Solr result data into a simple data object that could be used to display indexed type (i.e. product) data directly on the front end

OTHERS

Identity provider ?

Model loader ?

Model fields values provider ?

Default property value provider ?

productIdentityProvider

defaultModelLoader

demoIndexedTypeFieldsValuesPi

productPricesValueResolver

Result converter ?

Advanced Configuration

In the **Advanced Configuration** section you can define additional parameters for an indexed type, such a shard configuration. The number of shards and replication factor can be configured per `facetsearchconfig`, in which case all the Indexed Types related to it will use the same configuration. For details on configuring the `facetsearchconfig` item see: [Define Search and Indexer Configuration](#).

However, it is possible to override the configuration for an individual Indexed Type using the **Advanced Configuration** section.

ADVANCED CONFIGURATION

Additional parameters of Indexed type ?

solr.collection.numShards ▶ 2

solr.collection.replicationFactor ▶ 2

+ Add new item

i Note

If there are no parameters set, neither during the configuration of a `facetsearchconfig` item for an individual indexed type, the default values will be used:

- `solr.collection.numShards = 1`
- `solr.collection.replicationFactor = 1`

Manage Solr Ranges

Instructions on how to manage Solr range sets using the Backoffice Administration Cockpit.

Ranged facets for the Solr server can be created for any date or numeric field that supports range queries. With ranges you can put together a series of range queries for prices, dates, etc. Creating range-based facets enables you to target the search results more precisely.

The Solr search server supports numeric and date ranges that help you to get the results based on certain conditions. Some properties can be indexed according to the range they belong to. For example, for facet search approach it is convenient to have the price ranges as facets.

Data Types

You can create value range sets for a given data type. They are parent items that group the value ranges of the same type. The following types are allowed: string, int, double, float, and date. Having defined the range sets, you normally want to fill them with the actual value ranges. They are simple items, consisting of name, from, and to value.

i Note

While creating a range, keep in mind that the upper and lower limits (from and to) are validated according to the range set type. Hence, for numerical types only the numerical values are accepted. Similarly, date values are not accepted unless they follow `yyyy-MM-dd\ [HH:mm]` format.

[Create Ranges](#)

[Modify Ranges](#)

[Remove Ranges](#)

Create Ranges

Procedure

1. Navigate to **System** > **Search and Navigation** > **Solr Facet Search Configuration** > **Ranges**.

A list of already created sets of ranges (if any) appears on the right.

2. Click the  button to start the wizard.

3. In the **Essentials** section, provide the required information.

Create New Set of ranges

ESSENTIALS

Provide all mandatory fields

Name: ?

electronicsPriceRangeEUR

Type: ?

double

CANCEL

DONE

Field Name	Description
Name	A unique name for your ranges set.
Type	The type of the set, for example: text, double, float, string.

4. Click **Done** to finish.
- The defined range set will appear on the list.
5. Click the entry to enter the editor mode.
6. The editor appears.
7. In the **Essential** section, select the facet search configuration your set applies to.
- The configuration is added.

Name

Type

ElectronicsPriceRangeEUR

double

MegaPixelRange

double

ElectronicsPriceRangeEUR:double

RANGES

ADMINISTRATION

ESSENTIAL

Name

ElectronicsPriceRangeEUR

Qualifier

EUR

Facet search configuration

electronicsIndex

...

RANGE VALUES

Type

double

Ranges

Name

From

To

EUR0-EU...

0

50

+ Create new Value range

8. In the **Range values** section click the **Create new Value range** option.

Provide the required values.

Create New Value range

RANGES

Provide range values

Name

EUR0-EUR49.99

From

0

To

49.99

CANCEL

DONE

Field Name	Description
Name	A unique name for your range value
From	The minimum value for the range.

Field Name	Description
To	The maximum value for the range.

9. Click **Done** to finish.

A new range is added to the list.

ESSENTIAL

Name ⓘ

Qualifier ⓘ

Facet search configuration ⓘ

ElectronicsPriceRangeEUR

EUR

electronicsIndex

...

RANGE VALUES

Type ⓘ

Ranges ⓘ

double ▼

Name	From	To
⋮ EURO-EUR...	0	50
⋮ EURO-EUR...	0	49.99
+ Create new Value range		▼

10. You can now add some more ranges to the set.

Modify Ranges

Procedure

1. Navigate to **System > Search and Navigation > Solr Facet Search Configuration > Ranges**.

A list of already created sets of ranges (if any) appears on the right.

2. Select the set you want to modify and click it.

An editor appears.

ElectronicsPriceRangeEUR:double

RANGES ADMINISTRATION

ESSENTIAL

Name ⓘ

ElectronicsPriceRangeEUR

Qualifier ⓘ

EUR

Facet search configuration ⓘ

electronicsIndex

...

RANGE VALUES

Type ⓘ

double

Ranges ⓘ

	Name	From	To
⋮	EUR0-EUR...	0	50
⋮	EUR0-EUR...	0	49.99
+ Create new Value range			

Remove

Edit Details

You can now edit the settings for your set, add, modify and remove the ranges within a set.

3. Click **Save** to save your changes.

Remove Ranges

Procedure

1. Navigate to **System > Search and Navigation > Solr Facet Search Configuration > Ranges**.

A list of already created sets of ranges (if any) appears on the right.

2. Select the set you want to modify and click it.

An editor appears.

3. Use the bin icon to remove the set of ranges.

RANGES ADMINISTRATION

ESSENTIAL

Name ⓘ

ElectronicsPriceRangeEUR

Qualifier ⓘ

EUR

Facet search configuration ⓘ

electronicsIndex

...

RANGE VALUES

Type ⓘ

double

Ranges ⓘ

	Name	From	To
⋮	EUR0-EUR...	0	50
⋮	EUR0-EUR...	0	49.99
+ Create new Value range			

4. Click [Save](#) to save your changes.

Enable Statistics Collection

The `solrfacetsearch` extension supports collecting statistical data for queries sent to the Solr server..

i Note

To enable this functionality, you must be running in legacy mode and Solr must be installed locally. If you meet these prerequisites, you need to set the related properties in the `local.properties` file. You can set up the `solrfacetsearch` extension to collect important information in order to export it to the Solr server for further analysis. Use the configuration file to set the necessary options and collect information such as time stamp or results number.

Basic Configuration

Procedure

1. Enable the statistics collection.

```
solrStats.enableCollectingStatistics=true
```

2. By default, Solr collects statistical data in files located in the following folder:

```
solrStats.filesLocation=${HYBRIS_LOG_DIR}/solrstats
```

3. The files holding statistical data are have a prefix as in the example provided below

```
solrStats.filePrefix=stats.log
```

4. Files are rolled every single hour, this is configured by the following property

```
solrStats.dateFormat=yyyy-MM-dd.HH
```

Collected Data

If you enable the collection of statistical data, then the following information is stored to the Solr server once each query is executed:

- Date
- Index configuration
- Language
- Query
- Number of results

Use a Cronjob to Collect Statistics

Context

You should create cron job to store aggregated values into the database. The following information is than stored: date, index, configuration, language, query, average number of results for query, and count.

Procedure

1. Navigate to [System](#) [Background Processes](#) [CronJobs](#) 

A list of defined cronjobs appears on the right.

2. Click the  button.

A list of available cronjob types appears.

3. Select the **SolrQueryStatisticsCollectorCronJob** cronjob.

4. Use the wizard window provide the required values.

The most important values for you to provide are:

- **Code:** a code (name) for your cronjob that makes it easier to distinguish. It can be for example: Solr Statistics Collection for
- **Job definition:** a type of job to be executed. For this particular one, select **SolrQueryStatisticsCollectorJob**.

The remaining fields can be filled in at any time in the editor mode.

Create New SolrQueryStatisticsCollectorCronJob

×

PROVIDE ALL MANDATORY FIELDS

Retry execution, if the job can not be executed yet:

☒ True
 ☐ False

Once-only executable:

☐ True
 ☒ False

Process on server node group: ?

Code:

Process on server node: ?

Owner:

...

Allow system recovery after changes:

☒ True
 ☐ False

5. Click **Done** to finish.

Your new cronjob is now visible on the list. To run it, follow the steps described in: [Running a CronJob](#).

Query for Statistics

To get the statistics you can use the following query:

```
SELECT {PK} FROM {SolrQueryAggregatedStats}
```

SolrQueryAggregatedStats contains the following attributes:

- time
- indexConfig

- language
- query
- count
- avgNumberOfResults

Search and Navigation Module Architecture

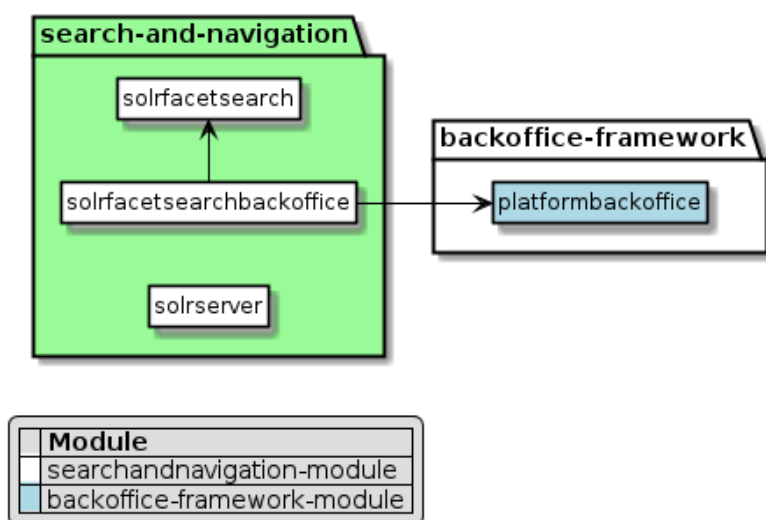
The Search and Navigation module is a set of extensions providing faceted search functionality based on the Solr server. Faceted search allows you to refine or navigate a collection of information by using a number of discrete attributes.

Dependencies

The following UML displays the dependencies between all extensions providing the functionality:

Click on an extension to view detailed information about it.

This image is interactive. Hover over each area for a description. Click highlighted areas for more information.



Please note that image maps are not interactive in PDF output.

Recipes

For a complete list of SAP Commerce recipes that may include this module, see [Installer Recipes](#).

For a complete list of the SAP Commerce Cloud, integration extension pack recipes that may include this module, see [Installer Recipe Reference](#).

Extensions

The Search and Navigation module consists of the following extensions:

[backofficesolrsearch Extension](#)

The backofficesolrsearch extension is an extension that adds the Apache Solr Search to backoffice. With Apache Solr Search, backoffice enables faster, more efficient and more optimized search functionality with the faceted navigation and near-real-time indexing.

[solrfacetsearch Extension](#)

The solrfacetsearch extension provides faceted search and navigation functionality based on the Apache Solr server. It enables you to do a faceted (also called dimensional) search over SAP Commerce items such as products and WCMS contents.

[solrserver Extension](#)

The solrserver extension includes a standalone Solr server which may be automatically configured, started and stopped together with the platform.

[solrfacetsearchbackoffice Extension](#)

The `solrfacetsearchbackoffice` extension provides the interface to create and manage search configurations using the Backoffice Administration Cockpit.

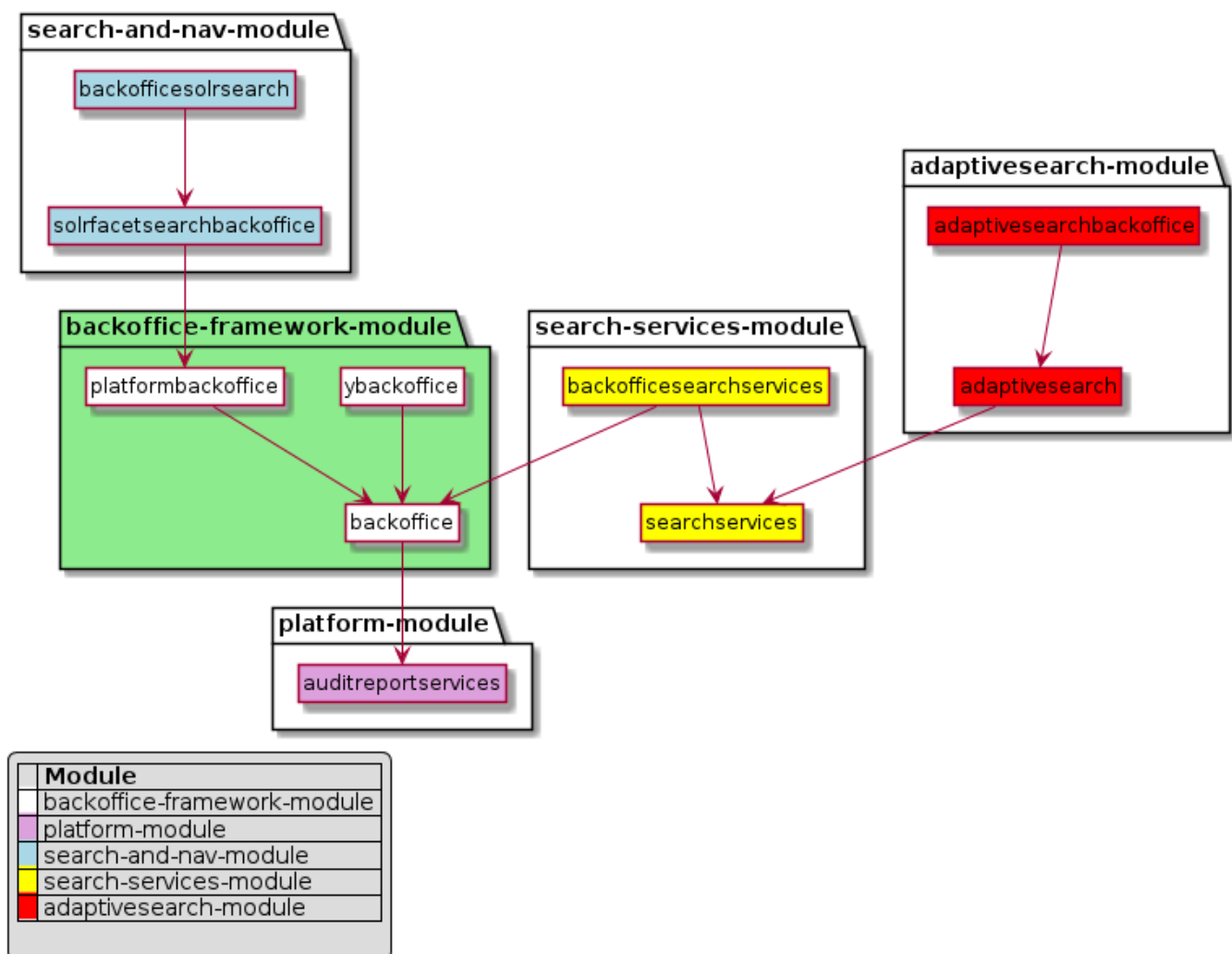
backofficesolrsearch Extension

The `backofficesolrsearch` extension is an extension that adds the Apache Solr Search to `backoffice`. With Apache Solr Search, `backoffice` enables faster, more efficient and more optimized search functionality with the faceted navigation and near-real-time indexing.

About the Extension

Name	Directory	Related Module
backofficesolrsearch	hybris/bin/modules/search-and-navigation/backofficesolrsearch	Search and Navigation Module

Dependencies



Dependencies

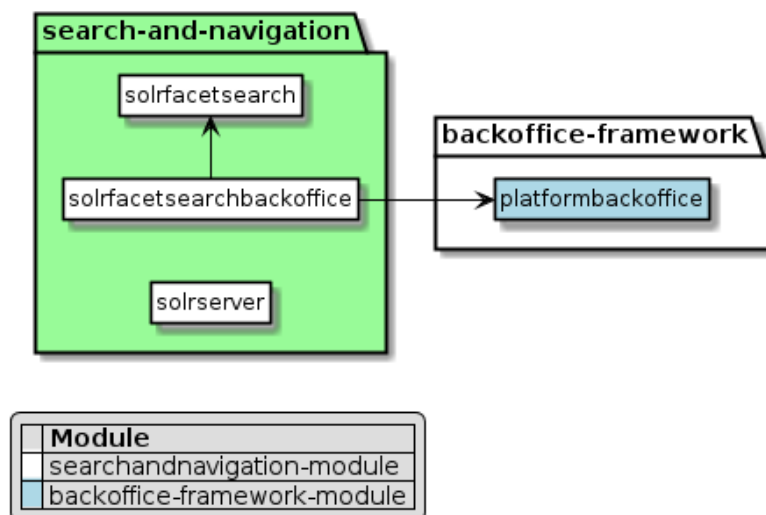
solrfacetsearch Extension

The `solrfacetsearch` extension provides faceted search and navigation functionality based on the Apache Solr server. It enables you to do a faceted (also called dimensional) search over SAP Commerce items such as products and WCMS contents.

⚠ Caution

Index only product-related data. Storing and indexing any sensitive data in Solr may pose a security risk.

Dependencies



solrserver Extension

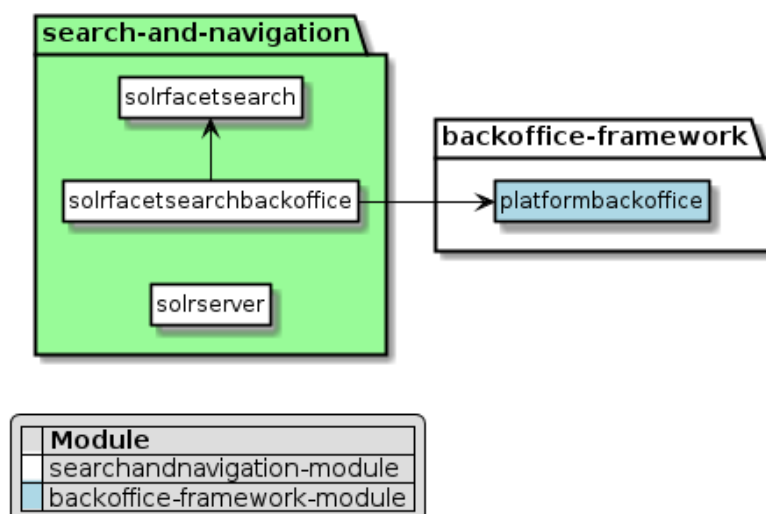
The **solrserver** extension includes a standalone Solr server which may be automatically configured, started and stopped together with the platform.

This extension is not required by default, but it is very convenient for you to enable it in CI or **developer** environments. Multiple Solr instances can be created. A Solr instance is a combination of configuration, data, and log files. Only one Solr server can be started/stopped per instance.

i Note

Do not use the **solrserver** extension on production systems. SAP recommends using a standalone Solr server on a separate machine. You can copy the **solrserver** from the extension and run it as standalone server.

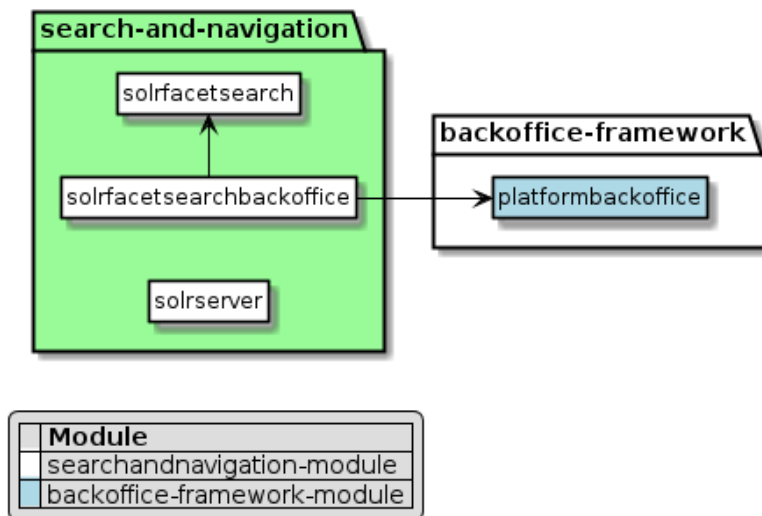
Dependencies



solfacetsearchbackoffice Extension

The `solrfacetsearchbackoffice` extension provides the interface to create and manage search configurations using the Backoffice Administration Cockpit.

Dependencies



Search and Navigation Module Implementation

Install and configure the Solr server to be able to use the indexing and search functionality. You can also define configurations for multiple Solr versions and learn how to make your Solr server secure.

[Solr Server Installation](#)

In order for the extension to work properly, it needs to be connected to a running Solr server. The installation and configuration depends on the mode you want to run the Solr server.

[Solr Facet Search Configuration](#)

The facet search configuration can be done through the graphical interface or the Backoffice Administration Cockpit.

[Solr Security](#)

The Solr security features include support for encrypting communication to and from Solr (as well as between the Solr nodes) using SSL and support for authentication and authorization provided by the Solr security frameworks. The security features are enabled by default when you use `solrserver` extension. In other case, you can use the provided sample configuration that you can easily enable.

[Multiple Solr Version Support](#)

SAP Commerce supports more than one version of Solr server.

[Common Functionality](#)

A part of functionality used in Search and Navigation module is common for search and indexing. For example, both of them use listeners API.

[Indexing Functionality](#)

There are a number of different topics relative to the indexing functionality. Each topic is documented in a separate guide document.

[Search Functionality](#)

To fully make use of the search functionality, get to know the strategies used in the search process, and learn how to customize your search by using Search API, sort providers and custom mapping.

Solr Server Installation

In order for the extension to work properly, it needs to be connected to a running Solr server. The installation and configuration depends on the mode you want to run the Solr server.

i Note

Make sure you have enabled the `solrfacetsearch` extension for the SAP Commerce by editing the `localextensions.xml` file in the `/hybris/config/` directory.

Compatibility with Apache Solr Server

Depending on the current release version, there can be several different versions of the Apache Solr server involved. The compatibility matrix along with the information on supporting the multiple versions of Solr are available in [Multiple Solr Version Support](#).

Use the links below to find out more about the functionality provided by the search and navigation module.

Security Considerations

When installing the Apache Solr, you should consider some security considerations. For details, on security features, see [Solr Security](#).

Standalone Solr Server Quick Installation

Quickly install solr server in standalone mode.

Context

Follow the steps for a quick installation and configuration of the solr server on your local machine.

Procedure

1. Enable the `solrserver` extension in your `localextensions.xml` file.

The extension is now ready to use. For further information on configuring extensions, see [Installation Based on Specified Extensions](#).

2. Create a Solr server instance. You can use the default instance included in the `solrserver` extension.

The default configuration is as follows:

```
solrserver.instances.default.autostart=true
solrserver.instances.default.mode=standalone
solrserver.instances.default.hostname=localhost
solrserver.instances.default.port=8983
solrserver.instances.default.memory=512m
```

If you need, you can override some of the properties in your `local.properties` file. For further information see: [solrserver Extension](#).

3. Start the SAP Commerce server.

Because autostart is enabled for the default Solr server instance, the Solr server will be started and stopped together with the platform. For further information see: [Installing SAP Commerce Manually](#) and [Installing SAP Commerce Using Installer Recipes](#).

4. Configure SAP Commerce to use the Solr server.

The most important attributes you should consider when updating the Solr server configuration are:

- **Mode:** set it to **standalone**.
- **Endpoint URLs:** should contain a single master URL with the the following value: `http://localhost:8983/solr`.

For details on how to easily configure the Solr server settings with the Backoffice Administration Cockpit see: [Create Facet Search Configuration](#).

Cloud Solr Server Quick Installation

Install and configure Solr server in cloud mode.

Context

Follow the steps for a quick installation and configuration of the solr server on your local machine.

Procedure

1. Enable the `solrserver` extension in your `localextensions.xml` file.

The extension is now ready to use. For further information on configuring extensions, see [Installation Based on Specified Extensions](#).

2. Create a Solr server instance. You can use the cloud instance included in the `solrserver` extension. However, you have to disable the `autostart` in the `project.properties` file for the default instance and enable it for the cloud instance as shown in the codeblock below.

```
solrserver.instances.default.autostart=false
solrserver.instances.cloud.autostart=true
```

If you want to, you can override the following default configuration included in the `local.properties` file to make it suitable to your needs:

```
# disables the autostart for the default Solr instance
solrserver.instances.default.autostart=false

solrserver.instances.cloud.autostart=true
solrserver.instances.cloud.mode=cloud
solrserver.instances.cloud.hostname=localhost
solrserver.instances.cloud.port=8983
solrserver.instances.cloud.memory=512m
solrserver.instances.cloud.zk.host=
solrserver.instances.cloud.zk.upconfig=true
```

For further information see: [solrserver Extension](#).

3. Start the SAP Commerce server.

Because `autostart` is enabled for the cloud Solr server instance, the Solr server will be started and stopped together with the platform. For further information see: [Installing SAP Commerce Manually](#) and [Installing SAP Commerce Using Installer Recipes](#).

4. Configure SAP Commerce to use the Solr server.

The most important attributes you should consider when updating the Solr server configuration are:

- **Mode:** set it to `cloud`.
- **Endpoint URLs:** should contain a single URL with value `localhost:9983`, this is the URL of the ZooKeeper server.

For details on how to easily configure the Solr server settings with the Backoffice Administration Cockpit see: [Create Facet Search Configuration](#).

Solr Facet Search Configuration

The facet search configuration can be done through the graphical interface or the Backoffice Administration Cockpit.

In order to smoothly index information, the Solr server needs a valid configuration. The overall configuration does not only determine the configuration of the Solr server, but also create a set of directives defining the specific platform items and the extent to which they should be indexed. The Solr facet search configuration is persistent in terms of items in the database.

The technical aspects of this configuration are covered in further sections of this document. Although this section contains the technical details, you can perform some configuration actions by yourself, using the Backoffice Administration Cockpit user interface.

Solr Server Configuration Files

As described in the installation guide, there are different ways to configure the Solr server. The default configuration should work in most cases, however you might want to change it a bit to adjust it to your needs.

Config Sets

It is possible to have a different configuration per indexed type. This is achieved by using named `configsets`, which are shared configuration directories stored under a configurable `configsets` base directory.

→ Tip

For information on customizing Solr by creating configuration files, see [Creating and Adding Custom Solr Files](#).

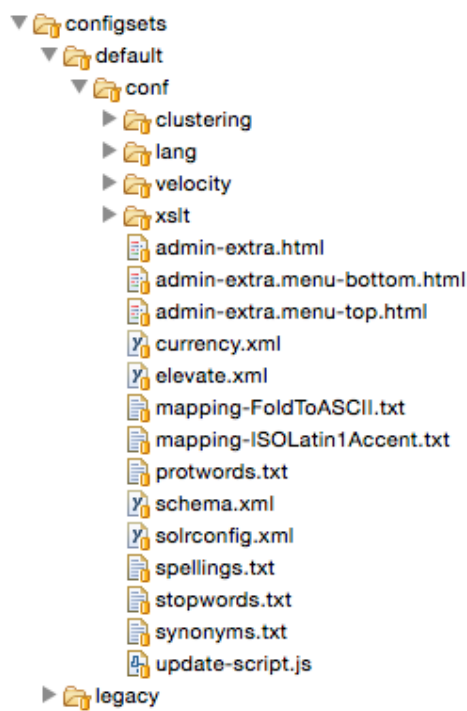
To create a `configsets`, simply add a new directory under the `configsets` base directory. The configset is identified by the name of this directory. It is recommended to use an existing config set as template. The structure is presented on the right.

The most important files are:

- `schema.xml`: defines the structure of your index, including fields and field types;
- `solrconfig.xml`: defines most of the parameters for configuring Solr itself.

i Note

It is not possible to change the `configsets` of an already existing index in the Solr server. In such a case, change the configset attribute in the indexed type, remove the corresponding index from the Solr server (configuration and data) and re-index.



Synonyms and Stopwords

Synonyms and Stopwords can be defined using the Backoffice Administration Cockpit. Once they are created, they have to be send to the Solr server. It is done automatically during the indexing process or can be done manually in the Backoffice Administration Cockpit using the **Export Stopwords/Synonyms** action, however using this action is **not recommended**.

Query Time vs. Index Time

Solr distinguishes between query time synonyms/stopwords and index time synonyms/stopwords, and analyzers may be defined for both. Runtime updates don't trigger reindexation of data. If synonyms/stopwords are updated and the index time is different from the query time, it may result in an inconsistency error. In such a case, reindexation may be required.

For further information, see: [Indexing Process](#).

Suggestions

Solr can provide suggestions for searched words. You can use one of the following components to activate this feature:

- `SpellCheckComponent`
- `SuggestComponent`

By default, the Solr server uses SpellCheckerComponent. You can change this configuration in the `solrconfig.xml` file. To switch to the SuggestComponent component, in the `config.xml` file, comment out the following code blocks:

```
<requestHandler name="/suggest" class="solr.SearchHandler">
  <lst name="defaults">
    <str name="spellcheck">true</str>
    <str name="spellcheck.dictionary">default</str>
    <str name="spellcheck.onlyMorePopular">true</str>
    <str name="spellcheck.count">5</str>
    <str name="spellcheck.collate">true</str>
  </lst>
  <arr name="components">
    <str>suggest</str>
  </arr>
</requestHandler>

<searchComponent name="spellcheck" class="solr.SpellCheckComponent">
  <str name="queryAnalyzerFieldType">text_spell</str>
  <lst name="spellchecker">
    <str name="name">default</str>
    <str name="classname">solr.DirectSolrSpellChecker</str>
    <str name="field">spellcheck</str>
  </lst>
  <lst name="spellchecker">
    <str name="name">en</str>
    <str name="classname">solr.DirectSolrSpellChecker</str>
    <str name="field">spellcheck_en</str>
  </lst>
  <lst name="spellchecker">
    <str name="name">cs</str>
    <str name="classname">solr.DirectSolrSpellChecker</str>
    <str name="field">spellcheck_cs</str>
  </lst>
  <lst name="spellchecker">
    <str name="name">de</str>
    <str name="classname">solr.DirectSolrSpellChecker</str>
    <str name="field">spellcheck_de</str>
  </lst>
  <lst name="spellchecker">
    <str name="name">es</str>
    <str name="classname">solr.DirectSolrSpellChecker</str>
    <str name="field">spellcheck_es</str>
  </lst>
  <lst name="spellchecker">
    <str name="name">es_CO</str>
    <str name="classname">solr.DirectSolrSpellChecker</str>
    <str name="field">spellcheck_es_co</str>
  </lst>
  <lst name="spellchecker">
    <str name="name">fr</str>
    <str name="classname">solr.DirectSolrSpellChecker</str>
    <str name="field">spellcheck_fr</str>
  </lst>
  <lst name="spellchecker">
    <str name="name">hi</str>
    <str name="classname">solr.DirectSolrSpellChecker</str>
    <str name="field">spellcheck_hi</str>
  </lst>
  <lst name="spellchecker">
    <str name="name">hu</str>
    <str name="classname">solr.DirectSolrSpellChecker</str>
    <str name="field">spellcheck_hu</str>
  </lst>
  <lst name="spellchecker">
    <str name="name">id</str>
    <str name="classname">solr.DirectSolrSpellChecker</str>
    <str name="field">spellcheck_id</str>
  </lst>
  <lst name="spellchecker">
    <str name="name">it</str>
    <str name="classname">solr.DirectSolrSpellChecker</str>
    <str name="field">spellcheck_it</str>
  </lst>
  <lst name="spellchecker">
    <str name="name">ja</str>
    <str name="classname">solr.DirectSolrSpellChecker</str>
    <str name="field">spellcheck_ja</str>
  </lst>
  <lst name="spellchecker">
    <str name="name">ko</str>
    <str name="classname">solr.DirectSolrSpellChecker</str>
    <str name="field">spellcheck_ko</str>
  </lst>
</searchComponent>
```

```

</lst>
<lst name="spellchecker">
  <str name="name">pl</str>
  <str name="classname">solr.DirectSolrSpellChecker</str>
  <str name="field">spellcheck_pl</str>
</lst>
<lst name="spellchecker">
  <str name="name">pt</str>
  <str name="classname">solr.DirectSolrSpellChecker</str>
  <str name="field">spellcheck_pt</str>
</lst>
<lst name="spellchecker">
  <str name="name">ru</str>
  <str name="classname">solr.DirectSolrSpellChecker</str>
  <str name="field">spellcheck_ru</str>
</lst>
<lst name="spellchecker">
  <str name="name">zh</str>
  <str name="classname">solr.DirectSolrSpellChecker</str>
  <str name="field">spellcheck_zh</str>
</lst>
<lst name="spellchecker">
  <str name="name">zh_TW</str>
  <str name="classname">solr.DirectSolrSpellChecker</str>
  <str name="field">spellcheck_zh_tw</str>
</lst>
</searchComponent>

```

Uncomment the following code blocks:

```

<!--
<requestHandler name="/suggest" class="solr.SearchHandler">
  <lst name="defaults">
    <str name="suggest">true</str>
    <str name="suggest.dictionary">default</str>
    <str name="suggest.count">5</str>
  </lst>
  <arr name="components">
    <str>suggest</str>
  </arr>
</requestHandler>
-->

<!--
<searchComponent name="suggest" class="solr.SuggestComponent">
  <lst name="suggester">
    <str name="name">default</str>
    <str name="dictionaryImpl">DocumentDictionaryFactory</str>
    <str name="lookupImpl">FreeTextLookupFactory</str>
    <str name="suggestAnalyzerFieldType">text_spell</str>
    <str name="suggestFreeTextAnalyzerFieldType">text_spell</str>
    <str name="field">autosuggest</str>
    <str name="buildOnCommit">true</str>
    <str name="buildOnOptimize">true</str>
    <str name="exactMatchFirst">true</str>
    <str name="separator"><![CDATA[ ]]></str>
  </lst>
  <lst name="suggester">
    <str name="name">en</str>
    <str name="dictionaryImpl">DocumentDictionaryFactory</str>
    <str name="lookupImpl">FreeTextLookupFactory</str>
    <str name="suggestAnalyzerFieldType">text_spell_en</str>
    <str name="suggestFreeTextAnalyzerFieldType">text_spell_en</str>
    <str name="field">autosuggest_en</str>
    <str name="buildOnCommit">true</str>
    <str name="buildOnOptimize">true</str>
    <str name="exactMatchFirst">true</str>
    <str name="separator"><![CDATA[ ]]></str>
  </lst>
  <lst name="suggester">
    <str name="name">cs</str>
    <str name="dictionaryImpl">DocumentDictionaryFactory</str>
    <str name="lookupImpl">FreeTextLookupFactory</str>
    <str name="suggestAnalyzerFieldType">text_spell_cs</str>
    <str name="suggestFreeTextAnalyzerFieldType">text_spell_cs</str>
    <str name="field">autosuggest_cs</str>
    <str name="buildOnCommit">true</str>
    <str name="buildOnOptimize">true</str>
    <str name="exactMatchFirst">true</str>
    <str name="separator"><![CDATA[ ]]></str>
  </lst>
  <lst name="suggester">

```



```

    <str name="name">de</str>
    <str name="dictionaryImpl">DocumentDictionaryFactory</str>
    <str name="lookupImpl">FreeTextLookupFactory</str>
    <str name="suggestAnalyzerFieldType">text_spell_de</str>
    <str name="suggestFreeTextAnalyzerFieldType">text_spell_de</str>
    <str name="field">autosuggest_de</str>
    <str name="buildOnCommit">true</str>
    <str name="buildOnOptimize">true</str>
    <str name="exactMatchFirst">true</str>
    <str name="separator"><![CDATA[ ]]></str>
</lst>
<lst name="suggester">
    <str name="name">es</str>
    <str name="dictionaryImpl">DocumentDictionaryFactory</str>
    <str name="lookupImpl">FreeTextLookupFactory</str>
    <str name="suggestAnalyzerFieldType">text_spell_es</str>
    <str name="suggestFreeTextAnalyzerFieldType">text_spell_es</str>
    <str name="field">autosuggest_es</str>
    <str name="buildOnCommit">true</str>
    <str name="buildOnOptimize">true</str>
    <str name="exactMatchFirst">true</str>
    <str name="separator"><![CDATA[ ]]></str>
</lst>
<lst name="suggester">
    <str name="name">es_CO</str>
    <str name="dictionaryImpl">DocumentDictionaryFactory</str>
    <str name="lookupImpl">FreeTextLookupFactory</str>
    <str name="suggestAnalyzerFieldType">text_spell_es_co</str>
    <str name="suggestFreeTextAnalyzerFieldType">text_spell_es_co</str>
    <str name="field">autosuggest_es_co</str>
    <str name="buildOnCommit">true</str>
    <str name="buildOnOptimize">true</str>
    <str name="exactMatchFirst">true</str>
    <str name="separator"><![CDATA[ ]]></str>
</lst>
<lst name="suggester">
    <str name="name">fr</str>
    <str name="dictionaryImpl">DocumentDictionaryFactory</str>
    <str name="lookupImpl">FreeTextLookupFactory</str>
    <str name="suggestAnalyzerFieldType">text_spell_fr</str>
    <str name="suggestFreeTextAnalyzerFieldType">text_spell_fr</str>
    <str name="field">autosuggest_fr</str>
    <str name="buildOnCommit">true</str>
    <str name="buildOnOptimize">true</str>
    <str name="exactMatchFirst">true</str>
    <str name="separator"><![CDATA[ ]]></str>
</lst>
<lst name="suggester">
    <str name="name">hi</str>
    <str name="dictionaryImpl">DocumentDictionaryFactory</str>
    <str name="lookupImpl">FreeTextLookupFactory</str>
    <str name="suggestAnalyzerFieldType">text_spell_hi</str>
    <str name="suggestFreeTextAnalyzerFieldType">text_spell_hi</str>
    <str name="field">autosuggest_hi</str>
    <str name="buildOnCommit">true</str>
    <str name="buildOnOptimize">true</str>
    <str name="exactMatchFirst">true</str>
    <str name="separator"><![CDATA[ ]]></str>
</lst>
<lst name="suggester">
    <str name="name">hu</str>
    <str name="dictionaryImpl">DocumentDictionaryFactory</str>
    <str name="lookupImpl">FreeTextLookupFactory</str>
    <str name="suggestAnalyzerFieldType">text_spell_hu</str>
    <str name="suggestFreeTextAnalyzerFieldType">text_spell_hu</str>
    <str name="field">autosuggest_hu</str>
    <str name="buildOnCommit">true</str>
    <str name="buildOnOptimize">true</str>
    <str name="exactMatchFirst">true</str>
    <str name="separator"><![CDATA[ ]]></str>
</lst>
<lst name="suggester">
    <str name="name">id</str>
    <str name="dictionaryImpl">DocumentDictionaryFactory</str>
    <str name="lookupImpl">FreeTextLookupFactory</str>
    <str name="suggestAnalyzerFieldType">text_spell_id</str>
    <str name="suggestFreeTextAnalyzerFieldType">text_spell_id</str>
    <str name="field">autosuggest_id</str>
    <str name="buildOnCommit">true</str>
    <str name="buildOnOptimize">true</str>
    <str name="exactMatchFirst">true</str>
    <str name="separator"><![CDATA[ ]]></str>
</lst>
<lst name="suggester">
    <str name="name">it</str>

```

```

    <str name="dictionaryImpl">DocumentDictionaryFactory</str>
    <str name="lookupImpl">FreeTextLookupFactory</str>
    <str name="suggestAnalyzerFieldType">text_spell_it</str>
    <str name="suggestFreeTextAnalyzerFieldType">text_spell_it</str>
    <str name="field">autosuggest_it</str>
    <str name="buildOnCommit">true</str>
    <str name="buildOnOptimize">true</str>
    <str name="exactMatchFirst">true</str>
    <str name="separator"><![CDATA[ ]]></str>
</lst>
<lst name="suggester">
    <str name="name">ja</str>
    <str name="dictionaryImpl">DocumentDictionaryFactory</str>
    <str name="lookupImpl">FreeTextLookupFactory</str>
    <str name="suggestAnalyzerFieldType">text_spell_ja</str>
    <str name="suggestFreeTextAnalyzerFieldType">text_spell_ja</str>
    <str name="field">autosuggest_ja</str>
    <str name="buildOnCommit">true</str>
    <str name="buildOnOptimize">true</str>
    <str name="exactMatchFirst">true</str>
    <str name="separator"><![CDATA[ ]]></str>
</lst>
<lst name="suggester">
    <str name="name">ko</str>
    <str name="dictionaryImpl">DocumentDictionaryFactory</str>
    <str name="lookupImpl">FreeTextLookupFactory</str>
    <str name="suggestAnalyzerFieldType">text_spell_ko</str>
    <str name="suggestFreeTextAnalyzerFieldType">text_spell_ko</str>
    <str name="field">autosuggest_ko</str>
    <str name="buildOnCommit">true</str>
    <str name="buildOnOptimize">true</str>
    <str name="exactMatchFirst">true</str>
    <str name="separator"><![CDATA[ ]]></str>
</lst>
<lst name="suggester">
    <str name="name">pl</str>
    <str name="dictionaryImpl">DocumentDictionaryFactory</str>
    <str name="lookupImpl">FreeTextLookupFactory</str>
    <str name="suggestAnalyzerFieldType">text_spell_pl</str>
    <str name="suggestFreeTextAnalyzerFieldType">text_spell_pl</str>
    <str name="field">autosuggest_pl</str>
    <str name="buildOnCommit">true</str>
    <str name="buildOnOptimize">true</str>
    <str name="exactMatchFirst">true</str>
    <str name="separator"><![CDATA[ ]]></str>
</lst>
<lst name="suggester">
    <str name="name">pt</str>
    <str name="dictionaryImpl">DocumentDictionaryFactory</str>
    <str name="lookupImpl">FreeTextLookupFactory</str>
    <str name="suggestAnalyzerFieldType">text_spell_pt</str>
    <str name="suggestFreeTextAnalyzerFieldType">text_spell_pt</str>
    <str name="field">autosuggest_pt</str>
    <str name="buildOnCommit">true</str>
    <str name="buildOnOptimize">true</str>
    <str name="exactMatchFirst">true</str>
    <str name="separator"><![CDATA[ ]]></str>
</lst>
<lst name="suggester">
    <str name="name">ru</str>
    <str name="dictionaryImpl">DocumentDictionaryFactory</str>
    <str name="lookupImpl">FreeTextLookupFactory</str>
    <str name="suggestAnalyzerFieldType">text_spell_ru</str>
    <str name="suggestFreeTextAnalyzerFieldType">text_spell_ru</str>
    <str name="field">autosuggest_ru</str>
    <str name="buildOnCommit">true</str>
    <str name="buildOnOptimize">true</str>
    <str name="exactMatchFirst">true</str>
    <str name="separator"><![CDATA[ ]]></str>
</lst>
<lst name="suggester">
    <str name="name">zh</str>
    <str name="dictionaryImpl">DocumentDictionaryFactory</str>
    <str name="lookupImpl">FreeTextLookupFactory</str>
    <str name="suggestAnalyzerFieldType">text_spell_zh</str>
    <str name="suggestFreeTextAnalyzerFieldType">text_spell_zh</str>
    <str name="field">autosuggest_zh</str>
    <str name="buildOnCommit">true</str>
    <str name="buildOnOptimize">true</str>
    <str name="exactMatchFirst">true</str>
    <str name="separator"><![CDATA[ ]]></str>
</lst>
<lst name="suggester">
    <str name="name">zh_TW</str>
    <str name="dictionaryImpl">DocumentDictionaryFactory</str>

```

```

        <str name="lookupImpl">FreeTextLookupFactory</str>
        <str name="suggestAnalyzerFieldType">text_spell_zh_tw</str>
        <str name="suggestFreeTextAnalyzerFieldType">text_spell_zh_tw</str>
        <str name="field">autosuggest_zh_tw</str>
        <str name="buildOnCommit">true</str>
        <str name="buildOnOptimize">true</str>
        <str name="exactMatchFirst">true</str>
        <str name="separator"><![CDATA[ ]]></str>
    </lst>
</searchComponent>
-->

```

After restarting the Solr server, the new suggestions method is active.

Keyword Redirects

For further information, see: [Configure Keyword Redirects](#)

All types of matches can be case-sensitive.

Redirects are search configuration and language specific. By default, we support five types of query matches:

The SAP Commerce provides a mechanism for keyword redirects. This feature allows users to define certain words that, when matched during the search, work like triggers causing the client to redirect to a specified search results.

- EXACT: Query must be equal to the given phrase.
- STARTS_WITH: Query must start with the given phrase.
- ENDS_WITH: Query must end with the given phrase.
- CONTAINS: Query must contain the given phrase.
- REGEX: Query must match the given expression.

Related Information

[Solr Server Installation](#)

[Schema.xml](#) ➔

[Solr Config.xml](#) ➔

Solr Server Installation for Standalone Mode

Learn how to install and configure a standalone Solr server in a way that can be used in SAP Commerce.

Quick Installation and Configuration

Context

Follow the steps for a quick installation and configuration of the solr server on your local machine.

Procedure

1. Enable the `solrserver` extension in your `localextensions.xml` file.

The extension is now ready for use. For further information on configuring extensions, see [Installation Based on Specified Extensions](#).

2. Create a Solr server instance. You can use the default instance included in the `solrserver` extension.

The default configuration is as follows:

```

solrserver.instances.default.autostart=true
solrserver.instances.default.mode=standalone
solrserver.instances.default.hostname=localhost

```

```
solrserver.instances.default.port=8983
solrserver.instances.default.memory=512m
```

If you need, you can override some of the properties in your `local.properties` file. For further information, see: [solrserver Extension](#).

3. Start the SAP Commerce server.

Because autostart is enabled for the default Solr server instance, the Solr server is started and stopped together with the platform. For further information, see: [Installing SAP Commerce Manually](#) and [Installing SAP Commerce Using Installer Recipes](#).

4. Configure SAP Commerce to use the Solr server.

The most important attributes you should consider when updating the Solr server configuration are:

- **Mode:** set it to **standalone**.
- **Endpoint URLs:** should contain a single master URL with the following value: `http://localhost:8983/solr`.

For details on how to easily configure the Solr server settings with the Backoffice Administration Cockpit see: [Create Facet Search Configuration](#).

Solr Server Installation

Using the solrserver Extension

By enabling the `solrserver` extension in your `localextensions.xml` file, you get a Solr server instance ready to be used. The default configuration ensures that a standalone Solr server is configured, started and stopped together with the platform. This is convenient in CI or developer environments. Note that the mode property should be set to **standalone**.

The following code shows an example of a configuration:

```
# disables the autostart for the default Solr instance
solrserver.instances.default.autostart=false

solrserver.instances.standalone.autostart=true
solrserver.instances.standalone.mode=standalone
solrserver.instances.standalone.hostname=localhost
solrserver.instances.standalone.port=8983
solrserver.instances.standalone.memory=512m
```

Using an External Standalone Server

Each version of SAP Commerce can support more than one Solr version to maintain compatibility. Each version includes the **server** directory that contains the server and the applied customizations, and the **customizations** directory that contains modifications that can be applied to an official Solr release. For the installation steps, see the **Installation** section in [Multiple Solr Version Support](#).

Use the following commands to start and stop the Solr server:

i Note

On Unix-based systems the `bin/solr` script may not be executable by default. To make it executable, run the following command in the Solr server root directory:

```
:#chmod +x bin/solr
```

- On Linux systems:

```
# ./bin/solr start -p 8983
Waiting up to 30 seconds to see Solr running on port 8983 [/]
Started Solr server on port 8983 (pid=23092). Happy searching!
```

```
# ./bin/solr stop -p 8983
```

```
Sending stop command to Solr running on port 8983 ... waiting 5 seconds to allow Jetty process 23092 to sto
```

- On Windows systems:

```
# bin\solr.cmd start -p 8983
```

```
Waiting up to 30 to see Solr running on port 8983
```

```
Started Solr server on port 8983. Happy searching!
```

```
# bin\solr.cmd stop -p 8983
```

```
Stopping Solr process 6256 running on port 8983
```

```
Waiting for 0 seconds, press a key to continue ...
```

Cluster Configuration

This section describes how to set up a cluster of Solr servers using replication. This is especially useful when:

- You have a high system load caused by searches and you want to balance the load among several servers
- Indexing consumes many resources, so you need to separate indexing and searching

In a cluster configuration, there's a leading server where indexing happens, and one or more subordinate servers. All indexes on the leading server are replicated to each one of the subordinate servers so that both leading and subordinate servers have the same indexes.

Note

The full indexing operation doesn't work unless the subordinate server is running. However, the server downtime isn't an issue during querying or index **updates**. In such situations, it only decrements the overall capacity of the clustered Solr setup. If your slave Solr instance is down during updates, your master Solr instance is indexed and serves the incoming queries.

You can enable an index replication feature in the Solr instance out of the box. Replication requests are handled by the `ReplicationHandler`, which can be configured in the `solrconfig.xml` file. By default, the corresponding section of the `solrconfig.xml` file is commented out:

```
<requestHandler name="/replication" class="solr.ReplicationHandler" >
  <!--
    To enable simple master/slave replication, uncomment one of the
    sections below, depending on whether this solr instance should be
    the "master" or a "slave". If this instance is a "slave" you will
    also need to fill in the masterUrl to point to a real machine.
  -->
  <lst name="master">
    <str name="replicateAfter">commit</str>
    <str name="replicateAfter">startup</str>
    <str name="confFiles">schema.xml,stopwords.txt,synonyms.txt</str>
  </lst>
  <!--
  <lst name="slave">
    <str name="masterUrl">http://your-master-hostname:port/solr/${solr.core.name}/replication</str>
    <str name="pollInterval">00:00:60</str>
  </lst>
  -->
</requestHandler>
```

The following are configuration examples using the `solrserver` extension:

- `local.properties` file:

```
solrserver.instances.master.autostart=true
solrserver.instances.master.mode=standalone
solrserver.instances.master.hostname=localhost
solrserver.instances.master.port=8983
solrserver.instances.master.memory=512m
```

```
solrserver.instances.slave.autostart=true
solrserver.instances.slave.mode=standalone
solrserver.instances.slave.hostname=localhost
solrserver.instances.slave.port=8984
solrserver.instances.slave.memory=512m
```

- config/solr/instances/master/configsets/default/conf/solrconfig.xml file:

```
<requestHandler name="/replication" class="solr.ReplicationHandler" >
  <lst name="master">
    <str name="replicateAfter">commit</str>
    <str name="replicateAfter">startup</str>
    <str name="confFiles">schema.xml,stopwords.txt,synonyms.txt</str>
  </lst>
</requestHandler>
```

- config/solr/instances/slave/configsets/default/conf/solrconfig.xml file:

```
<requestHandler name="/replication" class="solr.ReplicationHandler" >
  <lst name="slave">
    <str name="masterUrl">http://your-master-hostname:port/solr/${solr.core.name}/replication</str>
    <str name="pollInterval">00:00:30</str>
  </lst>
</requestHandler>
```

SAP Commerce Configuration in Backoffice Administration Cockpit

Update the Solr server configuration using the Backoffice Administration Cockpit.

The most important attributes you should consider are:

- **Mode:** set it to **standalone**.
- **Endpoint URLs:** for URLs of the servers that are part of the cluster - one of them should be set as master, for example, `http://master:8983/solr`, `http://slave:8983/solr`.

For details on how to easily configure the Solr server settings with the Backoffice Administration Cockpit see [Configure the Solr Server](#).

Cloud Solr Server Advanced Configuration

Learn how to install and configure a cloud Solr server in a way that can be used in SAP Commerce.

Using the solrserver Extension

By enabling the `solrserver` extension in your `localextensions.xml` file you will get a Solr server instance ready to be used. For the quick installation guide, see [Cloud Solr Server Quick Installation](#).

Using an External Standalone Server

Each version of SAP Commerce has the possibility of supporting more than one Solr version to maintain compatibility. Each version includes the **server** directory containing the server and the applied customizations, and the **customizations** directory containing modifications that can be applied to an official Solr release. For the installation steps, see the **Installation** section in [Multiple Solr Version Support](#).

Use the following commands to start and stop the Solr server:

i Note

On unix-based systems the bin/solr script may not be executable by default. To make it executable, run the following command in the Solr server root directory:

```
:#chmod +x bin/solr
```

- On linux systems:

```
# ./bin/solr start -p 8983
Waiting up to 30 seconds to see Solr running on port 8983 [/]
Started Solr server on port 8983 (pid=23092). Happy searching!

# ./bin/solr stop -p 8983
Sending stop command to Solr running on port 8983 ... waiting 5 seconds to allow Jetty process 23092 to stop
```

- On windows systems:

```
# bin\solr.cmd start -p 8983
Waiting up to 30 to see Solr running on port 8983
Started Solr server on port 8983. Happy searching!

# bin\solr.cmd stop -p 8983
Stopping Solr process 6256 running on port 8983
Waiting for 0 seconds, press a key to continue ...
```

Zookeeper Authentication and Authorization

By default, the authentication and authorization is **not** enforced when a server attempts to join a quorum. That's why it is recommended to turn them on. For instructions on how to do it, see [Authorization and Authentication Plugins in Solr Cloud Mode](#) 📖 .

Cluster Configuration

This section describes how to set up a cluster of Solr servers using SolrCloud. This is especially useful when you need advanced fault tolerance and high availability capabilities.

SolrCloud is flexible distributed search and indexing, without a leading node to allocate nodes, shards and replicas. Instead, Solr uses ZooKeeper to manage these locations, depending on configuration files and schemas. Queries and updates can be sent to any server. Solr uses the information in the ZooKeeper database to figure out which servers need to handle the request.

For more details on the cloud see: <https://cwiki.apache.org/confluence/display/solr/SolrCloud> 📖 .

Configuration example (using the solrserver extension):

- local.properties file:

```
solrserver.instances.cloud1.autostart=false
solrserver.instances.cloud1.mode=cloud
solrserver.instances.cloud1.port=8983
solrserver.instances.cloud1.memory=512m
solrserver.instances.cloud1.zk.host=
solrserver.instances.cloud1.zk.upconfig=true
solrserver.instances.default.autostart=false
solrserver.instances.cloud1.hostname=<hostIPAddress> / localhost

solrserver.instances.cloud2.autostart=false
solrserver.instances.cloud2.mode=cloud
solrserver.instances.cloud2.port=8984
solrserver.instances.cloud2.memory=512m
solrserver.instances.cloud2.zk.host=localhost:9983
```

```
solrserver.instances.cloud2.zk.upconfig=false
solrserver.instances.cloud2.hostname=<hostIPAddress> / localhost
```

i Note

Note that the autostart of the servers is disabled because the order in which the Solr servers are started cannot be guaranteed. In this example the Solr server for instance cloud1 has to be started before the server for instance cloud2. This is because the Solr server for instance cloud1 also starts an embedded ZooKeeper server.

- Starting the servers:

```
# ant startSolrServer -Dinstance.name=cloud1
# ant startSolrServer -Dinstance.name=cloud2
```

Configuration in Backoffice Administration Cockpit

Update the Solr server configuration using the Backoffice Administration Cockpit.

The most important attributes you should consider are:

- **Mode:** set it to `cloud`.
- **Endpoint URLs:** should contain one or more urls of the ZooKeeper servers. The urls should follow the pattern `{hostname}:{port}`:
`localhost:9983`

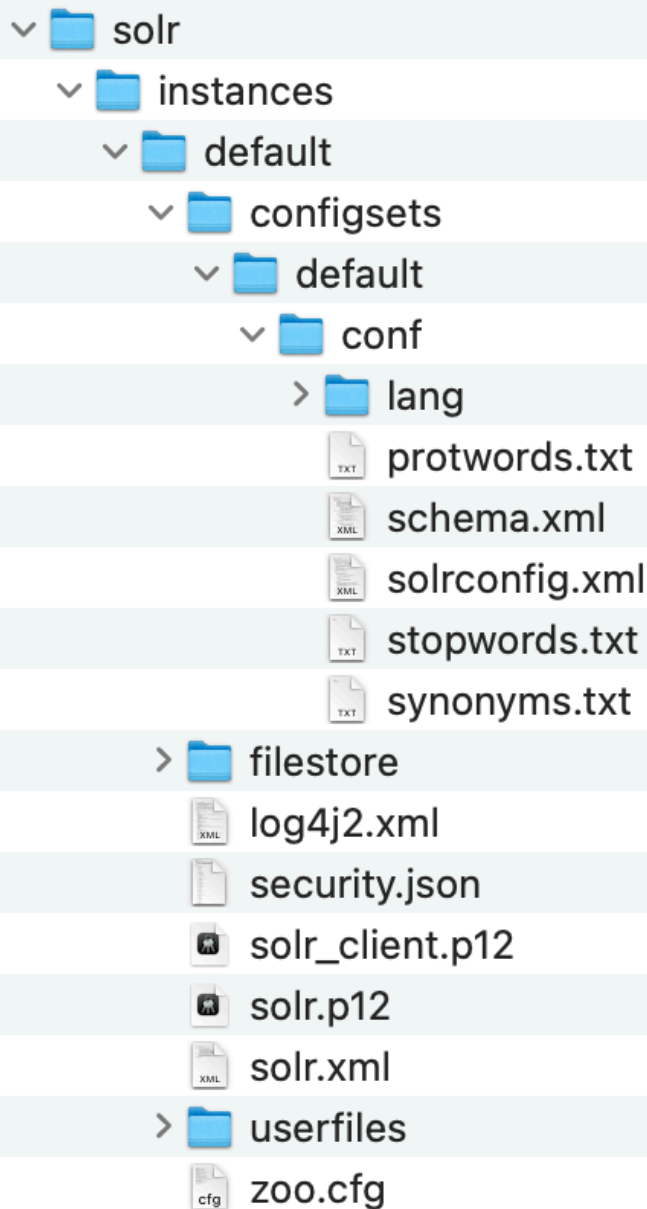
For details on how to easily configure the Solr server settings with the Backoffice Administration Cockpit see: [Create Facet Search Configuration](#).

Configure and Manage Solr Instances

To work properly, you need to configure the Solr server using a set of properties.

Solr Server Configuration Files

Each instance has its own set of configuration files that is placed under the instance configuration directory. Those files are created automatically when the instance is created. By default, it has a structure similar to the following:



Solr Instance General Settings

Define the general settings for your instances using the `project.properties` file. For details, see [Solr Instance Properties](#).

Managing Solr Instances

Solr instances and servers may be managed:

- **Automatically:** Solr servers for instances with `autostart=true` are started when the Spring global context is being initialized and are stopped when the context is being closed. For more details, see [Automatic Start of Solr Server](#).
- **Manually:** You can use ant tasks to manage your instances. For details, see [Solr Server Ant Tasks](#).

i Note

If the corresponding Solr instance does not exist when starting a Solr server, it's created automatically.

Solr Instance Properties

Functionality provided by the extension allows for managing of more than one instance. Each instance has a name and can be configured using the properties in the `local.properties` file.

You need to configure the `keyStore` and `trustStore` properties for both Server and Client modes. Refer the following table to learn more about instance properties:

Property Name	Default Value	Description
<code>solrserver.instances.<instance name>.autostart</code>	false	If set to true , a given Solr instance is started/stopped automatically with the Platform.
<code>solrserver.instances.<instance name>.mode</code>	standalone	The mode in which the Solr server should be started: it can be standalone or cloud .
<code>solrserver.instances.<instance name>.zk.host</code>		Start Solr with the defined ZooKeeper connection string. If this option isn't provided, Solr will start the embedded ZooKeeper instance. This is also used for configuration upload. Example: <code>server1:2181, server2:2181</code> .
<code>solrserver.instances.<instance name>.zk.upconfig</code>	true	If set to true , the configuration will be uploaded to the ZooKeeper server after the Solr server starts up.
<code>solrserver.instances.<instance name>.zk.prop.<property name></code>		Sets cluster-wide properties in ZooKeeper.
<code>solrserver.instances.<instance name>.hostname</code>	localhost	Starts the Solr server with the defined hostname.
<code>solrserver.instances.<instance name>.port</code>	8983	The TCP port that is used by Solr server (each instance should use a different port).
<code>solrserver.instances.<instance name>.config.dir</code>	<code><HYBRIS_CONFIG_DIR>/solr/instances/<instance name></code>	A directory that contains Solr instance configuration.
<code>solrserver.instances.<instance name>.data.dir</code>	<code><HYBRIS_DATA_DIR>/solr/instances/<instance name></code>	A directory that contains Solr instance data.
<code>solrserver.instances.<instance name>.log.dir</code>	<code><HYBRIS_LOG_DIR>/solr/instances/<instance name></code>	A directory that contains Solr instance logs.
<code>solrserver.instances.<instance name>.memory</code>	512 m	The amount of memory (heap) to be used by the Solr server process.
<code>solrserver.instances.<instance name>.javaoptions</code>		An additional command line options to be used by the Solr server process.
<code>solrserver.instances.<instance name>.authtype</code>	basic	The authentication type, only basic is supported. An empty value disables authentication.
<code>solrserver.instances.<instance name>.user</code>		The user used for authentication.
<code>solrserver.instances.<instance name>.password</code>		The password used for authentication.
<code>solrserver.instances.<instance name>.ssl.enabled</code>	true	If set to true, SSL is enabled.
For server mode configuration		
<code>solrserver.instances.<instance name>.ssl.keyStoreType</code>	PKCS12	The keyStore type.
<code>solrserver.instances.<instance name>.ssl.keyStore</code>	<code>solr.p12</code> file in the instance configuration directory	The keyStore file.

Property Name	Default Value	Description
solrserver.instances.<instance name>.ssl.keyStorePassword		The keyStore password.
solrserver.instances.<instance name>.ssl.trustStoreType	PKCS12	The trustStore type.
solrserver.instances.<instance name>.ssl.trustStore	solr.p12 file in the instance configuration directory	The trustStore file.
solrserver.instances.<instance name>.ssl.trustStorePassword		The trustStore password.
For client mode configuration		
solrserver.instances.<instance name>.ssl.client.keyStoreType	PKCS12	The client keyStore type.
solrserver.instances.<instance name>.ssl.client.keyStore	solr.p12 file in the instance configuration directory	The keyStore file.
solrserver.instances.<instance name>.ssl.client.keyStorePassword		The client keyStore password.
solrserver.instances.<instance name>.ssl.client.trustStoreType	PKCS12	The client trustStore type.
solrserver.instances.<instance name>.ssl.client.trustStore	solr.p12 file in the instance configuration directory	The client trustStore file.
solrserver.instances.<instance name>.ssl.client.trustStorePassword		The client trustStore password.
solrserver.instances.<instance name>.ssl.needClientAuth	false	If set to true, require clients to authenticate.
solrserver.instances.<instance name>.ssl.wantClientAuth	false	If set to true, enable clients to authenticate (but not required).

Automatic Start of Solr Server

Procedure

1. Check if the corresponding Solr is running. If no, go to Step 6.
2. Check if the running Solr server is the proper one. If not, throw an error.
3. If `forceRestart` property is set to **false** go to step 6.
4. Stop the Solr server.
5. Check if the Solr server is properly stopped. If not, throw an error.
6. Start the Solr server.
7. Check if the Solr server is properly started. If not, throw an error.

Some parts for the algorithm can be customized via properties in the `local.properties` file. The following properties are available:

Property Name	Default Value	Description
solrserver.forceRestart	true	If a proper Solr server is already running when starting the platform, it will be restarted.
solrserver.failOnError	true	If an error occurs during the start of any Solr server, it is treated as serious error and whole platform will not run.

Solr Server Ant Tasks

The `solrserver` extension provides some additional ant tasks that can be used to control the Solr instances and servers.

By using a parameter it is possible to override some of the instance properties. The default instance name is `default`. The following ant tasks are available:

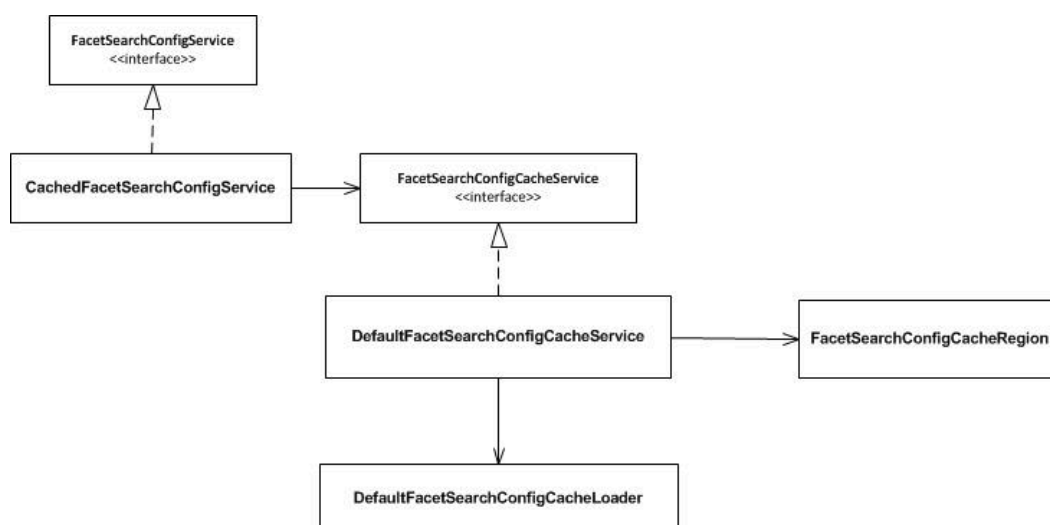
Task Name	Description	Parameters
<code>createSolrInstance</code>	Creates a Solr instance.	allowed parameters: <ul style="list-style-type: none"> <code>-Dinstance.name</code>
<code>deleteSolrInstance</code>	Deletes a Solr instance. This means deleting all the configuration, data and log files.	allowed parameters: <ul style="list-style-type: none"> <code>-Dinstance.name</code>
<code>uploadSolrConfig</code>	Uploads configuration to a ZooKeeper instance.	allowed parameters: <ul style="list-style-type: none"> <code>-Dinstance.name</code>
<code>startSolrServer</code>	Starts the Solr server.	allowed parameters: <ul style="list-style-type: none"> <code>-Dinstance.name</code>
<code>startSolrServers</code>	Starts all Solr servers that have <code>autostart</code> set to true.	
<code>stopSolrServer</code>	Stops the Solr server.	allowed parameters: <ul style="list-style-type: none"> <code>-Dinstance.name</code>
<code>stopSolrServers</code>	Stops all Solr servers that have <code>autostart</code> set to true.	
<code>configureSolrServer</code>	Applies the SAP Commerce customizations on an official Solr release.	allowed parameters: <ul style="list-style-type: none"> <code>-DsolrServerPath</code>

Solr Configuration Caching

This document describes the Solr configuration cache.

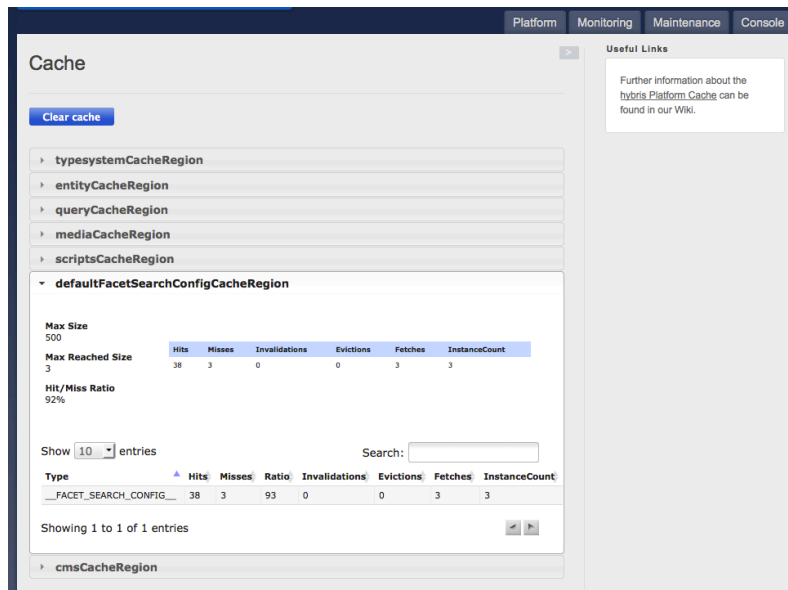
Facet search config cache was created to improve the performance for the solr search functionality. The config cache is used by `cachedFacetSearchConfigService` which is defined in the `solrfacetsearch-spring.xml` file. By default it has the `facetSearchConfigService` alias assigned.

The figure presents an overview of the class structure used to implement the facet search config cache.



Cache Region for the FacetSearchConfig Object

A cache for the FacetSearchConfig objects was implemented as one of the hybris platform region caches. It can be monitored in the SAP Commerce Administration Console (Administration Console).



Definition of this region can be found in the `global-solrfacetsearch-spring.xml` file as shown in the following example:

```
<!-- Solr facet search config cache -->
<alias name="defaultFacetSearchConfigCacheRegion" alias="facetSearchConfigCacheRegion" />
<bean name="defaultFacetSearchConfigCacheRegion" class="de.hybris.platform.solrfacetsearch.config.cache.impl.
  <constructor-arg name="name" value="defaultFacetSearchConfigCacheRegion"/>
  <constructor-arg name="maxEntries" value="${facetsearchconfig.cache.maxEntries}"/>
  <constructor-arg name="statsEnabled" value="${regioncache.stats.enabled}"/>
  <property name="handledTypes">
    <array>
      <value>__FACET_SEARCH_CONFIG__</value>
    </array>
  </property>
</bean>
<bean id="facetSearchConfigCacheRegionRegistrar" class="de.hybris.platform.regioncache.region.CacheRegionRegis
```

See also: [Region Cache](#).

Cache Invalidation

The FacetSearchConfig data is created based on many model objects and it needs to be invalidated each time one of these objects is changed. The invalidation mechanism was implemented based on the assumption that search config does not change very often. The entire cache is cleared, whenever any model object of type related to FacetSearchConfig data is changed. InvalidationListener is defined in the DefaultFacetSearchConfigCacheService.

For further information, see: [Specifying a Deployment for hybris Platform Types](#).

A set of type codes for which the cache should be invalidated is created in the `solrfacetsearch-spring.xml` file. The type code values correspond to the `type code` attribute from the `<deployment>` element in the type definition (`...items.xml`).

The following code excerpt is from the `solrfacetsearch-spring.xml` file:

```
<bean id="defaultFacetSearchConfigInvalidationTypeSet"
  class="de.hybris.platform.solrfacetsearch.config.cache.FacetSearchConfigInvalidationTypeSet">
  <constructor-arg>
    <set>
      <value>2200</value> <!-- SolrFacetSearchConfig -->
      <value>2204</value> <!-- SolrValueRangeSet -->
      <value>2205</value> <!-- SolrValueRange -->
      <value>2206</value> <!-- SolrSearchConfig -->
      <value>2207</value> <!-- SolrIndexedType -->
      <value>2208</value> <!-- SolrIndexedProperty -->
      <value>2209</value> <!-- SolrIndexerQuery -->
      <value>2210</value> <!-- SolrIndexerQueryParameter -->
      <value>2211</value> <!-- SolrIndexConfig -->
      <value>2212</value> <!-- SolrServerConfig -->
      <value>2213</value> <!-- SolrEndpointUrl -->
    </set>
  </constructor-arg>
</bean>
```

```

        <value>2218</value> <!--SolrSynonymConfig -->
        <value>2221</value> <!--SolrAbstractKeywordRedirect -->
        <value>2222</value> <!--SolrStopWord -->
        <value>2223</value> <!--SolrFacetSearchKeywordRedirect -->
        <value>32</value> <!--Language -->
        <value>33</value> <!--Currency -->
        <value>601</value> <!--CatalogVersion -->
        <value>610</value> <!--ClassAttributeAssignment -->
        <value>2013</value> <!--BaseStore -->
    </set>
</constructor-arg>
</bean>

```

The set of type codes can be extended by adding another bean of type

`de.hybris.platform.solrfacetsearch.config.cache.FacetSearchConfigInvalidationTypeSet`.

Example configuration:

```

<bean id="someTypeSet" class="de.hybris.platform.solrfacetsearch.config.cache.FacetSearchConfigInvalidationTypeSet"
    <constructor-arg>
        <set>
            <value>9251</value>
            <value>9256</value>
        </set>
    </constructor-arg>
</bean>

```

Adding Currencies to the Solr Facet Search

Adding a new currency to your storefront and assigning it to products, allows you to display products in that currency. Before customers can filter search results for the currency, you must add the currency to the Solr facet search.

Prerequisites

- You have added a currency to the required storefront.
- You have assigned prices to your products in the currency.

Context

Before customers can filter their products, you must add a new currency to the Solr facet configuration and assign a range for the currency.

Procedure

1. In Backoffice, assign the currency to the Solr facet config.
2. Go to **Search > Search and navigation > Ranges**.
3. Create a new **SolrValueRangeSet** for the new currency.
4. Enter a name, select the **double** range type, and click **Done**.
The range set is created and opens in the editor.
5. On the **Ranges** tab, under **Essential**, enter a qualifier for the range set.
6. Under **Ranges**, add a new range to the type.

Ensure that the maximum value of the range is at least as high as the highest price of your products.

7. On the **Administration** tab, under **Unbound**, select the unbound property **price**.
8. Perform a full index on the correct Solr index.

Results

The currency and range that you created are visible in the storefront.

Solr Security

The Solr security features include support for encrypting communication to and from Solr (as well as between the Solr nodes) using SSL and support for authentication and authorization provided by the Solr security frameworks. The security features are enabled by default when you use `solrserver` extension. In other case, you can use the provided sample configuration that you can easily enable.

When installing the Apache Solr, you should bear in mind the following security considerations:

- While Apache Solr provides some security features, it is not recommended to expose it to the outside world, like for example the Internet. The Solr servers should be placed in a demilitarized zone (DMZ) behind a firewall.
- Always use a secure client computer and web browser when using Apache Solr administration interface. Be mindful of the risk of [CSRF](#) attacks - always end the browser session and logout when done with your task, and avoid opening potentially corrupted web sites and emails while being authenticated with administration console.
- Follow standard hardening procedures for web applications, for example:
 - Solr servers should not be run as a root or with administrator privileges
 - Access permissions to configuration files and data files should be restricted
- If you index user-generated content like reviews or comments, then keep in mind that cross-site scripting (XSS) attacks are a risk if the search results are shown. To avoid this risk, the content needs to be escaped properly to prevent it from being wrongly interpreted.

Secure Communication Using SSL

To enable secure communication using SSL, you need:

- a valid SSL certificate.
- set SSL-related system properties.

We provide a sample `solr.jks` file, containing the certificates that can be user for development.

i Note

Sample SSL certificates are provided for developer environments and allow you to use localhost only. For production environments new certificates should be generated. The system needs to trust the new certificates, so remember to add them to default JVM trust store or configure an additional trust store in a similar way as the platform does for the sample certificates. For an example, see the `advanced.properties` file in `platform/resources`:

```
# Additional trust store. If configured the trust store (in the JKS format) is added as a fallback trust store
# by the JVM. Its intention is to provide a trusted self-signed CA certificate for developers/testers convenience
additional.java.net.ssl.trustStore=${platformhome}/resources/devcerts/ydevelopers.jks
additional.java.net.ssl.trustStorePassword=your_password
```

For details on different options related to enabling SSL, see [Enabling SSL](#).

Solr Server Configuration

Using the solrserver Extension

To enable SSL when using `solrserver` extension, add the following properties to the `local.properties` file:

```
solrserver.instances.<instance name>.ssl.enabled=true
solrserver.instances.<instance name>.ssl.keyStorePassword=your_password
solrserver.instances.<instance name>.ssl.trustStorePassword=your_password
```

The default configuration assumes that there is a `keyStore/trustStore` file named `solr.jks` in the instance configuration directory.

When you use cloud mode (SolrCloud) add also the following property:

```
solrserver.instances.<instance name>.zk.prop.urlScheme=https
```


i Note

Some cluster-wide properties (such as `urlScheme`) need to be set before any Solr node starts up. the Solr server will be restarted to make sure the configuration is applied properly.

To completely disable SSL set the following properties:

```
solrserver.instances.<instance name>.ssl.enabled=false
solrserver.instances.<instance name>.ssl.keyStorePassword=
solrserver.instances.<instance name>.ssl.trustStorePassword=
solrserver.instances.<instance name>.zk.prop.urlScheme=http
```

Using an External Standalone Server

SSL is not enabled by default when you use an external standalone server. To enable it, follow the instructions in [Enabling SSL](#) . To make it easier for you, we provide a sample configuration including :

- a `server/solr/solr.jks` file which contains the sample certificates,
- a `bin/solr.in.cmd` and `bin/solr.in.sh` files defining the sample configuration (they are both commented by default).

Solr Server Configuration in Backoffice Administration Cockpit

The only required change is to change the Solr server url from **http** to **https**. For example, instead of `http://localhost:8983/solr` it should be `https://localhost:8983/solr`. Use the Backoffice Administration Cockpit to access the Solr server configuration and set the new value: [Configure the Solr Server](#).


Authentication and Authorization Support for Solr

All authentication and authorization configuration is stored in the `security.json` file. A sample `security.json` file is provided which contains the following users:


i Note

For each username listed in the table, use the password you defined for that account.

User	Description
solradmin	admin user, can perform any operation
solrserver	for communication between server nodes
solrclient	can perform search queries
solrindexingclient	can perform search queries and indexing

For details on different options and plugins, see [Authentication and Authorization Plugins](#) .

i Note

For production environments new passwords should be generated, for example, the password can be changed using the Solr API. For details, see [Adding a User or Editing a Password](#) .

Solr Server Configuration

Using the solrserver Extension

To enable authentication and authorization when using the `solrserver` extension, add the following properties to the `local.properties` file:



```
solrserver.instances.<instance name>.authtype=basic
solrserver.instances.<instance name>.user=solrserver
solrserver.instances.<instance name>.password=your_password
```

The `security.json` file is located in the instance configuration directory. For the complete list of supported properties, see [solrserver Extension](#).

To completely disable authentication and authorization, remove the `security.json` file and set the following properties:

```
solrserver.instances.<instance name>.authtype=
solrserver.instances.<instance name>.user=
solrserver.instances.<instance name>.password=
```

Using an External Standalone Server

The authentication and authorization is not enabled by default when using an external standalone server. To enable it you can follow the instructions on the Solr reference guide in [Authentication and Authorization Plugins](#) .

The sample configuration is provided in the `server/solr/security.json.example` file. To enable it, rename the file to `security.json`.

Solr Server Configuration in Backoffice Administration Cockpit

The only required change is to configure a user and password for the Solr clients in the Solr server configuration. For instructions, see [Configure the Solr Server](#).

Multiple Solr Version Support

SAP Commerce supports more than one version of Solr server.

Apache Solr is under active development, which results in frequent feature releases on the current major version. The previous major version still receives some security and bug fixes for feature releases as the Long Term Support (LTS) version. Older versions are considered End of Mainstream Maintenance (EOMM) and are not updated further. As a result you can use more than one version of the Solr server for each version of SAP Commerce.

Compatibility of Solr Versions

Depending on the current release version, there can be several different versions of the Apache Solr server involved.

The following table provides a brief compatibility overview:

SAP Commerce Search & Navigation Module	Solr Server Version	Comments
6.7	7.2.x ²	Delivered with 7.7.x from patch 6.70.15
1808	7.4.x ²	Delivered with 7.7.x from patch 1808.10
1811	7.5.x ²	Delivered with 7.7.x from patch 1811.6
1905	7.7.x ² and higher	
2005	8.4.x ² and higher	
2011	8.6.3 and higher	
2105	8.9.0 and higher	
2205	8.11.1 and higher	

^[1] Use the Solr server from a previous SAP Commerce version and set the legacy mode attribute in the indexer configuration to true.

^[2] x indicates the version with backwards-compatible bug fixes.

Directory Structure

The directory structure was changed to provide support for multiple versions. The directory name is composed of only the major and minor part of the version, because customizations and server may have different patch versions. You can find the details on the included versions in the `meta.properties` files (the examples are provided below).

i Note

Only a single Solr server is included in the release and its version may change over time, even for maintenance releases. The provided Solr server can be useful for CI or developer environments, but for production systems use an external standalone server.

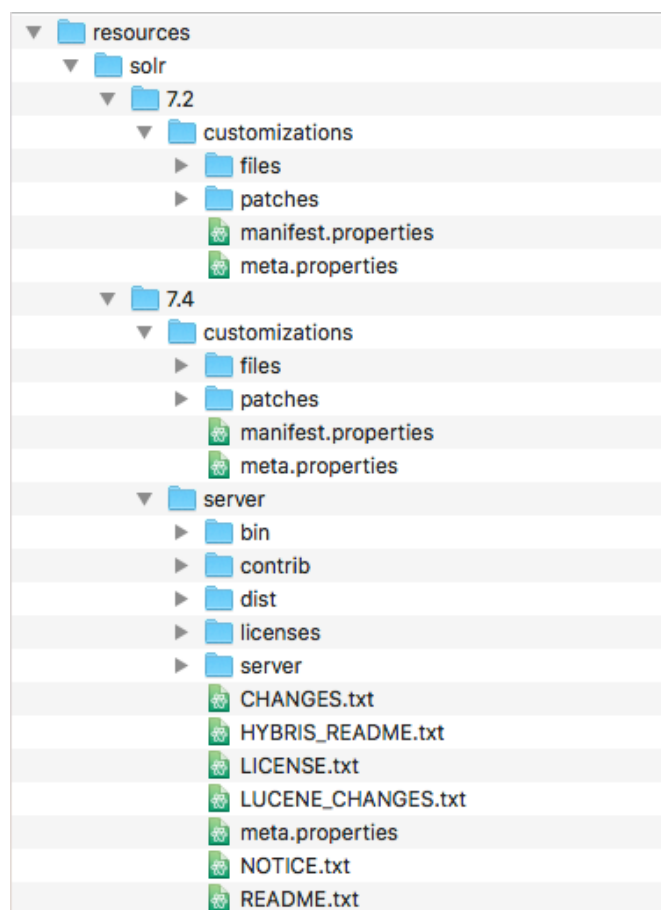
An example of `solrserver/resources/7.4/server/meta.properties`:

```
version=7.4.0
customizationsVersion=7.4.3
```

An example of `solserver/resources/7.4/customizations/meta.properties`:

```
version=7.4.3
```

The structure of the directory is as follows:



The following directories exist for each version:

- **server:** contains the Solr server with applied customizations.
- **customizations:** contains the modifications to be applied to the official Solr release. These modifications make the Solr server more lightweight and compatible with SAP Commerce platform. For the list of changes, see the `HYBRIS_README.txt` file.

Installation of an External Solr Server

Configure an external standalone Solr server for local development in SAP Commerce.

Procedure

1. Download an official Solr release from <https://solr.apache.org/downloads.html> .
2. Extract the contents of the archive to the directory.
3. Apply the SAP Commerce customizations using the following command:

```
ant configureSolrServer -DsolrServerPath=/path/to/solr
```

Common Functionality

A part of functionality used in Search and Navigation module is common for search and indexing. For example, both of them use listeners API.

Listeners API

You can use listeners to intercept and customize some parts of the indexing and search processes.

Listeners follow priority rules. Before methods, the listeners with higher priority are called before the listeners with lower priority. After methods, they are called in reverse order. The following priorities apply, from highest to lowest: global listeners, listeners configured in the FacetSearchConfig, and finally, listeners configured for an IndexedType.

→ Tip

The `solrQueryDebuggingListener` logs information about the parsed query. It can be helpful to solve the problems encountered during the development stage.

Creating a Custom Listener

To create a listener, you need to implement one of the supported listener types. The following code example shows how to create a listener:

Custom Listener Example

```
public class EnableLocalizationFallbackListener implements IndexerListener, IndexerBatchListener
{
    private I18NService i18nService;

    @Required
    public void setI18NService(final I18NService i18nService)
    {
        this.i18nService = i18nService;
    }

    public I18NService getI18NService()
    {
        return i18nService;
    }

    @Override
    public void beforeIndex(final IndexerContext context) throws IndexerException
    {
        enableLocalizationFallback(context.getFacetSearchConfig(), context.getIndexedType());
    }

    @Override
    public void afterIndex(final IndexerContext context) throws IndexerException
    {
        // NOOP
    }

    @Override
    public void afterIndexError(final IndexerContext context) throws IndexerException
    {
        // NOOP
    }
}
```

```

    }

    @Override
    public void beforeBatch(final IndexerBatchContext batchContext) throws IndexerException
    {
        enableLocalizationFallback(batchContext.getFacetSearchConfig(), batchContext.getIndexType());
    }

    @Override
    public void afterBatch(final IndexerBatchContext batchContext) throws IndexerException
    {
        // NOOP
    }

    @Override
    public void afterBatchError(final IndexerBatchContext batchContext) throws IndexerException
    {
        // NOOP
    }

    protected void enableLocalizationFallback(final FacetSearchConfig facetSearchConfig, final IndexedType indexedType)
    {
        i18nService.setLocalizationFallbackEnabled(true);
    }
}

```

The following example shows the Spring configuration:

```

<bean id="enableLocalizationFallbackListener" class="com.example.EnableLocalizationFallbackListener">
    <property name="i18nService" ref="i18nService" />
</bean>

```

Registering a Listener

Registering a Listener Globally

You can register a listener globally by adding a Spring configuration. The default priority is 100--if it is not specified as part of the Spring bean declaration. Priorities above 1000 are reserved to be used by the system. The following is an example of the Spring configuration:

```

<bean id="enableLocalizationFallbackListenerDefinition" parent="solrListenerDefinition">
    <property name="priority" value="150" />
    <property name="listener" ref="enableLocalizationFallbackListener" />
</bean>

```

Facet Search Configuration

Add the Spring bean IDs of the listeners to the listeners attribute in the Facet Search Configuration. The listeners that appear first in the list have the highest priority.

IndexedType

Add the Spring bean IDs of the listeners to the listeners attribute in the IndexedType. Listeners that appear first in the list have the highest priority.

Indexing Functionality

There are a number of different topics relative to the indexing functionality. Each topic is documented in a separate guide document.

[Indexing Process](#)

Get to know more about the indexing types to be able to successfully start the indexing process and deal with potential errors.

[Indexing API](#)

This document provides information on the top level APIs for indexing functionality and how to perform some simple customizations.

[Value Provider API](#)

Learn how you can write a value provider using the available API.

[Document Identifiers](#)

Every indexed document needs to have a unique identifier. The default one should work in most cases, however you may also define a custom one.

Indexing Process

Get to know more about the indexing types to be able to successfully start the indexing process and deal with potential errors.

Since indexing can be time demanding, the items are split into batches and indexed in parallel.

There are two ways for making it work:

- Using an Indexer worker (for each worker a thread is assigned)
- Using distributed processing and task engine, which is a proposed solution for indexing in clustered environment.

By changing value of `distributedIndexing` attribute of the Solr configuration, it is possible to switch from one to other way of indexing. In case of distributed indexing, it is possible to configure node group of a cluster process that will be executed (`nodeGroup` attribute of Solr configuration).

Indexing Operations

Instead of indexing all available items, in some cases you may want to index only new available items or update the ones that changed since last index operation. You can choose among the following index operations:

- **FULL**: recreates the index, all items will be indexed based on a FULL index query;
- **UPDATE**: updates some documents in the index. Items to be indexed/updated are typically selected by an UPDATE query;
- **PARTIAL_UPDATE**: similar to the UPDATE operation but allows to select the fields to be updated (for faster updates);
- **DELETE**: deletes documents from the index. Items to be deleted are typically selected by a DELETE query.

i Note

It is recommended to use the FULL indexing operation regularly to make sure the data, which has already been removed from the database, is also removed from solr .

Indexing Mode for the FULL Operation

When performing a FULL operation, the following indexing modes are supported:

- **DIRECT**: indexing occurs directly on the live index. New and updated documents are marked and old ones are removed after a successful operation;
- **TWO_PHASE**: indexing occurs on a temporary index. After a successful operation, the temporary index replaces the current live one.

Limitations of the PARTIAL_UPDATE operation

A PARTIAL_UPDATE is potentially much faster than a normal UPDATE, however it also features the following limitations:

For instructions on performing partial updates see: [Update Solr Index with CronJob](#).

- requires `stored=true` on all attributes on the Solr `schema.xml` file (for example, currently `fulltext_<lang>` fields have `stored=false` by default and this value should be changed to `true`);
- should not be used on attributes that are used for spellchecking / suggestions;
- refreshing the index continuously may have side-effects on search performance.

Triggering the Indexing Process

Using the IndexerService

The IndexerService service provides methods that allow you to perform all the index operations mentioned before. It is important to mention that:

- If a method allows to pass the items to be indexed as parameter, it means that the default index query for the operation will not be used;
- The only methods that receive the indexed properties as parameter, are the ones used for the PARTIAL_UPDATE operation.

Using a Cron Job

Indexing can also be triggered by a CronJob. The following Cron Job types are available:

- `SolrIndexerCronJob` (combined with the `solrIndexerJob` Job):
 - supports the FULL, UPDATE and DELETE index operations;
 - does not support the PARTIAL_UPDATE index operation;
 - uses the default indexer queries;
 - the session user will be the one configured in the indexer query;
 - uses the data from a read-only secondary data source if you configure it that way in Backoffice
- `SolrExtIndexerCronJob` (combined with the `solrExtIndexerJob` Job):
 - supports the UPDATE, PARTIAL_UPDATE and DELETE index operations;
 - does not support the FULL index operation;
 - does not update the last index time;
 - the session user will be the one configured in the Cron Job;
 - has additional attributes:
 - `indexedType`: the indexed type to use, for example, "Product" or "Product_indexName";
 - `indexedProperties`: the properties to index, only used if the index operation is PARTIAL_UPDATE;
 - `query`: the flexible search query that selects the items to be indexed;
 - `queryParameterProvider`: id of the spring bean that allows to create parameters to be passed to the query. It should implement one of the following types: `ParameterProvider`, `ContextAwareParameterProvider` or `CronJobAwareParameterProvider`, all from the `de.hybris.platform.solrfacetsearch.provider` package.

Handling Suspend and Resume

The indexing mechanism supports suspending and resuming of the platform. When the indexing is running and the platform is for some reason suspended, the indexing process will be gracefully suspended and then resumed after the start of the platform. The batches for which the indexing started before suspension will be fully indexed before the platform is suspended.

This feature is supported by:

- default indexing
- distributed indexing

Selecting any of the indexing type causes the same behavior of the suspend and resume functionality.

For further details on suspend and resume, see [Suspending and Resuming SAP Commerce](#).

Error Handling

Regarding error handling during the indexing process, the following applies since hybris version 5.5.0:

- if an exception is detected, the indexing process is cancelled;
- in this case, it cancels workers that have not started execution, tries also to cancel workers currently running;
- only successfully fully indexed items will be seen by searches.

i Note

By setting the `ignoreErrors` flag to true in the indexer configuration, some errors are ignored during the indexing process.

Indexing API

This document provides information on the top level APIs for indexing functionality and how to perform some simple customizations.

Indexer Service

Triggering the indexing process is easy: you only need to call one of the methods of the indexer service, as shown in the example below.

```
public interface IndexerService
{
    /**
     * Performs a full index operation which recreates the index. All the types associated with the facet search
     * configuration are considered. Items to be indexed are selected based on the full index query.
     *
     * @param facetSearchConfig
     *      - configuration for indexer instance
     *
     * @throws IndexerException
     *      exception is thrown when full index query is missing in configuration or an unexpected error occurs
     *      during indexing.
     */
    void performFullIndex(FacetSearchConfig facetSearchConfig) throws IndexerException;

    /**
     * Same as {@link #performFullIndex(FacetSearchConfig)} but allows to pass the indexer hints as parameter.
     *
     * @param facetSearchConfig
     *      - configuration for indexer instance
     * @param indexerHints
     *      - the indexer hints
     *
     * @throws IndexerException
     *      exception is thrown when full index query is missing in configuration or an unexpected error occurs
     *      during indexing.
     */
    void performFullIndex(FacetSearchConfig facetSearchConfig, Map<String, String> indexerHints) throws IndexerException;

    /**
     * Updates some items on the index. All the types associated with the facet search configuration are considered.
     * Items to be updated are selected based on the update index query.
     *
     * @param facetSearchConfig
     *      - configuration for indexer instance
     *
     * @throws IndexerException
     *      exception is thrown when update index query is missing in configuration or an unexpected error occurs
     *      during indexing.
     */
    void updateIndex(FacetSearchConfig facetSearchConfig) throws IndexerException;

    /**
     * Same as {@link #updateIndex(FacetSearchConfig)} but allows to pass the indexer hints as parameter.
     *
     * @param facetSearchConfig
     *      - configuration for indexer instance
     * @param indexerHints
     *      - the indexer hints
     *
     * @throws IndexerException
     *      exception is thrown when update index query is missing in configuration or an unexpected error occurs
     *      during indexing.
     */
    void updateIndex(FacetSearchConfig facetSearchConfig, Map<String, String> indexerHints) throws IndexerException;

    /**
     * Updates some items on the index for a specific type. Items to be updated are selected based on the update
     * query.
     *
     * @param facetSearchConfig
     *      - configuration for indexer instance
     * @param indexedType
     *      - selected type
     *
     * @throws IndexerException
     *      exception is thrown when update index query is missing in configuration or an unexpected error occurs
     *      during indexing.
     */
    void updateTypeIndex(FacetSearchConfig facetSearchConfig, IndexedType indexedType) throws IndexerException;
}
```

```

/**
 * Same as {@link #updateTypeIndex(FacetSearchConfig, IndexedType)} but allows to pass the indexer hints as
 * parameter.
 *
 * @param facetSearchConfig
 *         - configuration for indexer instance
 * @param indexedType
 *         - selected type
 * @param indexerHints
 *         - the indexer hints
 *
 * @throws IndexerException
 *         exception is thrown when update index query is missing in configuration or an unexpected error
 *         during indexing.
 */
void updateTypeIndex(FacetSearchConfig facetSearchConfig, IndexedType indexedType, Map<String, String> indexerHints)
    throws IndexerException;

/**
 * Updates some items on the index for a specific type.
 *
 * @param facetSearchConfig
 *         - configuration for indexer instance
 * @param indexedType
 *         - selected type
 * @param pks
 *         - pks of items to be updated
 *
 * @throws IndexerException
 *         exception is thrown when an unexpected error occurs during indexing.
 */
void updateTypeIndex(final FacetSearchConfig facetSearchConfig, final IndexedType indexedType, List<PK> pks)
    throws IndexerException;

/**
 * Same as {@link #updateTypeIndex(FacetSearchConfig, IndexedType, List)} but allows to pass the indexer hints
 * parameter.
 *
 * @param facetSearchConfig
 *         - configuration for indexer instance
 * @param indexedType
 *         - selected type
 * @param pks
 *         - pks of items to be updated
 * @param indexerHints
 *         - the indexer hints
 *
 * @throws IndexerException
 *         exception is thrown when an unexpected error occurs during indexing.
 */
void updateTypeIndex(final FacetSearchConfig facetSearchConfig, final IndexedType indexedType, List<PK> pks,
    Map<String, String> indexerHints) throws IndexerException;

/**
 * Updates some properties of some items on the index for a specific type.
 *
 * @param facetSearchConfig
 *         - configuration for indexer instance
 * @param indexedType
 *         - selected type
 * @param indexedProperties
 *         - properties to update
 * @param pks
 *         - pks of items to be updated
 *
 * @throws IndexerException
 *         exception is thrown when an unexpected error occurs during indexing.
 */
void updatePartialTypeIndex(final FacetSearchConfig facetSearchConfig, final IndexedType indexedType,
    final Collection<IndexedProperty> indexedProperties, List<PK> pks) throws IndexerException;

/**
 * Same as {@link #updatePartialTypeIndex(FacetSearchConfig, IndexedType, Collection, List)} but allows to pass
 * indexer hints as parameter.
 *
 * @param facetSearchConfig
 *         - configuration for indexer instance
 * @param indexedType
 *         - selected type
 * @param indexedProperties
 *         - properties to update
 * @param pks
 *         - pks of items to be updated
 * @param indexerHints
 *         - the indexer hints

```



```

*
* @throws IndexerException
*         exception is thrown when an unexpected error occurs during indexing.
*/
void updatePartialTypeIndex(final FacetSearchConfig facetSearchConfig, final IndexedType indexedType,
    final Collection<IndexedProperty> indexedProperties, List<PK> pks, Map<String, String> indexerHints)
    throws IndexerException;

/**
 * Removes some items from the index. All the types associated with the facet search configuration are consid
 * Items to be removed are selected based on the delete index query.
 *
 * @param facetSearchConfig
 *         - configuration for indexer instance
 *
 * @throws IndexerException
 *         exception is thrown when delete index query is missing in configuration or an unexpected error
 *         during indexing.
 */
void deleteFromIndex(FacetSearchConfig facetSearchConfig) throws IndexerException;

/**
 * Same as {@link #deleteFromIndex(FacetSearchConfig)} but allows to pass the indexer hints as parameter.
 *
 * @param facetSearchConfig
 *         - configuration for indexer instance
 * @param indexerHints
 *         - the indexer hints
 *
 * @throws IndexerException
 *         exception is thrown when delete index query is missing in configuration or an unexpected error
 *         during indexing.
 */
void deleteFromIndex(FacetSearchConfig facetSearchConfig, Map<String, String> indexerHints) throws IndexerExc

/**
 * Removes some items from the index for a specific type. Items to be removed are selected based on the delet
 * query.
 *
 * @param facetSearchConfig
 *         - configuration for indexer instance
 * @param indexedType
 *         - selected type
 *
 * @throws IndexerException
 *         exception is thrown when delete index query is missing in configuration or an unexpected error
 *         during indexing.
 */
void deleteTypeIndex(FacetSearchConfig facetSearchConfig, IndexedType indexedType) throws IndexerException;

/**
 * Same as {@link #deleteTypeIndex(FacetSearchConfig, IndexedType)} but allows to pass the indexer hints as
 * parameter.
 *
 * @param facetSearchConfig
 *         - configuration for indexer instance
 * @param indexedType
 *         - selected type
 * @param indexerHints
 *         - the indexer hints
 *
 * @throws IndexerException
 *         exception is thrown when delete index query is missing in configuration or an unexpected error
 *         during indexing.
 */
void deleteTypeIndex(FacetSearchConfig facetSearchConfig, IndexedType indexedType, Map<String, String> indexe
    throws IndexerException;

/**
 * Removes some items from the index for a specific type.
 *
 * @param facetSearchConfig
 *         - configuration for indexer instance
 * @param indexedType
 *         - selected type
 * @param pks
 *         - pks of items to be removed
 *
 * @throws IndexerException
 *         exception is thrown when an unexpected error occurs during indexing.
 */
void deleteTypeIndex(FacetSearchConfig facetSearchConfig, IndexedType indexedType, List<PK> pks) throws Inde

/**
 * Same as {@link #deleteTypeIndex(FacetSearchConfig, IndexedType, List)} but allows to pass the indexer hint

```

```

    * parameter.
    *
    * @param facetSearchConfig
    *         - configuration for indexer instance
    * @param indexedType
    *         - selected type
    * @param pks
    *         - pks of items to be removed
    * @param indexerHints
    *         - the indexer hints
    *
    * @throws IndexerException
    *         exception is thrown when an unexpected error occurs during indexing.
    */
    void deleteTypeIndex(FacetSearchConfig facetSearchConfig, IndexedType indexedType, List<PK> pks,
        Map<String, String> indexerHints) throws IndexerException;

    ...
}

```

Some of the most common uses are:

- Performing a full index operation:

```
indexerService.performFullIndex(facetSearchConfig);
```

- Updating some documents in the index:

```
indexerService.updateTypeIndex(facetSearchConfig, indexedType, pks);
```

- Deleting some documents from the index:

```
indexerService.deleteTypeIndex(facetSearchConfig, indexedType, pks);
```

Context and Listeners

Context

Some context objects can be accessed during the indexing process. You can access them by using the corresponding factory, or, in some cases, as a parameter of the method (e.g. listeners).

The following context types are available:

- `IndexerQueryContext` (together with `IndexerQueryContextFactory`): Represents a context valid for the duration of an indexer query;
- `IndexerContext` (together with `IndexerContextFactory`): Represents a context valid for the duration of an indexer index operation, however it is not valid inside an indexer batch (an indexer batch may run on a different thread or even on a different machine);
- `IndexerBatchContext` (together with `IndexerBatchContextFactory`): Represents a context valid for the duration of an indexer batch. Each batch runs on a separate thread and this context is only valid for the corresponding thread.

Listeners

This section describes the listener types that you can implement to intercept the indexing process.

IndexerQueryListener

You use the `IndexerQueryListener` to intercept indexer queries execution. The following code excerpt shows the `IndexerQueryListener` interface:

```

/**
 * Interface for receiving notifications about indexer queries execution.
 */
public interface IndexerQueryListener
{
    /**
     * Handles a notification that an indexer query is about to begin execution.
    */
}

```

```

*
* @param queryContext
*       - the {@link IndexerQueryContext}
*
* @throws IndexerException
*       if an error occurs
*/
void beforeQuery(IndexerQueryContext queryContext) throws IndexerException;

/**
 * Handles a notification that an indexer query has just completed.
 *
 * @param queryContext
 *       - the {@link IndexerQueryContext}
 *
 * @throws IndexerException
 *       if an error occurs
 */
void afterQuery(IndexerQueryContext queryContext) throws IndexerException;

/**
 * Handles a notification that an indexer query failed (this may also be due to listeners failing).
 *
 * @param queryContext
 *       - the {@link IndexerQueryContext}
 *
 * @throws IndexerException
 *       if an error occurs
 */
void afterQueryError(IndexerQueryContext queryContext) throws IndexerException;
}

```

IndexerListener

You use the `IndexerListener` to intercept execution of an indexer operation. The following code excerpt shows the `IndexerListener` interface:

```

/**
 * Interface for receiving notifications about {@link IndexerContext} instances.
 */
public interface IndexerListener
{
    /**
     * Handles a notification that the indexing for a particular {@link IndexerContext} is about to begin.
     *
     * @param context
     *       - the {@link IndexerContext}
     *
     * @throws IndexerException
     *       if an error occurs
     */
    void beforeIndex(IndexerContext context) throws IndexerException;

    /**
     * Handles a notification that the indexing for a particular {@link IndexerContext} has just been completed.
     *
     * @param context
     *       - the {@link IndexerContext}
     *
     * @throws IndexerException
     *       if an error occurs
     */
    void afterIndex(IndexerContext context) throws IndexerException;

    /**
     * Handles a notification that the indexing for a particular {@link IndexerContext} failed.
     *
     * @param context
     *       - the {@link IndexerContext}
     *
     * @throws IndexerException
     *       if an error occurs
     */
    void afterIndexError(IndexerContext context) throws IndexerException;
}

```

IndexerBatchListener

You can use the IndexerBatchListener to intercept the execution of an indexer operation for a particular batch. The following code excerpt shows the IndexerBatchListener interface:

```
/**
 * Interface for receiving notifications about {@link IndexerBatchContext} instances.
 */
public interface IndexerBatchListener
{
    /**
     * Handles a notification that the processing for a particular {@link IndexerBatchContext} is about to begin.
     *
     * @param batchContext
     *        - the {@link IndexerBatchContext}
     *
     * @throws IndexerException
     *        if an error occurs
     */
    void beforeBatch(IndexerBatchContext batchContext) throws IndexerException;

    /**
     * Handles a notification that the processing for a particular {@link IndexerBatchContext} has just been completed.
     *
     * @param batchContext
     *        - the {@link IndexerBatchContext}
     *
     * @throws IndexerException
     *        if an error occurs
     */
    void afterBatch(IndexerBatchContext batchContext) throws IndexerException;

    /**
     * Handles a notification that the processing for a particular {@link IndexerBatchContext} failed.
     *
     * @param batchContext
     *        - the {@link IndexerBatchContext}
     *
     * @throws IndexerException
     *        if an error occurs
     */
    void afterBatchError(IndexerBatchContext batchContext) throws IndexerException;
}
```

Indexer Hints

Indexer Hints are intended to provide additional information to the indexing process. The tips can be accessed at any time; they are part of the indexer context and indexer batch context.

The following hints are supported:

Hint	Description	Possible Values	Since
commitMode	Overrides the commit mode defined in the indexer configuration.	MIXED, NEVER, AFTER_INDEX, AFTER_BATCH	5.6
optimizeMode	Overrides the optimize mode defined in the indexer configuration.	NEVER, AFTER_INDEX, AFTER_FULL_INDEX	5.6

Value Provider API

Learn how you can write a value provider using the available API.

The API has the following advantages:

- It groups the indexed properties that use the same value provider;
- It provides more context information, for example, it is possible to access all the items that are being indexed for a particular batch.

Creating a Custom Provider

Create a custom value provider by using the `QualifierProvider` or implementing the `ValueResolver` interface.

Extending `AbstractValueResolver` and Using a `QualifierProvider`

i Note

The following procedure shows the recommended way of creating a value provider.

It follows patterns that were observed and avoids code duplication in your value providers. You can implement the following methods:

- **addFieldValues:** Mandatory method. It is responsible for resolving the values to be indexed. The data and qualifier data are available as part of the `resolverContext` parameter.
- **loadData:** Optional method. By default it returns `null`. It loads data that is valid in the context of a model.
- **loadQualifierData:** Optional method. By default, it returns `null`. It loads data that is valid in the context of a model and qualifier. For example, a qualifier can be a specific language or currency.

The following code sample shows the `AbstractValueResolver` class:

```
/**
 * Abstract class for value resolvers that want to use a {@link QualifierProvider}
 *
 * @param <T>
 *         the type of the model
 * @param <MDATA>
 *         the type of the data that is valid in the context of a model
 * @param <QDATA>
 *         the type of the data that is valid in the context of a model and qualifier
 */
public abstract class AbstractValueResolver<T extends ItemModel, MDATA, QDATA> implements ValueResolver<T>
{
    ...

    /**
     * Loads data that is valid in the context of a model.
     *
     * @param batchContext
     *        - the current indexer batch context
     * @param indexedProperties
     *        - the indexed properties that use the same value resolver
     * @param model
     *        - the values should be resolved for this model instance
     *
     * @throws FieldValueProviderException
     *        if an error occurs
     */
    protected MDATA loadData(final IndexerBatchContext batchContext, final Collection<IndexedProperty> indexedPr
        final T model) throws FieldValueProviderException
    {
        return null;
    }

    /**
     * Loads data that is valid in the context of a model and qualifier.
     *
     * @param batchContext
     *        - the current indexer batch context
     * @param indexedProperties
     *        - the indexed properties that use the same value resolver
     * @param model
     *        - the values should be resolved for this model instance
     *
     * @throws FieldValueProviderException
     */
}
```

```

        *           if an error occurs
        */
protected QDATA loadQualifierData(final IndexerBatchContext batchContext, final Collection<IndexedProperty> indexedProperties,
        final T model, final Qualifier qualifier) throws FieldValueProviderException
{
    return null;
}

protected abstract void addFieldValues(final InputDocument document, final IndexerBatchContext batchContext,
        final IndexedProperty indexedProperty, final T model, ValueResolverContext<MDATA, QDATA> resolverContext) throws FieldValueProviderException;
...
}

```

The following code sample shows the QualifierProvider interface:

```

/**
 * This interface provides support for different types of qualifiers.
 */
public interface QualifierProvider
{
    /**
     * Returns all the supported types by this provider.
     *
     * @return the supported types
     */
    Set<Class<?>> getSupportedTypes();

    /**
     * Checks if qualifiers can be applied/used with the indexed property passed as parameter.
     *
     * @param indexedProperty
     *        - the indexed property
     *
     * @return {@code true} if qualifiers can be used, {@code false} otherwise
     */
    boolean canApply(IndexedProperty indexedProperty);

    /**
     * Returns all the possible qualifiers for a given index configuration and indexed type.
     *
     * @param facetSearchConfig
     *        - the facet search configuration
     * @param indexedType
     *        - the indexed type
     *
     * @return the available qualifiers
     */
    Collection<Qualifier> getAvailableQualifiers(FacetSearchConfig facetSearchConfig, IndexedType indexedType);

    /**
     * Applies the qualifier passed as parameter. This normally consists in setting some attributes on the session
     * by calling a service to set the current session language, currency, etc.
     *
     * @param qualifier
     *        - the {@link Qualifier} to be applied
     */
    void applyQualifier(Qualifier qualifier);

    /**
     * Returns the current qualifier. This normally consists in getting some attributes from the session and creating
     */
}

```

```

    * corresponding qualifier.
    *
    * @return the current qualifier
    */
    Qualifier getCurrentQualifier();
}

```

The following shows a Java code sample for `ProductUrlsValueResolver` class:

```

public class ProductUrlsValueResolver extends AbstractValueResolver<ProductModel, Object, Object>
{
    private UrlResolver<ProductModel> urlResolver;

    public UrlResolver<ProductModel> getUrlResolver()
    {
        return urlResolver;
    }

    @Required
    public void setUrlResolver(final UrlResolver<ProductModel> urlResolver)
    {
        this.urlResolver = urlResolver;
    }

    @Override
    protected void addFieldValues(final InputDocument document, final IndexerBatchContext batchContext,
        final IndexedProperty indexedProperty, final ProductModel product,
        final ValueResolverContext<Object, Object> resolverContext) throws FieldValueProviderException
    {
        final String productUrl = urlResolver.resolve(product);
        if (!StringUtils.isBlank(productUrl))
        {
            document.addField(indexedProperty, productUrl, resolverContext.getFieldQualifier());
        }
    }
}

```

The following is the Spring configuration:

```

<bean id="productUrlsValueResolver" class="com.example.ProductUrlsValueResolver" parent="abstractValueResolver">
    <property name="qualifierProvider" ref="languageQualifierProvider" />
    <property name="urlResolver" ref="productModelUrlResolver" />
</bean>

```

Implementing the `ValueResolver` Interface

i Note

This is one of the possible ways to create a value provider, however not the recommended one.

The following code excerpt shows the `ValueResolver` interface:.

```

/**
 * <p>
 * Implementations of this interface are responsible for resolving the values to be indexed. This interface works
 * indexed property level.
 * </p>
 *
 * @see TypeValueResolver
 */
public interface ValueResolver<T extends ItemModel>
{
    /**

```

```

    * Resolves the values to be indexed. The indexed properties that use the same value resolver are grouped and
    * method is called once for each one of these groups.
    *
    * @param document
    *       - document that will be indexed, all the resolved values should be added as fields to this document
    * @param batchContext
    *       - the current indexer batch context
    * @param indexedProperties
    *       - the indexed properties that use the same value resolver
    * @param model
    *       - the values should be resolved for this model instance
    *
    * @throws FieldValueProviderException
    *       if an error occurs
    */
    void resolve(final InputDocument document, final IndexerBatchContext batchContext,
                final Collection<IndexedProperty> indexedProperties, T model) throws FieldValueProviderException;
}

```

The following example shows the Java code:

```

public class ProductUrlsValueResolver implements ValueResolver<ProductModel>
{
    private CommonI18NService commonI18NService;
    private UrlResolver<ProductModel> urlResolver;

    public CommonI18NService getCommonI18NService()
    {
        return commonI18NService;
    }

    @Required
    public void setCommonI18NService(final CommonI18NService commonI18NService)
    {
        this.commonI18NService = commonI18NService;
    }

    public UrlResolver<ProductModel> getUrlResolver()
    {
        return urlResolver;
    }

    @Required
    public void setUrlResolver(final UrlResolver<ProductModel> urlResolver)
    {
        this.urlResolver = urlResolver;
    }

    @Override
    public void resolve(final InputDocument document, final IndexerBatchContext batchContext,
                        final Collection<IndexedProperty> indexedProperties, final ProductModel model) throws FieldValueProviderException
    {
        // index non localized properties
        for (final IndexedProperty indexedProperty : indexedProperties)
        {
            if (!indexedProperty.isLocalized())
            {
                addFieldValue(document, indexedProperty, model, null);
            }
        }

        // index localized properties
        final LanguageModel currentLanguage = commonI18NService.getCurrentLanguage();
        try
        {
            final FacetSearchConfig facetSearchConfig = batchContext.getFacetSearchConfig();
            final IndexConfig indexConfig = facetSearchConfig.getIndexConfig();
            final Collection<LanguageModel> languages = indexConfig.getLanguages();

            for (final LanguageModel language : languages)
            {
                commonI18NService.setCurrentLanguage(language);

                for (final IndexedProperty indexedProperty : indexedProperties)
                {
                    if (indexedProperty.isLocalized())
                    {
                        addFieldValue(document, indexedProperty, model, language);
                    }
                }
            }
        }
        finally
    }
}

```



```

    {
        commonI18NService.setCurrentLanguage(currentLanguage);
    }
}

protected void addFieldValue(final InputDocument document, final IndexedProperty indexedProperty, final ProductModel product,
    final LanguageModel language) throws FieldValueProviderException
{
    final String productUrl = urlResolver.resolve(product);
    if (!StringUtils.isBlank(productUrl))
    {
        document.addField(indexedProperty, productUrl, language == null ? null : language.getIsocode());
    }
}
}

```

The following example shows the Spring configuration:

```

<bean id="productUrlsValueResolver" class="com.example.ProductUrlsValueResolver">
    <property name="commonI18NService" ref="commonI18NService" />
    <property name="urlResolver" ref="productModelUrlResolver" />
</bean>

```

Using Parameters

Another improvement is the possibility of having multiple parameters (using the `IndexedProperty`). The following example shows how you can take advantage of this.

```

public class ProductUrlsValueResolver extends AbstractValueResolver<ProductModel, Object, Object>
{
    public static final String OPTIONAL_PARAM = "optional";
    public static final boolean OPTIONAL_PARAM_DEFAULT_VALUE = true;

    private UrlResolver<ProductModel> urlResolver;

    public UrlResolver<ProductModel> getUrlResolver()
    {
        return urlResolver;
    }

    @Required
    public void setUrlResolver(final UrlResolver<ProductModel> urlResolver)
    {
        this.urlResolver = urlResolver;
    }

    @Override
    protected void addFieldValues(final InputDocument document, final IndexerBatchContext batchContext,
        final IndexedProperty indexedProperty, final ProductModel product,
        final ValueResolverContext<Object, Object> resolverContext) throws FieldValueProviderException
    {
        boolean hasValue = false;

        final String productUrl = urlResolver.resolve(product);
        if (!StringUtils.isBlank(productUrl))
        {
            hasValue = true;
            document.addField(indexedProperty, productUrl, resolverContext.getFieldQualifier());
        }

        if (!hasValue)
        {
            final boolean isOptional = ValueProviderParameterUtils.getBoolean(indexedProperty, OPTIONAL_PARAM,
                OPTIONAL_PARAM_DEFAULT_VALUE);
            if (!isOptional)
            {
                throw new FieldValueProviderException("No value resolved for indexed property " + indexedProperty);
            }
        }
    }
}

```

The following code is the Spring configuration:

```

<bean id="productUrlsValueResolver" class="com.example.ProductUrlsValueResolver" parent="abstractValueResolver">
    <property name="qualifierProvider" ref="languageQualifierProvider" />
    <property name="urlResolver" ref="productModelUrlResolver" />
</bean>

```

Available Resolvers and Providers

ModelAttributeValueResolver

The `ModelAttributeValueResolver` gets the values from the attributes in the model. By default, if a parameter attribute is not specified, it tries to get the attribute with the same name as the one configured on the indexed property.

Parameter	Default value	Description
<code>optional</code>	<code>true</code>	If false, indicates that the resolved values should not be null and not an empty string (for every qualifier). If these conditions are not met, an exception of type <code>FieldValueProviderException</code> is thrown.
<code>attribute</code>		If specified, this is the name of the attribute.
<code>split</code>	<code>false</code>	If true, splits any resolved value around matches of a regular expression (only if the value is of type <code>String</code>).
<code>splitRegex</code>	<code>\s+</code>	If split is true this is the regular expression to use.
<code>format</code>		The ID of the <code>Format Bean</code> that is going to be used to format the attribute value object before applying the split
<code>evaluateExpression</code>	<code>false</code>	If true the attribute name is assumed to be a spring expression language that need to be evaluated

Available Qualifier Providers

The following qualifier providers are available:

- `NoOpQualifierProvider`: A dummy qualifier provider for the cases where no qualifiers are required.
- `LanguageQualifierProvider`: A qualifier provider for languages.
- `CurrencyQualifierProvider`: A qualifier provider for currencies.

Document Identifiers

Every indexed document needs to have a unique identifier. The default one should work in most cases, however you may also define a custom one.

Creating an Identity Provider

In order to create your custom identity provider, you need to first implement the `IdentityProvider` interface which provides a unique string representation of indexed type identifier that is generated with respect to the model type restrictions:

```
public interface IdentityProvider<T>
{
    /**
     * Provides unique string representation of indexed type identifier that is generated with respect to the model
     * restrictions.
     *
     * @param indexConfig
     * @param model
     * @return identifier
     */
    String getIdentifier(IndexConfig indexConfig, T model);
}
```

The `ProductIdentityProvider` class, which resolves a unique identity of a provider, is an example of implementing the interface:

```
public class ProductIdentityProvider implements IdentityProvider<ProductModel>
{
    @Override
    public String getIdentifier(final IndexConfig indexConfig, final ProductModel product)
    {
        final CatalogVersionModel catalogVersion = product.getCatalogVersion();
        final String code = product.getCode();
        return catalogVersion.getCatalog().getId() + "/" + catalogVersion.getVersion() + "/" + code;
    }
}
```

You need to remember to create a bean in the spring context for your identity provider:

```
<bean id="productIdentityProvider" class="com.example.ProductIdentityProvider"/>
```

Registering an Identity Provider

A custom identity provider should be configured in the indexed type.

Grouping on a Sharded Environment

Using distributed result grouping in an sharded SolrCloud environment has some caveats. To make it work properly, all the documents in each group should be co-located on the same shard and this can be achieved by indexing using a `compositeId` router. To use a `compositeId` router, the document's unique identifier should be prefixed by an attribute's (on which we want to group on) value followed by a character "!" and the unique document id.

`ShardAwareProductIdentityProvider`, which is the default implementation used for sharded environment, allows grouping basing on the product by creating a `compositeId` with the prefix, which is a base product id.

Related Information

[Distributed Result Grouping Caveats](#) ➡

[Shards and Indexing Data in SolrCloud](#) ➡

[Introduction to Solr Indexing](#) ➡

Search Functionality

To fully make use of the search functionality, get to know the strategies used in the search process, and learn how to customize your search by using Search API, sort providers and custom mapping.

[Search Process](#)

General information about the search process and various search strategies.

[Search API](#)

Learn more about the top level APIs for the search functionality to perform some simple customizations.

[Customized Sort Providers](#)

This document provides information about customizing sort providers for the Solr facet search.

[Direct Access to the Solr Query](#)

This document provides information on the direct access to the solr queries.

[Custom Mapping Search Results](#)

This topic describes how to supply custom data transfer objects and result converters.

Search Process

General information about the search process and various search strategies.

The different search strategies were implemented in order to have a more advanced search functionality but at the same time keep the compatibility with previous versions. Depending on your needs, you can choose to use one or the other. You can find more details about them in the next sections.

Creating heavier search queries with a large number of facets and search fields might cause Solr resource issues. You must:

- Define a new facet and search field only when it's necessary.
- Review your settings regularly and purge the facets that not required anymore.
- Run performance tests to validate if your current infrastructure can still sustain your expected user load (particularly after any new changes).

i Note

Search queries and search results are not affected by FlexibleSearch restrictions. For more information, see [Restrictions](#).

LegacyFacetSearchStrategy

This strategy is used when legacy mode is enabled in the search configuration. It exists for the compatibility purposes and, when used, search functionality behaves in a similar way as in previous versions.

The main limitation is that many of the query conditions and parameters come from the configuration and not from the search query object. This makes it difficult to have queries that behave differently depending on the context. For example, it is difficult to specify the facets that should be included in the response in a dynamic way.

DefaultFacetSearchStrategy

This strategy will be used when the legacy mode is disabled in the search configuration. Some of the new functionality is available only when using this strategy.

The main advantage is that almost every query condition and parameter can be set directly in the search query object. This makes it more suitable for cases where you want queries that depend on the context. Due to its simplicity it also makes it easier to create new functionality on top of it. The following functionality is only available when using this strategy:

- default query operator (for a specific query);
- filter queries;
- free text search queries;
- facets (not facet values);
- fields to be included in the response;
- boosts.

i Note

The `SolrQueryPostProcessor` and `SolrResultPostProcessor` interfaces are not supported by this strategy. The `FacetSearchListener` interface can be used for similar functionality.

Free Text Search

One of new features of this strategy is the support for free text search queries directly in the search query object. As there are different ways of performing the free text query, we created the `FreeTextQueryBuilder` interface. Implementations of this interface are responsible for converting the representation in the search query to something that Solr/Lucene can understand.

The following free text query builders are available:

- `MultiFieldFreeTextQueryBuilder`: builds the query in a way that the final score will be the sum of the scores of all the subqueries;
- `DisMaxFreeTextQueryBuilder`: similar to `MultiFieldFreeTextQueryBuilder` but groups some of the subqueries. The score for the group will be the maximum score of the subqueries that belong to that group (and not the sum). This query builder supports the following parameters:

Key	Default Value	Possible Values	Description
groupByQueryType	true	true,false	Changes the way of grouping disjunction max queries. If set to true, it also groups queries by type (where the types are: free text query, free text fuzzy query, free text wildcard query).
tie	0.0	0.0-1.0	<p>Allows you to you configure how much the final score of the query will be influenced by the scores of the lower scoring fields compared to the highest scoring field:</p> <ul style="list-style-type: none">◦ a value of "0.0" makes the query a pure "disjunction max query";◦ a value of "1.0" makes the query a pure "disjunction sum query" where it doesn't matter what the maximum scoring sub query is. <p>Tie equation: (score of matching clause with the highest score) + ((tie parameter) * (scores of any other matching clauses))</p>

Filter Query

The filter query functionality is a part of search and navigation module and it is used to facilitate filtering the search query content.

The aim was to replace using hidden facets to filter the results. Consequently, the SolrSearchFilterQueryData list was introduced as part of SolrSearchQueryData to contain a list of the required data to be filtered.

```
<!-- FilterQueryOperator -->
<enum class="de.hybris.platform.commerceservices.search.solrfacetsearch.data.FilterQueryOperator">
  <value>AND</value>
  <value>OR</value>
</enum>

<bean class="de.hybris.platform.commerceservices.search.solrfacetsearch.data.SolrSearchFilterQueryData">
  <property name="key" type="String"/>
  <property name="operator" type="de.hybris.platform.commerceservices.search.solrfacetsearch.data.FilterQueryOperator"/>
  <property name="values" type="java.util.Set<String>"/>
</bean>
```

The SearchFiltersPopulator, which populates the data for SearchQuery, has been adjusted to populate indexed properties as filter queries, in case they are not of type Facet with OR as the operator, otherwise they will be handled as ordinary facets.

The populator also collects the lists of SolrSearchQueryData and adds them as filter queries with AND as the default operator, in case an operator does not exist.

An example of filter query solution is the implementation in the commerceservices extension and used for example in the Bundling Module to search for the products available for a given component.

Grouping

Result grouping puts documents with a common field value into groups, and returns the top documents for each group. There are two implementations that you can use:

Method	Priority	More Information
Collapse and Expand	Default	https://solr.apache.org/guide/8_9/collapse-and-expand-results.html
Group	Optional	https://solr.apache.org/guide/8_9/result-grouping.html

Collapse and expand is the default method, and has better performance. However, when using the collapse and expand implementation, highlighting only works for the top-level documents. To use the previous grouping implementation, set the following property in your project configuration:

```
solrfacetsearch.search.grouping.method=group
```

The default method is set as follows:

```
solrfacetsearch.search.grouping.method=collapse-expand
```

Search API

Learn more about the top level APIs for the search functionality to perform some simple customizations.

In order to execute search you need to call one of the methods provided by the `FacetSearchService` interface. Creating and populating the search query as well as processing the results might seem to be more complex, that's why we have provided some examples of using the interface methods.

Creating a Search Query

Start with creating a Search Query object. To do so you can use the `SearchQuery` constructor to create an empty search query:

```
SearchQuery searchQuery = new SearchQuery(facetSearchConfig, indexedType);
```

You can also use the `FacetSearchQuery` interface methods:

- `createSearchQuery` method to create an empty search query:

```
SearchQuery searchQuery = facetSearchService.createSearchQuery(facetSearchConfig, indexedType);
```

- `createPopulatedSearchQuery` method to create a populated search query. Details about grouping, facets and fields are taken from the search configuration.

```
SearchQuery searchQuery = facetSearchService.createPopulatedSearchQuery(facetSearchConfig, indexedType);
```

- `createFreeTextSearchQuery` method to create a populated search query. Details about grouping, facets, fields and free text search are taken from the search configuration.

```
SearchQuery searchQuery = facetSearchService.createFreeTextSearchQuery(facetSearchConfig, indexedType, user
```

- `createSearchQueryFromTemplate` method to create a populated search query from a template. Details about grouping, facets and fields are taken from the template.

```
SearchQuery searchQuery = createSearchQueryFromTemplate(facetSearchConfig, indexedType, queryTemplateName)
```

- `createSearchQueryFromTemplate` method to create a populated search query from a template. Details about grouping, facets, fields and free text search are taken from the template.

```
SearchQuery searchQuery = createFreeTextSearchQueryFromTemplate(facetSearchConfig, indexedType, queryTempla
```

Populating the Search Query

In the previous section we gave some examples of how to create the search query object. In some of the cases, the query was not only created, but also populated with some query parameters. That might not be enough and you might want to add some additional query parameters. Some of the most common use cases are:

- Adding search conditions:

```
searchQuery.addQuery(field, fieldValue);
```

- Adding search conditions that do not affect scoring:

```
searchQuery.addFilterQuery(field, fieldValue);
```

- Adding facets and facet values:

```
searchQuery.addFacet(field);
searchQuery.addFacetValue(field, fieldValue);
```

- Sorting the results:

```
searchQuery.addSort(field);
```

i Note

Some of the query parameters in the search query object will only be used together with the new search strategy. When using the legacy search strategy, that information will be taken directly from the configuration and not from the search query object. The affected parameters are:

- Default query operator
- Filter queries
- Free text search queries
- Facets (not facet values)
- Fields to be included in the response
- Boosts

Executing the Search Query

Once the search query object is created and populated, we need to execute it in order to obtain the result. For this we just need to call one of the search methods from the `FacetSearchService` interface.

- Example 1:

```
SearchResult searchResult = facetSearchService.search(searchQuery);
```

- Example 2, same as before with search hints:

```
SearchResult searchResult = facetSearchService.search(searchQuery, searchHints);
```

Using the Search Result

After executing the query we get a search result object. Some of the most common uses are:

- Get the number of results that match the search conditions. Example:

```
long numberOfResults = searchResult.getNumberOfResults();
```

- Get the result documents. This only returns a part of all the search results depending on the page size and offset in the search query. Example:

```
List<Document> documents = searchResult.getDocuments();
```

- Get the facets. Example:

```
List<Facet> facets = searchResult.getFacets();
```

Context and Listeners

Context

Some context objects are accessible during the search process. You can access them by using the corresponding factory, or, in some cases, as a parameter of the method (e.g. listeners).

The following context types are available:

- **FacetSearchContext** (together with FacetSearchContextFactory): this interface represents a context valid for the duration of a search.

Listeners

In this section you can find a list of the listener types that you can implement in order to intercept the search process.

FacetSearchListener

Use the FacetSearchListener interface if you want to intercept execution of search queries.

```
/**
 * Interface for receiving notifications about facet search execution.
 */
public interface FacetSearchListener
{
    /**
     * Handles a notification that a facet search service is about to begin execution.
     *
     * @param facetSearchContext
     *      - the {@link FacetSearchContext}
     *
     * @throws FacetSearchException
     *      if an error occurs
     */
    void beforeSearch(FacetSearchContext facetSearchContext) throws FacetSearchException;

    /**
     * Handles a notification that a facet search service has just completed.
     *
     * @param facetSearchContext
     *      - the {@link FacetSearchContext}
     *
     * @throws FacetSearchException
     *      if an error occurs
     */
    void afterSearch(FacetSearchContext facetSearchContext) throws FacetSearchException;

    /**
     * Handles a notification that a facet search service failed (this may also be due to listeners failing).
     *
     * @param facetSearchContext
     *      - the {@link FacetSearchContext}
     *
     * @throws FacetSearchException
     *      if an error occurs
     */
    void afterSearchError(FacetSearchContext facetSearchContext) throws FacetSearchException;
}
```

Search Hints

Search hints are a way to pass additional information to the search process. Those hints can be accessed at any time, they are part of the search context. No specific hint is supported yet.

Customized Sort Providers

This document provides information about customizing sort providers for the Solr facet search.

You can choose one of the sort providers that come out of the box with the `solrfacetsearch` extension. You can also define your own sort provider as described in this document.

Default Sort Providers

The `solrfacetsearch` extension comes with several sort providers. See the code below for example.

```
<bean id="facetNameSortProviderAscending" class="de.hybris.platform.solrfacetsearch.config.impl.DefaultFacetSortProvider"
      <property name="comparator" ref="facetDisplayNameComparator"/>
      <property name="descending" value="false"/>
    </bean>

    <bean id="facetCountSortProviderDescending" class="de.hybris.platform.solrfacetsearch.config.impl.DefaultFacetSortProvider"
      <property name="comparator" ref="facetCountComparator"/>
      <property name="descending" value="true"/>
    </bean>
    ...
```

The different sort providers always use the same base class:

`de.hybris.platform.solrfacetsearch.config.impl.DefaultFacetSortProvider`. The only deviation is a difference in the `Comparator` implementation and descending flag. For details on creating customized sort providers, see [Create Customized Sort Providers](#).

Create Customized Sort Providers

Context

Learn how to create a custom sort provider.

Procedure

1. Implement a comparator for the `de.hybris.platform.solrfacetsearch.search.FacetValue` class. This would be `Comparator<FacetValue>` and it can be used to sort your facet values. For example, assume you need to make sort by facet value name.

```
public class CustomFacetNameComparator implements Comparator<FacetValue>
{
    @Override
    public int compare(final FacetValue value1, final FacetValue value2)
    {
        if (value1 == null || value2 == null)
        {
            return 0;
        }
        return value1.getName().compareTo(value2.getName());
    }
}
```

2. Register your comparator in the Spring context and define a sort provider that uses it.

```
<!-- COMPARATOR -->
<bean id="customFacetNameComparator" class="de.hybris.platform.solrfacetsearch.search.impl.comparator.CustomFacetNameComparator"/>

<!-- SORT PROVIDER -->
<bean id="customFacetNameSortProviderAscending" class="de.hybris.platform.solrfacetsearch.config.impl.DefaultFacetSortProvider"
      <property name="comparator" ref="customFacetNameComparator"/>
      <property name="descending" value="false"/>
    </bean>
```

3. Provide localized names for your bean in the localization properties files in the `extension/resources/localization` directory. The convention for the property key is `solrfacetsearch.sortprovider.yourbeannamealllowercase`

```
#Sort provider bean names
solrfacetsearch.sortprovider.facetnamesortproviderascending=Sort by displayed name
solrfacetsearch.sortprovider.facetcountsortproviderdescending=Sort by facet value count
solrfacetsearch.sortprovider.sizeattributesortprovider=Sort by size
solrfacetsearch.sortprovider.numericfacetsortproviderdesc=Sort by numeric value descending
solrfacetsearch.sortprovider.numericfacetsortproviderasc=Sort by numeric value ascending

solrfacetsearch.sortprovider.customfacetnamesortproviderascending=Sort by name ascending
```

4. Compile the code base and restart the hybris server.

Direct Access to the Solr Query

This document provides information on the direct access to the solr queries.

i Note

This functionality is considered deprecated and only works when the legacy mode is enabled in the search configuration. Similar functionality can be achieved by using the `FacetSearchListener` interface. For more information, see [Search Process](#) and [Search API](#) documentation.

Overview

The post-processors enhance the default converter classes with additional logic that depends on the particular configuration and implementation.

Search Query Post-Processors

The default implementation of the `de.hybris.platform.solrfacetsearch.search.impl.DefaultFacetSearchService` service uses a dedicated query converter.

```
public interface SolrQueryConverter {
    /**
     * Converts {@link SearchQuery} instance into valid {@link SolrQuery}.
     */
    SolrQuery convertSolrQuery(SearchQuery searchQuery) throws FacetSearchException;
}
```

The main purpose of the API is to provide translation of the search query to the Solr server query. By default, the `convertSolrQuery` method is called in the `DefaultFacetSearchService` for the `de.hybris.platform.solrfacetsearch.search.impl.DefaultFacetSearchService.translateSearchQuery(SearchQuery)`.

The resulting query content comes from hybris to the Solr integration which is implemented in the `solrfacetsearch` extension. However, it is possible to use some other features of the Solr server, that are not yet supported directly by the `solrfacetsearch` extension.

This is the reason why the default implementation of the `de.hybris.platform.solrfacetsearch.search.SolrQueryConverter.convertQuery(SearchQuery)` method has been given given configurable query post-processing hooks that support customization of the default conversion process.

There is a configurable list of the query post-processors available. All the changes coming from these post-processors are applied in the same order as they appear in the configuration file. This is done at the end of the translation process.

Query converter is declared as a spring bean and used by the `DefaultFacetSearchService` service. Therefore all the post-processor hooks have to be declared at the converter level.

Custom Query Converter and Configuration

To declare a hook you need to implement the `de.hybris.platform.solrfacetsearch.search.SolrQueryPostProcessor` interface. The code sample below provides an example of such implementation.

```

public class SimpleSolrQueryPostProcessor implements SolrQueryPostProcessor
{
    @Override
    SolrQuery process(final SolrQuery query, final SearchQuery solrSearchQuery)
    {
        query.setSortField("id", ORDER.asc);
        query.setStart(Integer.valueOf(0));
        query.setRows(Integer.valueOf(10));
    }
}

```

Spring Configuration

The `defaultSolrQueryConverter` holds the list referencing all defined query post-processors. The post-processors can be defined in a form of utility list instance. The default translation defined in the converter is performed at the beginning of the process. The post-processors are applied to the query one after another in the end of the conversion process.

```

<alias name="defaultSolrQueryConverter" alias="solrQueryConverter"/>
<bean id="defaultSolrQueryConverter" class="de.hybris.platform.solrfacetsearch.search.impl.DefaultSolrQueryConverter">
    <property name="queryPostProcessors" ref="solrQueryPostProcessors"/>
    <property name="solrFieldNameProvider" ref="solrFieldNameProvider"/>
    <property name="defaultLimit" value="${facet.limit.default}"></property>
    <property name="forbiddenChar" value="${solr.indexedproperty.forbidden.char}"></property>
</bean>

<alias name="defaultSolrQueryPostProcessors" alias="solrQueryPostProcessors"/>
<util:list id="defaultSolrQueryPostProcessors">
    <ref bean="simpleSolrQueryPostProcessor" />
</util:list>

<bean id="simpleSolrQueryPostProcessor" class="my.package.SimpleSolrQueryPostProcessor" />

```

Search Result Post-Processors

Similarly to the query post-processors that operate on the Solr query object, there are also result post-processors that can modify the search result object returned from the Solr index search.

The search result post-processors work on the `de.hybris.platform.solrfacetsearch.search.impl.SolrSearchResult` level. If you want to register a post-processor you need to provide a proper spring configuration and create your own implementation.

Spring Configuration

All post-processors are registered in the `defaultFacetSearchService` by default.

```

<alias name="defaultFacetSearchService" alias="facetSearchService"/>
<alias name="defaultFacetSearchService" alias="solrFacetSearchService"/>
<bean id="defaultFacetSearchService" class="de.hybris.platform.solrfacetsearch.search.impl.DefaultFacetSearchService">
    <property name="userService" ref="userService" />
    <property name="i18nService" ref="i18nService" />
    <property name="solrService" ref="solrService"/>
    <property name="catalogService" ref="catalogService"/>
    <property name="fieldNameProvider" ref="solrFieldNameProvider"/>
    <property name="solrQueryConverter" ref="solrQueryConverter"/>
    <property name="resultPostProcessors" ref="solrSearchResultPostProcessors"/>
</bean>

<alias name="defaultSolrSearchResultPostProcessors" alias="solrSearchResultPostProcessors"/>
<util:list id="defaultSolrSearchResultPostProcessors">
    <ref bean="idleSolrResultPostProcessor" />
</util:list>

<bean id="idleSolrResultPostProcessor" class="my.package.IdleSolrResultPostProcessor" />

```

The search result post-processors must implement the `de.hybris.platform.solrfacetsearch.search.SolrResultPostProcessor` abstract class. The code sample contains the example.

```

public class IdleSolrResultPostProcessor implements SolrResultPostProcessor {

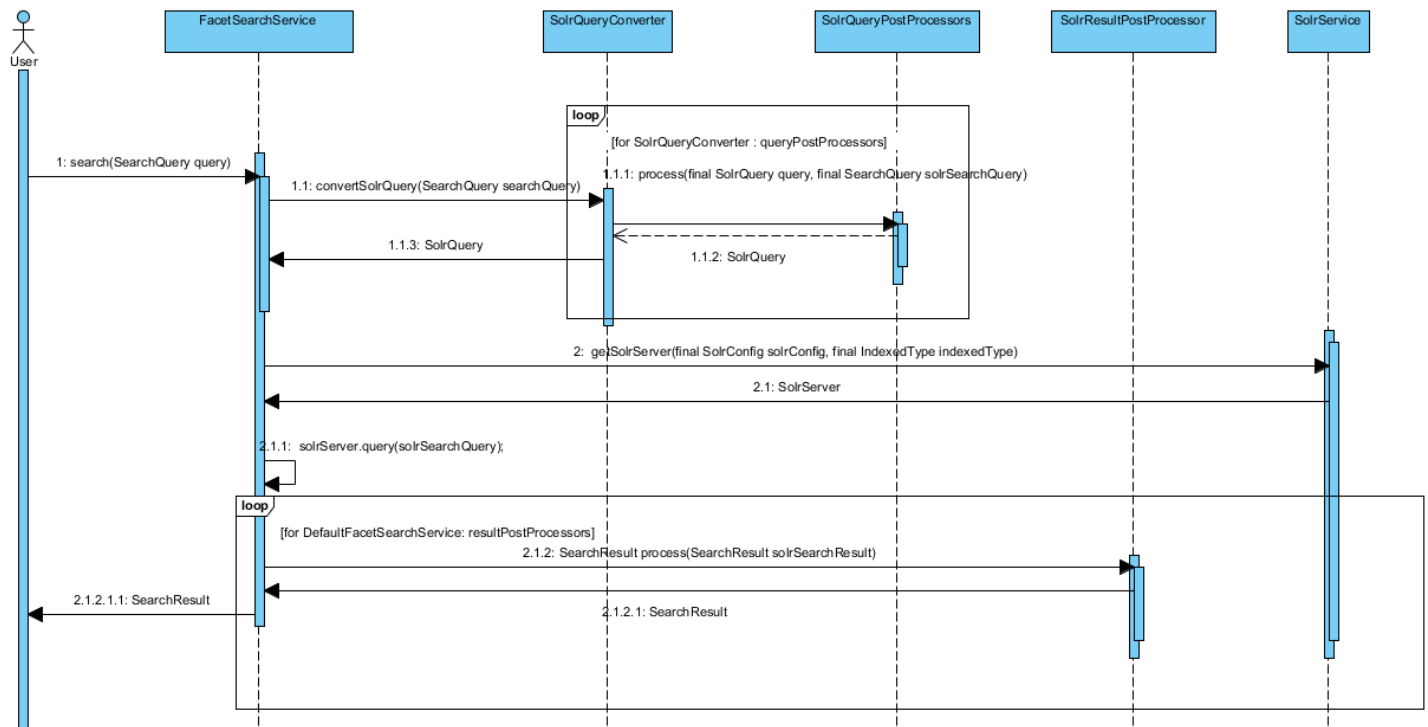
```

```

@Override
public SearchResult process(SearchResult solrSearchResult)
{
    //do sth with thte search result : hide some result DTO's, sort facet values, manipulate with breadcri
}

```

Overview of the Solr Query and Search Result Post-Processing



Further Customization of the Query Converter

For flexibility reasons it is possible to set several options in the implementation of the Solr query converter. The table provides explanation for these options.

Configurable Attribute Name	Description	Possible Values	Default Value
fieldOperator	Logical operator for the facet fields in the Solr query. The converter uses it when building the Solr query.	AND / OR	AND
facetSort	Sort option of the facet values in facets.	COUNT/INDEX	INDEX
defaultFacetLimit	Default limit of the facet values for a single facet.	int	project property = facet.limit.default
forbiddenChar	Char being the separator in the Solr fields names (for example, name_en_string)	char	project property = solr.indexedproperty.forbidden.char

The default setup of the Solr query converter.

```

<alias name="defaultSolrQueryConverter" alias="solrQueryConverter"/>
<bean id="defaultSolrQueryConverter" class="de.hybris.platform.solrfacetsearch.search.impl.DefaultSolrQueryConverter">
    <property name="queryPostProcessors" ref="solrQueryPostProcessors"/>
    <property name="solrFieldNameProvider" ref="solrFieldNameProvider"/>
    <property name="defaultLimit" value="${facet.limit.default}"></property>
    <property name="forbiddenChar" value="${solr.indexedproperty.forbidden.char}"></property>
</bean>

```

The following code sample provides an example of the customization in spring.

```
<alias name="customSolrQueryConverter" alias="solrQueryConverter"/>
  <bean id="customSolrQueryConverter" parent="defaultSolrQueryConverter">
    <property name="fieldOperator" value="OR"/>
    <property name="facetSort" value="COUNT"/>
  </bean>
```

Related Information

[Spring Framework in the SAP Commerce](#)

Custom Mapping Search Results

This topic describes how to supply custom data transfer objects and result converters.

i Note

This functionality is considered deprecated. If you need a similar functionality write a custom converter that converts a Document to your Data Transfer Object. For more details on how to get a list of documents from the search result, see [Search API](#).

You can convert the Solr search result into a convenient data transfer object. Additionally, you also can customize the conversion process by implementing your own converters and DTO classes. It helps you to process the information received from the Solr server without making additional round-trips to the database.

Result Converter

The Solr document conversion adds a performance value to the `solrfacetsearch` extension by allowing it to use the Solr result to display the indexed content in the web front. The goal is to transform the `SOLR org.apache.solr.common.SolrDocument`, which is the Solr API representation of a result item, into a lightweight, customized DTO object.

i Note

The `solrfacetsearch` extension comes with a simple implementation for the product conversion. In the following parts of the document it will serve as an example of:

- Converter: `de.hybris.platform.solrfacetsearch.search.product.DefaultSolrProductConverter` which converts the Solr result document into DTO object.
- DTO: `de.hybris.platform.solrfacetsearch.search.product.SolrProductData`

To benefit from the converted DTO object, you need two elements:

- DTO: It needs to be your own DTO class. The DTO object has to be a simple POJO object, which fits all the properties of the indexed content that you want to present.
- Converter: It needs to be your own Solr result converter implementation. The converter must implement the `de.hybris.platform.servicelayer.dto.converter.Converter<SOURCE, TARGET>` interface which represents the generic conversion API in the platform service layer.

Having implemented all the necessary classes, you need to bind the indexed type in your Solr configuration with your converter. Having done so, you can call the new special method from our search API `de.hybris.platform.solrfacetsearch.search.SearchResult<T> List<T> getResultData()`. This method returns DTO-enabled search results.

Using DTO-enabled Search Results

Context

The following sections describe the steps that are necessary in order to use DTO-enabled search results.

Procedure

1. Create a POJO class that contains the fields for the properties you want to read from the Solr search results. For example:
`de.hybris.platform.solrfacetsearch.search.product.SolrProductData`

SolrProductData

```
public class SolrProductData
{
    private String code;
    private String name;
    private String description;
    private String catalogVersion;
    private String catalog;
    private Long pk;
    private Collection<String> categories;
    private Double price;
    private String ean;

    /**
     * @return the code
     */
    public String getCode()
    {
        return code;
    }

    /**
     * @param code
     *         the code to set
     */
    public void setCode(final String code)
    {
        this.code = code;
    }

    /**
     * @return the name
     */
    public String getName()
    {
        return name;
    }

    /**
     * @param name
     *         the name to set
     */
    public void setName(final String name)
    {
        this.name = name;
    }

    /**
     * @return the description
     */
    public String getDescription()
    {
        return description;
    }

    /**
     * @param description
     *         the description to set
     */
    public void setDescription(final String description)
    {
        this.description = description;
    }

    /**
     * @return the catalogVersion
     */
    public String getCatalogVersion()
    {
        return catalogVersion;
    }

    /**
     * @param catalogVersion
     *         the catalogVersion to set
     */
    public void setCatalogVersion(final String catalogVersion)
    {
        this.catalogVersion = catalogVersion;
    }
}
```

```
}

/**
 * @return the catalog
 */
public String getCatalog()
{
    return catalog;
}

/**
 * @param catalog
 *        the catalog to set
 */
public void setCatalog(final String catalog)
{
    this.catalog = catalog;
}

/**
 * @return the pk
 */
public Long getPk()
{
    return pk;
}

/**
 * @param pk
 *        the pk to set
 */
public void setPk(final Long pk)
{
    this.pk = pk;
}

/**
 * @return the categories
 */
public Collection<String> getCategories()
{
    return categories;
}

/**
 * @param categories
 *        the categories to set
 */
public void setCategories(final Collection<String> categories)
{
    this.categories = categories;
}

/**
 * @return the price
 */
public Double getPrice()
{
    return price;
}

/**
 * @param price
 *        the price to set
 */
public void setPrice(final Double price)
{
    this.price = price;
}

/**
 * @return the ean
 */
public String getEan()
{
    return ean;
}

/**
 * @param ean
 *        the ean to set
 */
public void setEan(final String ean)
{
    this.ean = ean;
}
```

```

    }
}

```

The solrfacetsearch extension has an abstract implementation for the Solr result converters:

de.hybris.platform.solrfacetsearch.search.AbstractSolrConverter<T>. It implements the Converter interface (de.hybris.platform.servicelayer.dto.converter.Converter<SOURCE, TARGET>) and provides protected utility methods for getting the indexed properties values from the Solr documents. For translating the Solr indexed properties, it uses the de.hybris.platform.solrfacetsearch.provider.FieldNameProvider interface.

```

public abstract class AbstractSolrConverter<T> implements Converter<SolrResult, T>
{
    private FieldNameProvider fieldNameProvider;

    /**
     * Returns empty template instance for the conversion target
     */
    protected abstract T createDataObject();

    ....
}

```

2. Extend the AbstractSolrConverter class for your DTO class to indirectly implement the de.hybris.platform.servicelayer.dto.converter.Converter<SOURCE, TARGET> interface for it. For example, de.hybris.platform.solrfacetsearch.search.product.DefaultSolrProductConverter

```

public class DefaultSolrProductConverter extends AbstractSolrConverter<SolrProductData>
{
    private static final String CATALOG = "catalog";
    private static final String PK = "pk";
    private static final String DESCRIPTION = "description";
    private static final String NAME = "name";
    private static final String CATALOG_VERSION = "catalogVersion";
    private static final String CATEGORIES = "category";
    private static final String PRICE = "priceValue";
    private static final String CODE = "code";
    private static final String EAN = "ean";

    @Override
    public SolrProductData convert(final SolrResult solrResult, final SolrProductData target) throws Co
    {
        target.setCatalog(this.<String> getValue(solrResult, CATALOG));
        target.setPk(this.<Long> getValue(solrResult, PK));
        target.setName(this.<String> getValue(solrResult, NAME));
        target.setDescription(this.<String> getValue(solrResult, DESCRIPTION));
        target.setCatalogVersion(this.<String> getValue(solrResult, CATALOG_VERSION));
        target.setCategories((Collection<String>) this.getValue(solrResult, CATEGORIES));
        target.setPrice(this.<Double> getValue(solrResult, PRICE));
        target.setCode(this.<String> getValue(solrResult, CODE));
        target.setEan(this.<String> getValue(solrResult, EAN));

        return target;
    }

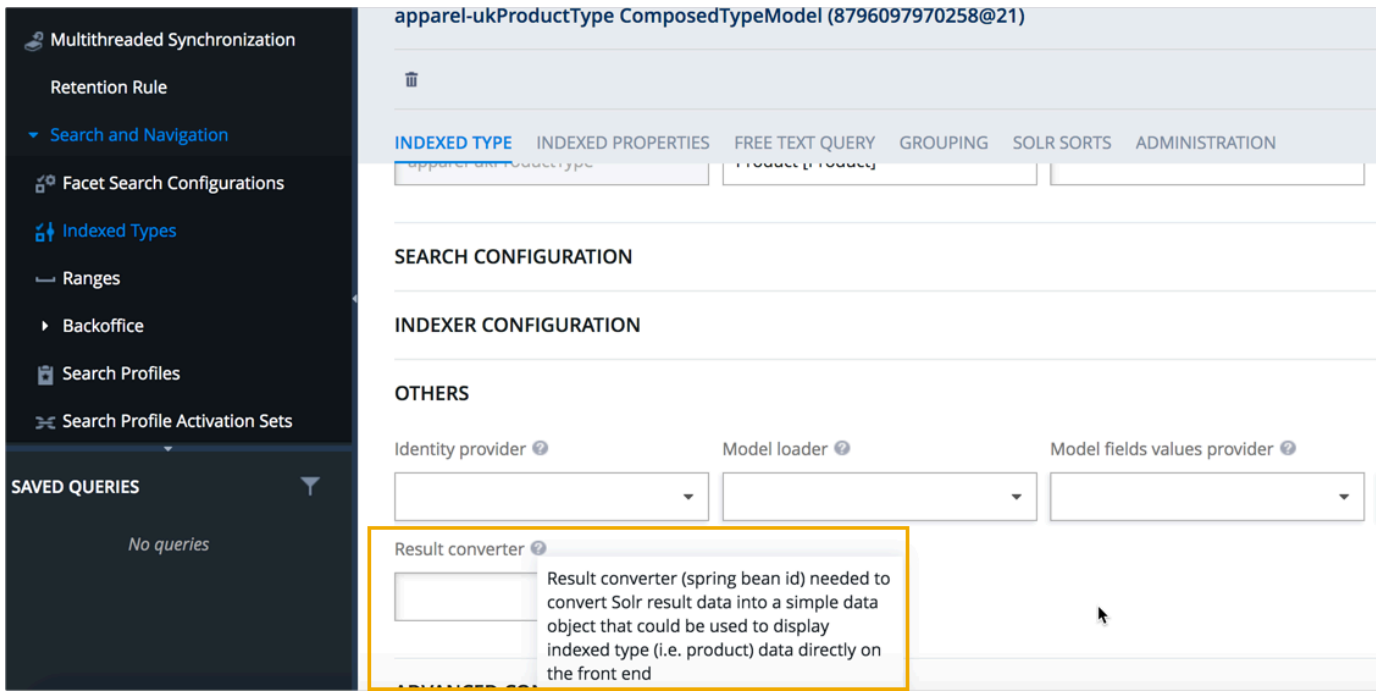
    @Override
    protected SolrProductData createDataObject()
    {
        return new SolrProductData();
    }
}

```

3. Register the converter in the Spring Context.


```
<alias name="defaultSolrProductConverter" alias="mySolrProductConverter"/>
<bean id="defaultSolrProductConverter" class="de.hybris.platform.solrfacetsearch.search.product.Def
```

4. If you use Backoffice Administration Cockpit to configure your Solr index, you find the corresponding editor field for the Solr document converter as well. Move the mouse over the label **Result converter** to display a help information providing short description of the text input box.



Hybris Utilities Module

E2E Supportability is used to gain more insight into a SAP Commerce installation and to enable Product Support to quickly identify possible issues. It allows you to view and manage customer orders or review the installation process either in case of project customization or upgrade scenarios.

Features

[Issue Analysis](#)

Architecture

[hybrisdatasupplier Extension](#)
[hybrisdatasupplierbackoffice Extension](#)
[hybrisrootcauseanalysis Extension](#)
[hybrisrootcauseanalysis Extension](#)

Issue Analysis

E2E Supportability is used to gain more insight into a SAP Commerce installation and to enable Product Support to quickly identify possible issues. It allows you to view and manage customer orders or review the installation process either in case of project customization or upgrade scenarios.

E2E thrives to achieve the following benefits:

- Avoid downtimes through proactive monitoring
- Increase system performance through analysis capabilities

- Reduce maintenance, integration and training costs by leveraging one toolset across products
- Reliable measure performance and identify exceptions
- Continuous integration support

It consists of the following general topics:

- Landscape Management
- Trace Analysis
- Workload Analysis
- Change Analysis
- Exception Analysis
- Technical Monitoring
- Transport and Deployment/CTS+

SAP Commerce implements the full feature set of E2E Supportability.

- Landscape Management - automatic registration of technical systems with installed software components into SAP Software Landscape Directory and SAP Landscape Management Database (like SAP Commerce in particular version or SAP DataHub in particular version); it is a prerequisite for Root Cause Analysis functionality (Trace Analysis, Workload Analysis, Change Analysis, Exception Analysis, Technical Monitoring)
- Transport and Deployment - transport of Java objects representing ongoing continuous integration between systems in the landscape (not to confuse with the landscape mentioned above)
- Root Cause Analysis features:
 - Trace Analysis - isolating a single user interaction through a complete landscape (by means of SAP Passport) and providing trace information on each of the involved components, starting with the user interaction in the browser going forward with data being managed by business logic and ending in the request coming back to the user
 - Workload Analysis - identifying performance issues by determining a bottleneck in a system
 - Change Analysis - monitor changes of crucial system settings either by inspecting configuration files or by listening to dynamically changing configuration on runtime
 - Exception Analysis - tracking and analysing of errors and warnings in the system
 - Technical Monitoring - monitoring SAP Commerce system being alive and gathering in one place several performance and health statistics

The following extensions are implementing E2E features:

- `hybrisdatasupplier` (Landscape Management)
- `hybrisrootcauseanalysis` (Root Cause Analysis)
- `hybristransportandchange` (Change and Transport)

Further Information

SAP Notes

- Data Supplier
 - Data Supplier for SAP Commerce (see [2018243](#) 📄)
 - Data Supplier for SAP Commerce standalone web applications (see [2187606](#) 📄)
 - SAP Commerce documentation with additional instructions about Data Supplier installation/configuration from a product perspective ([Running SAP Commerce Data Supplier](#))
- Change and Transport
 - Change and transport for SAP Commerce (see [2187631](#) 📄)
 - SAP Commerce documentation about change and transport scope and configuration
- Root Cause Analysis
 - Root cause analysis for SAP Commerce (see [2179990](#) 📄 with diagnostic templates)
 - Additional instructions for Introscope Java Agent for SAP Commerce (see [2181279](#) 📄)
 - Main SAP Note for Root Cause Analysis in Solution Manager 7.1 (SAP Commerce reference) (see [1478974](#) 📄).

- o Main SAP Note for Root Cause Analysis in Solution Manager 7.2 (SAP Commerce reference) (see [2248724](#) 📄).

SAP Wiki

Main SAP Wiki for System Monitoring (part of Technical Monitoring) in Solution Manager (SAP Commerce reference) (see <https://wiki.scn.sap.com/wiki/x/TQK9Dg> 📄)

Meet the Expert Sessions Recordings

These are short demos of available functionality prepared by SAP developers:

i Note

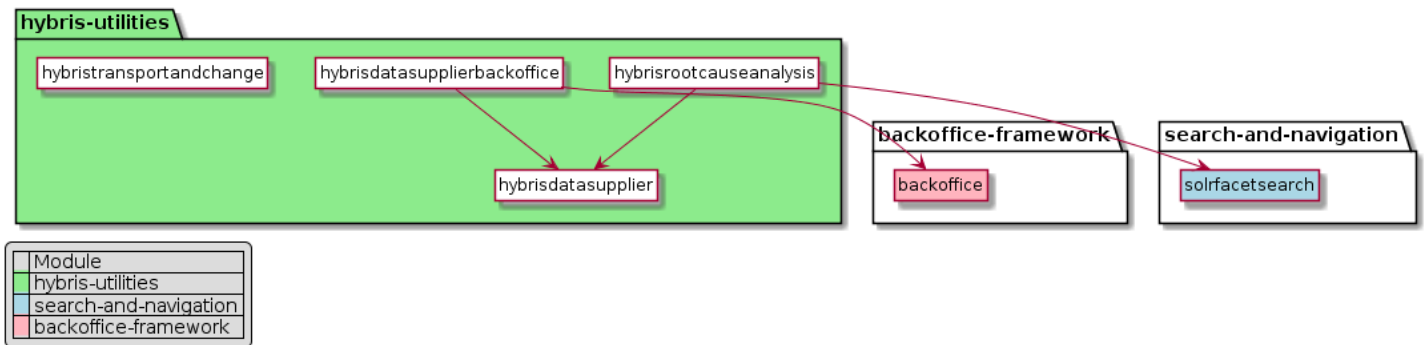
Please note that this content is only available to you if your company has purchased access to the **SAP Learning Hub**.

- [Integration with Change and Transport System \(CTS+\) 📄](#)
- [E2E Supportability: Installation and Configuration 📄](#)
- [Root Cause Analysis for SAP Commerce in SAP Solution Manager 📄](#)

Hybris Utilities Module Architecture

The Hybris Utilities module provides a set of extensions that implement E2E support in SAP Commerce.

Dependencies



For more information, see [SAP Commerce Directory Structure](#).

Recipes

The Hybris Utilities module is not included in any of the available SAP Commerce recipes.

For a full list of available installation recipes, see [Installer Recipes](#).

Extensions

The Hybris Utilities module consists of the following extensions:

[hybrisdatasupplier Extension](#)

The `hybrisdatasupplier` extension implements Landscape Management as part of the E2E support.

[hybrisdatasupplierbackoffice Extension](#)

The `hybrisdatasupplierbackoffice` extension implements Landscape Management for Backoffice as part of the E2E support.

[hybrisrootcauseanalysis Extension](#)

The `hybrisrootcauseanalysis` extension implements Root Cause Analysis as part of the E2E support.

[hybristransportandchange Extension](#)

The `hybristransportandchange` extension implements the Change and Transport feature as part of the E2E support.

hybrisdatasupplier Extension

The hybrisdatasupplier extension implements Landscape Management as part of the E2E support.

About the Extension

Name	Directory	Related Module
hybrisdatasupplier	hybris/bin/modules/hybris-utilities	Hybris Utilities Module Architecture

hybrisdatasupplierbackoffice Extension

The hybrisdatasupplierbackoffice extension implements Landscape Management for Backoffice as part of the E2E support.

About the Extension

Name	Directory	Related Module
hybrisdatasupplierbackoffice	hybris/bin/hybris-utilities	Hybris Utilities Module Architecture

hybrisrootcauseanalysis Extension

The hybrisrootcauseanalysis extension implements Root Cause Analysis as part of the E2E support.

About the Extension

Name	Directory	Related Module
hybrisrootcauseanalysis	hybris/bin/modules/hybris-utilities	Hybris Utilities Module Architecture

hybristransportandchange Extension

The hybristransportandchange extension implements the Change and Transport feature as part of the E2E support.

About the Extension

Name	Directory	Related Module
hybristransportandchange	hybris/bin/modules/hybris-utilities	Hybris Utilities Module Architecture