Enfinity Suite 6

Advanced Search Module Implementation and Customization

2010-05-10 Document ID: EMS-LIB-64-21-01

Table of Contents

CHAPTER 1

Overview

About this Guide

By default, Enfinity Suite 6 uses the Oracle ConText search to query database data. In order to provide a more powerful full-text search, guided search and other advanced search features, Enfinity Suite 6 also integrates an advanced search module (ASM) for product and content searches in consumer and business channel storefronts. This interface enables the integration of external search engines, which then build and query search indexes and add search features such as full-text search, guided search, stop word and synonym configuration to Enfinity Suite 6-based e-commerce applications.

This document describes the main features of the advanced search module (ASM) and discusses important administration and development issues.

Knowledge Assumed

This paper is addressed to both Enfinity Suite 6 developers and Enfinity Suite 6 administrators. The paper assumes familiarity with the technical architecture of Enfinity Suite 6, as well as the overall cartridge development process.

Typographical Conventions

The following typographical conventions are used throughout the guide:

- **Cross-references**
  References to other parts of this guide and to other documentation appear in *italics*.
- **Commands**
  All commands to be typed at command prompts appear in `Courier` font.
- **Reserved or Special Words**
  Names of files, directories, or cartridges appear in *italics*.

Example code, attribute names, methods and database table names appear in Courier; for example, `init()`. In addition, the `#` sign refers to the number of an Enfinity Suite 6 instance. If `#` precedes a shell prompt, it indicates that the current user is `root`.

Chapter Overview

The remainder of this chapter introduces the advanced search module, outlining general search options.

Chapter 2 outlines the general ASM functionalty, including its activation, administration and general back-office usage.

Chapter 3 covers the ASM implementation and customization.

Advanced Search Module Overview

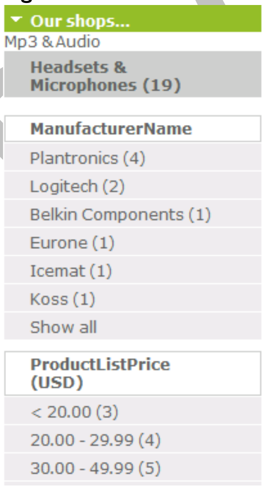
Through integrating an external search engine, the advanced search module provides powerful search options to Enfinity Suite 6. Depending on the actual search engine, these options may include:

- **Simple Search Index Interface**
  The advanced search module interface supports the creation and configuration of search indexes to include specified standard and custom product attributes for product searches and content parameters for content searches.

- **Synonyms and Antonyms**
  The advanced search module interface provides the ability to configure and store synonym and antonym configurations.

- **Guided Search ("After Search Navigation")**
  The advanced search module interface provides a standard pipeline and template set to add a guided search function to an Enfinity Suite 6 storefront. When a user submits a query to the simple search, a set of filter attributes is presented along with the results. The filter attribute values can be used to refine the search, hence narrow down the result set. Behind each filter attribute value, a number indicates how many hits the respective filter attribute value has produced.

Figure 1: Guided search in the consumer storefront

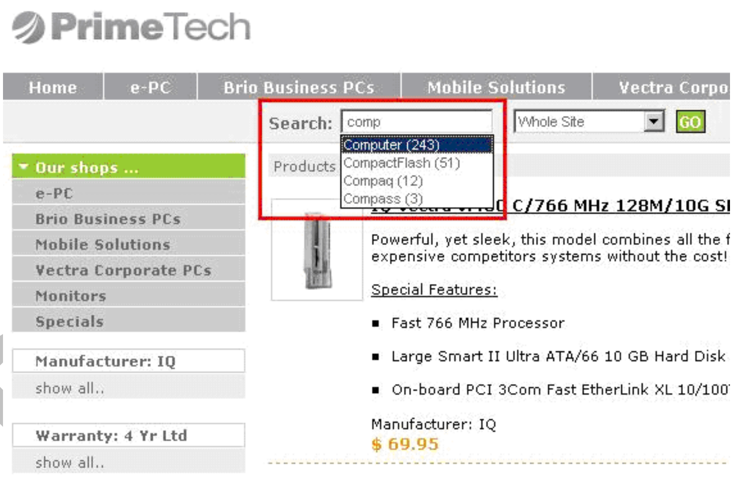


- **Dynamic Navigation**
  The advanced search module interface can be used to implement a dynamic navigation. Dynamic navigation facilitates catalog browsing by indicating for each category how many products the individual categories and sub-categories contain.

> **NOTE:** Dynamic navigation can be enabled via the search index preferences.

- **Suggest Function**
  The advanced search module interface can also provide a dynamic recommendation function that offers possible completions for search term fragments while typing the search term. Along with each recommended completion, the number of possible hits, an optional image and a suggestion type may be indicated.

Figure 2: Example for a suggest function based on Omikron FACT-Finder



C/766 MHz 128M/10G SI

Powerful, yet sleek, this model combines all the f
expensive competitors systems without the cost!

Special Features:

- Fast 766 MHz Processor
- Large Smart II Ultra ATA/66 10 GB Hard Disk
- On-board PCI 3Com Fast EtherLink XL 10/100

Manufacturer: IQ
$ 69.95

CHAPTER 2

Common ASM Features

Enabling the ASM

Setting up an advanced search module implementation comprises, basically,

- installing a corresponding adapter cartridge, and
- setting up the external search engine as required.

This adds new search index types to your Enfinity Suite 6 installation. After deploying the intended search engine and the corresponding adapter cartridge, the new search index types must be enabled within the intended channel(s). Consequently, the new search indexes must be created, configured and built to become operative.

NOTE: The Search Index Preferences page is only available if there is an advanced search module implementation installed.

To enable the advanced search module, proceed as follows:

1. **In the Enfinity Suite 6 back office, select the intended channel from the channel bar.**

   This opens the navigation bar of the channel.

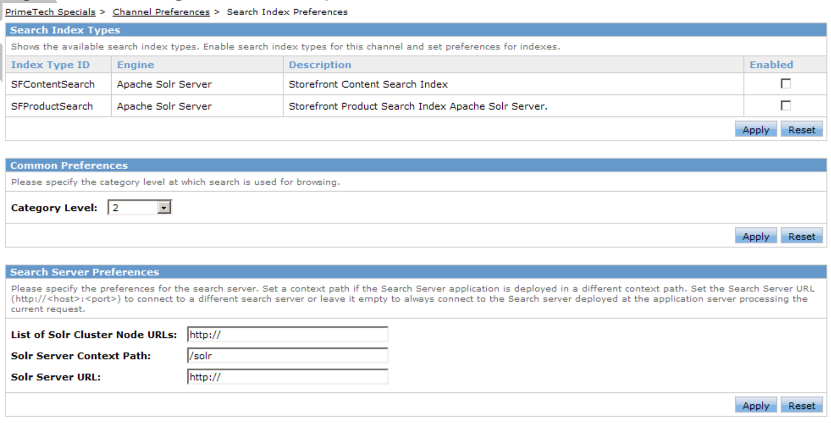
2. **In the navigation bar, select Preferences.**

   The overview page is displayed.

3. **Select Search Indexes.**

   The Search Index detail page is displayed.

Figure 3: Defining search index preferences



PrimeTech Specials > Channel Preferences > Search Index Preferences

Search Index Types

Shows the available search index types. Enable search index types for this channel and set preferences for indexes.

Index Type ID	Engine	Description	Enabled
SFContentSearch	Apache Solr Server	Storefront Content Search Index	☐
SFProductSearch	Apache Solr Server	Storefront Product Search Index Apache Solr Server.	☐

Common Preferences

Please specify the category level at which search is used for browsing.

Category Level: 2

Search Server Preferences

Please specify the preferences for the search server. Set a context path if the Search Server application is deployed in a different context path. Set the Search Server URL (http://<host>:<port>) to connect to a different search server or leave it empty to always connect to the Search server deployed at the application server processing the current request.

List of Solr Cluster Node URLs:	http://
Solr Server Context Path:	/solr
Solr Server URL:	http://

NOTE: The available options depend on the given search module. Figure 3 shows the options for the Apache Solr-based ASM as an example.

By default, the advanced search options are not selected. That is, the standard Oracle ConText search is enabled.

4. **Select the intended options.**

   The available options depend on the given search module. Generally, the following index types are used within the standard storefront implementation:

   - SFProductSearch

     This type indexes product data to support features like product search, after search navigation, etc.
   - SFContentSearch

     This type indexes page and component data to support content search within the storefront editing content.

5. **Optionally, select a category level.**

   This enables the guided navigation at the selected category level.

6. **Optionally, specify preferences for the given search engine.**

   The available options depend on the given search engine and its actual implementation. For details, refer to the corresponding search module documentation.

7. **Click Apply.**

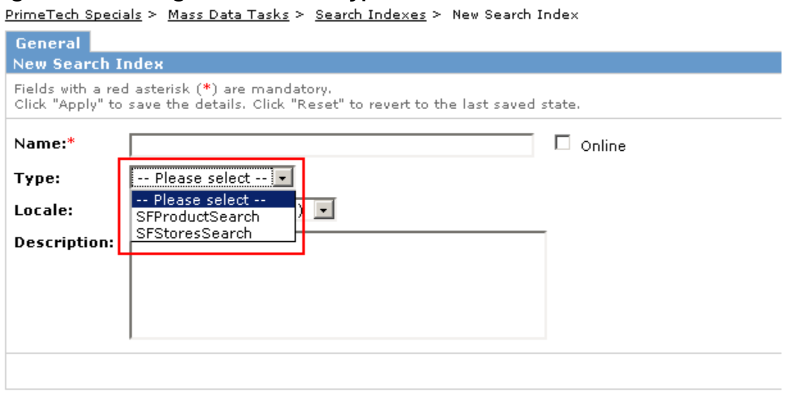
   This enables the settings immediately, i.e., without restarting the Enfinity Suite 6 server.

Using Multiple Indexes

You can define multiple index configurations, hence work with multiple indexes in parallel. For example, when supporting two different locales with your storefront (e.g., en_US and de_DE), you obviously need one index for en_US and another index for de_DE data.

Moreover, you can define separate index files for different index types. For example, assume you have implemented a new index type SFStores for a store finder (see Figure 4). The index to be used by the SFStores store finder index requires a configuration that differs from the one used for the standard storefront search.

Figure 4: Selecting a search index type



PrimeTech Specials > Mass Data Tasks > Search Indexes > New Search Index

General

New Search Index

Fields with a red asterisk (*) are mandatory.
Click "Apply" to save the details. Click "Reset" to revert to the last saved state.

Name:*

Online

Type: -- Please select --

-- Please select --
SFProductSearch
SFStoresSearch

Locale:

Description:

Index types are defined using a dedicated configuration file (*searchindexfeatures.properties*) included with the adapter cartridge *ac_search_**. Note that you can add new search index types upon customizing the search module (see *ASM Implementation and Customization*, on p. 26).

NOTE: You can only use one online index per type (feature) and locale. You can, however, create and build additional offline indexes.

Common ASM Concepts

This section introduces general search index management concepts, which are common for all advanced search module implementations.

NOTE: Managing indexes in the Enfinity Suite 6 back office requires the permission *SLD_MANAGE_SEARCH_INDEXES*. By default, this permission is assigned to the access privilege “Catalog Manager”.

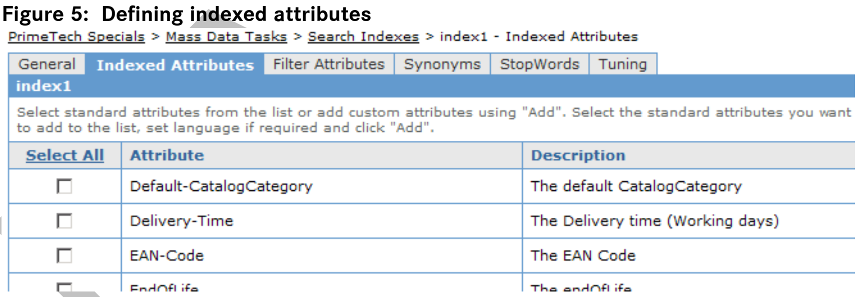
The index creation (and search, consequently) is based on configurable attributes. Hence, indexing is generally based on defining index attributes and filter attributes, as well as on defining synonyms and stop words.

NOTE: Any additional search functionality (tuning, campaigns, etc.) depends on the actual search engine implemented as advanced search module in Enfinity Suite 6.

Index Attributes

When creating an index in the Enfinity Suite 6 back office, the responsible user has to select the product attributes to be included in the index.

Figure 5: Defining indexed attributes


PrimeTech Specials > Mass Data Tasks > Search Indexes > index1 - Indexed Attributes
General
Indexed Attributes
Filter Attributes
Synonyms
StopWords
Tuning
index1

Select standard attributes from the list or add custom attributes using "Add". Select the standard attributes you want to add to the list, set language if required and click "Add".

Select All	Attribute	Description
☐	Default-CatalogCategory	The default CatalogCategory
☐	Delivery-Time	The Delivery time (Working days)
☐	EAN-Code	The EAN Code
☐	EndOfLife	The endOfLife

- Certain standard product attributes are automatically included with the index, most of them for technical purposes, such as “productuuid”. These attributes cannot be managed on the Indexed Attributes tab. In addition, the index automatically includes information regarding catalog and category structure, starting with the top level category 0, down to category level 5. This information is used to enable the filter mechanism using catalog categories in the storefront.

  If the multiple category assignment attribute is added to the index, it is possible to include the category assignments for products that are assigned to more than one category. The category levels 1 through 5 are then replaced by the special CategoryUUIDLevelMulti field. The category display names for the product are still indexed and are taken from the default category.

- Business attributes defined on products are offered on the Indexed Attributes tab, where they can be selected for inclusion in the index (see Figure 5).
- Custom attributes defined on a product can be included in the index using the input fields on the Indexed Attributes tab. To include all custom attributes, select the option “AllCustomAttributes”.

NOTE: Although you can select all custom attributes to be indexed, Intershop recommends to select only those attributes that contain useful, searchable information.

The standard product attributes to be indexed automatically as well as additional business attributes to be offered on the Indexed Attributes tab of the Enfinity Suite 6 back office are defined in the search index type definition file for the corresponding search index.

NOTE: The sections above describe the standard implementation of managing indexed attributes provided by Enfinity Suite 6. It can be overridden for specific index types (see *Search Index Types,* on p. 26).

Filter Attributes

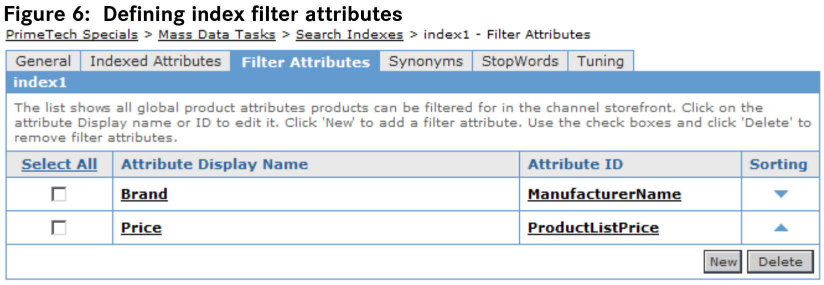
Enfinity Suite 6 provides a convenient way to easily configure filter attributes for guided search in the consumer storefront (see *Advanced Search Module Overview*, on p. 5).

Two types of filter attributes are available:

■ **Index Filter Attributes**

Index filter attributes are defined in the Search Index area of the back office. In Figure 6, two index filter attributes are defined, “Brand” and “Price”. See *Figure 1*, on p. 5 on how these filter attributes are displayed in the storefront.

Figure 6: Defining index filter attributes



PrimeTech Specials > Mass Data Tasks > Search Indexes > index1 - Filter Attributes

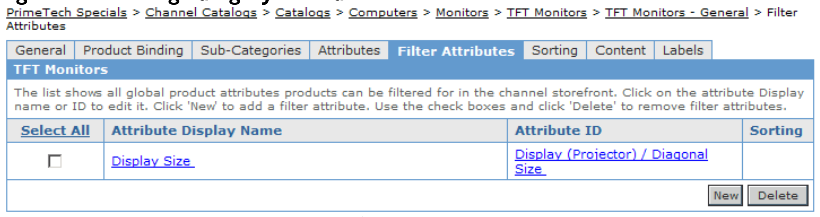
The list shows all global product attributes products can be filtered for in the channel storefront. Click on the attribute Display name or ID to edit it. Click 'New' to add a filter attribute. Use the check boxes and click 'Delete' to remove filter attributes.

Select All	Attribute Display Name	Attribute ID	Sorting
☐	Brand	ManufacturerName	▼
☐	Price	ProductListPrice	▲

Index filter attributes are defined specifically for each index, with the according definitions stored in the corresponding configuration file (see *Configuration Files*, on p. 19). They are applied to all objects of the index.

- **Category Filter Attributes**
  Category filter attributes are defined in the Channel Catalog area of the back office. This mechanism is used for filter attributes that are relevant for specific categories only. For example, in Figure 7, a filter attribute “Display Size” has been defined for the catalog category “TFT Monitors”. In the PrimeTech demo storefront, this filter attribute will only be displayed if users submit a simple search and then drill down to the category “TFT Monitors”.

Figure 7: Defining category filter attributes



PrimeTech Specials > Channel Catalogs > Catalogs > Computers > Monitors > TFT Monitors > TFT Monitors - General > Filter Attributes

TFT Monitors

The list shows all global product attributes products can be filtered for in the channel storefront. Click on the attribute Display name or ID to edit it. Click 'New' to add a filter attribute. Use the check boxes and click 'Delete' to remove filter attributes.

Select All	Attribute Display Name	Attribute ID	Sorting
☐	Display Size	Display (Projector) / Diagonal Size	

Category filter attributes are only applied to products that are assigned to the category on which the filter attribute has been defined. Note that category filter attributes are defined globally for all indexes of a site.

The category specific filter attributes are stored in a dedicated configuration file, *categoryFilterAttributes.xml*, in the indexes directory (see *Configuration Files*, on p. 19).

NOTE: Make sure one and the same attribute is not defined both as index filter attribute and category filter attribute.

When defining an attribute, you set the “unit” (e.g., inch) to display with the filter attribute values. You also determine the data type (e.g., string or integer). The data type set for an attribute in turn determines how the attribute values are displayed.

NOTE: These are standard configuration views to manage filter attributes as deployed by default. The actual view depends on the implementation of the used search engine.

Synonyms

Synonyms are important to inform a search engine which words have similar meanings, without any similarity in form (for example: “Jeans” -> “Pants”).

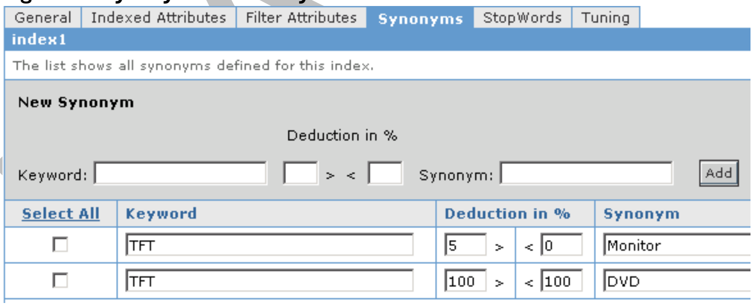
Antonyms are used to tell a search engine that two words have nothing to do with one another, despite the fact that a search engine would report a similarity because the forms are similar (for example: “TFT” -> “DVD”).

The advanced search module provides a configuration view to define both synonyms and antonyms in the back office on the Synonym tab of the respective index. The difference is in the reduction used for the word relationship. Synonyms have a word relationship reduction of 0%. Antonyms have a reduction of 100%. Intermediate values may be used to define word relationships that are not exactly synonyms, but also not antonyms. For example, Jeans -> Pants. This word relationship could be given a slight reduction (such as 5%), so that a search for “Pants” would also find “Blue Jeans”, but still give “corduroy pants” earlier in the list.

Synonym and antonym relations defined in the back office are mapped to entries within the dedicated synonyms section in the general configuration file (see *Configuration Files*, on p. 19). A search engine specific implementation may map these synonym configurations to the search engine-specific configurations or may replace the synonym configuration tab completely if the search engine provides its own configuration pages.

For each synonym and antonym relation, you can define the reduction separately depending on the direction in which the relation is traversed. For example, in Figure 8, when a search engine searches semantic relations for “TFT”, it may identify “Monitor” as synonym item, but apply a 5% reduction. On the other hand, when looking for items semantically related to “Monitor”, a search engine may identify “TFT” as a synonym, without applying a reduction. Also in Figure 8, an antonym relation is defined connecting “TFT” and “DVD”, indicating that these words are not related semantically.

Figure 8: Synonym and antonym relations



index1

The list shows all synonyms defined for this index.

New Synonym

Deduction in %

Keyword: > < Synonym:

Select All	Keyword	Deduction in %		Synonym
☐	TFT	5 >	< 0	Monitor
☐	TFT	100 >	< 100	DVD

NOTE: Depending on the actually integrated search engine, the synonym evaluation may differ. Possible differences in the search approach may include, among others, recursive evaluation, synonym look up (exact or fuzzy search) or search for single words only or phrases.

Stop Words

Stop words can be defined in the back office to exclude words from the index that have no relevance. A typical example are function words such as “the”, “a”, etc., which have little meaning and can usually be ignored when searching.

Stop word definitions defined in the back office are mapped to a dedicated stopword section in the general configuration file (see *Configuration Files*, on p. 19).

CHAPTER 3

Implementation and Customization

Advanced Search Module Architecture

This section gives an overview of the standard ASM components as included with Enfinity Suite 6. This covers the involved cartridges, the configuration files, the involved pipelines and the standard template integration.

Cartridge Structure

The functionality provided by advanced search module is distributed across cartridges as follows:

- The cartridge *bc_foundation* defines the main CAPI interfaces that expose the functionality of the advanced search module. In addition, *bc_foundation* and *bc_mvc* provide a number of basic pipelets that access these interfaces and that are used by the search pipelines in the storefront. For more details, refer to the *searchindex* packages in the JavaDocs generated for these cartridges.
- The interfaces and base implementation classes exposed by *bc_foundation*, *bc_mvc* and *bc_pmc* are implemented or extended by resources contained in the search engine-specific *ac_search_** cartridge.
- In addition, pipelines and templates provide a common user interface to create, configure, build and update search indexes in the backoffice (*sld_ch_base*) and to facilitate search and navigation with search indexes in the storefront (*sld_ch_consumer_app*, *sld_ch_adv_corporate_app*).

Figure 9 gives a graphical overview of the involved cartridges, and Table 1 lists the ASM specific contents of each involved cartridge.

Figure 9: Cartridges including ASM functionality

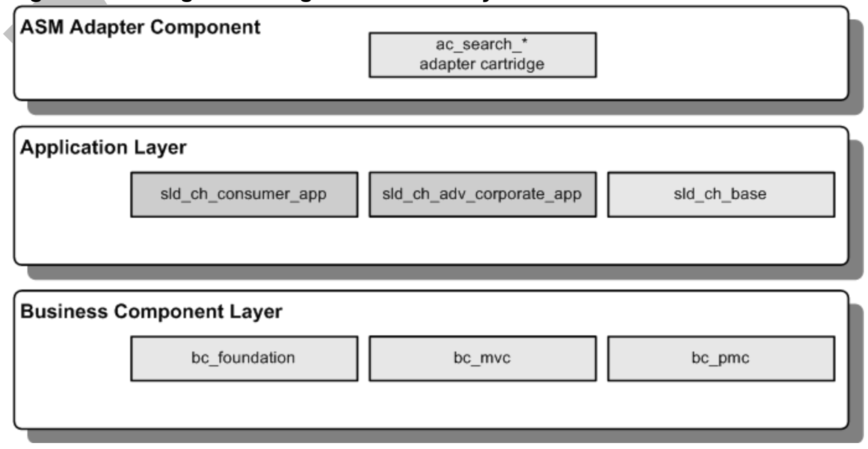


Table 1: ASM-specific cartridge contents

Cartridge	ASM Contents
bc_foundation	Interface and base implementations, pipelets, pipelines.
bc_mvc	Data providers for catalog and product data.
bc_pmc	Data providers for storefront editing (pagelet) content.
sld_ch_consumer_app	Pipelines and templates for the B2C channel storefront.
sld_ch_adv_corporate_app	Pipelines and templates for the B2B channel storefront.
sld_ch_base	Pipelines and templates for the search index-specific back office modules.
sld_system_app	Job pipeline for the index creation.
ac_search_*	Search engine-specific adapter cartridge(s) that implement the ASM interfaces provided by the above cartridges.

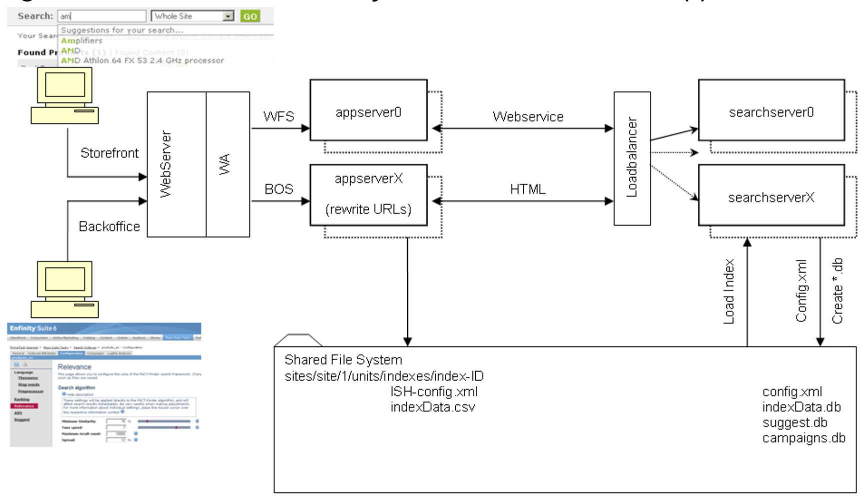
Index Directory and Server Structure

Since several search engines operate with file based indexes, the advanced search module provides means to store these indexes into the Shared File System of Enfinity Suite 6. Depending on the given search engine, an index file may be created when building the index. When rebuilding the index (e.g., when structural changes have been applied to the index configuration), the index file is updated or created anew, and a backup of the old index file is saved.

Each index defined in the back office has its own index directory. The index directory is located in the unit directories of the site for which the indexes have been defined. For example, the index files for the PrimeTechSpecials consumer storefront are located in *<IS.INSTANCE.SHARE>/sites/PrimeTech-PrimeTechSpecials-Site/1/units/PrimeTech-PrimeTechSpecials/indexes/<index_name>*.

Figure 10 illustrates – based on FACT-Finder as an example – the general deployment and the information flow between the Enfinity Suite 6 application servers and one or more search servers. As you see, the search index files are located in the Shared File System of Enfinity Suite 6, and the search requests are sent to the search server(s) using a search engine-specific client library.

Figure 10: Data flow between Enfinity Suite 6 and the search server(s)



Configuration Files

An ASM implementation is configured using a general configuration file, *ISH-config.xml*, and search engine-specific configuration files, which are created automatically when configuring an index in the back office. The configuration files are index-specific, hence contained in *<IS.INSTANCE.SHARE>/sites/<site>/<active_directory>/units/<unit>/indexes/<indexID>*.

The general *ISH-config.xml* file defines the Enfinity Suite 6-specific search index configuration. It is written by the search index configuration class of the foundation cartridge. The file stores the attributes currently selected to be indexed, as well as additional, basic information about the index. This general data is displayed in the General tab of the back office Search Index dialog.

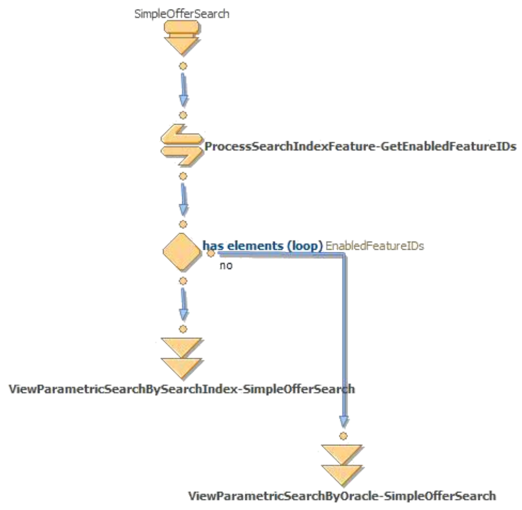
CAUTION: ASM implementations may use *ISH-config.xml* and its engine specific configuration files to automatically synchronize each others configuration. Hence, Intershop recommends not to modify the configuration manually but use the configuration back office (unless stated otherwise). When modifying the configuration settings manually, make sure to restart the application server since your changes may be overwritten from an in-menory representation of the index configuration.

Search Pipelines

Generally, the advanced search module is intended to replace the so-called "Simple Search". In the demo application, the "Advanced Search" option, which continues to be based on the Oracle ConText search, is hidden.

The pipeline that is executed when submitting a simple product search is *ViewParametricSearch-SimpleOfferSearch*. This pipeline first checks whether there is a search index type enabled (see *Enabling the ASM*, on p. 8). If so, the pipeline jumps to the target pipeline *ViewParametricSearchBySearchIndex*. If not, the standard search pipeline *ViewParametricSearchByOracle* is used as fallback.

Figure 11: Search pipeline gateway


SimpleOfferSearch
ProcessSearchIndexFeature-GetEnabledFeatureIDs
has elements (loop) EnabledFeatureIDs
no
ViewParametricSearchBySearchIndex-SimpleOfferSearch
ViewParametricSearchByOracle-SimpleOfferSearch

NOTE: Requests to all other sub-pipelines of *ViewParametricSearch* are automatically routed the corresponding sub-pipeline of *ViewParametricSearchByOracle*.

Table 2 lists the standard pipelines that provide ASM functionality.

Table 2: ASM pipelines

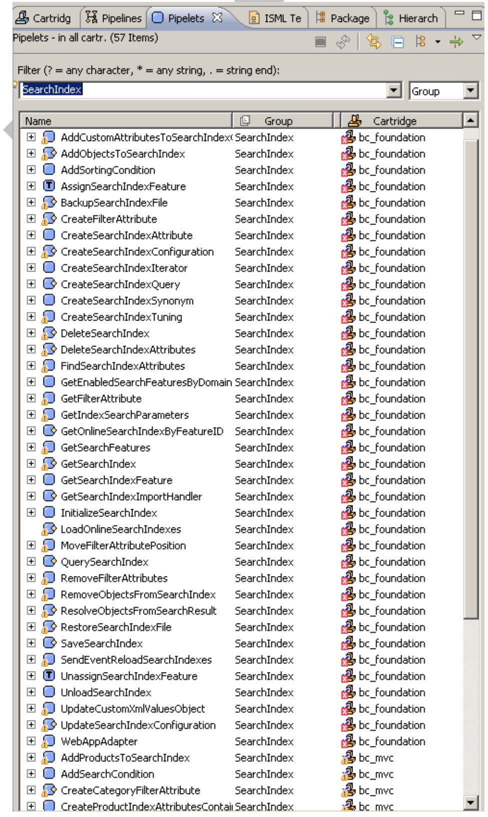
Pipeline	Cartridge	Description
ViewParametric SearchBySearchIndex	sld_ch_consumer_app sld_ch_corporate_adv_app	Storefront viewing pipeline that handles the standard search for the SFProductSearch and SFContentSearch index types.
ProcessParametric SearchBySearchIndex	sld_ch_consumer_app sld_ch_corporate_adv_app	Processing pipeline providing standard processing of ASM search, to the *ViewParametricSearchBySearch Index* pipeline.
ViewSearchIndex Preferences	sld_ch_base	Handles activation/deactivation of search index types and additional preferences for search indexes.
ViewSearchIndex	sld_ch_base	The main back-office pipeline that handles the creation, update and deletion of search indexes.

Pipeline	Cartridge	Description
ViewCustom Searchindex	sld_ch_base	Contains additional back-office functionality for index configuration.
ProcessSearchIndex	sld_ch_base	Process pipeline for back-office functionality, includes also the batch process implementation for building and updating indexes.
ProcessSearchIndex Feature	bc_foundation	Processing pipeline to handle available search index types and search index features.
UpdateSearchIndexes Job	sld_system_app	Job pipeline that updates the available online indexes of the system.

Search Pipelets

The standard pipelines as mentioned above use the provided standard pipelets related to the advanced search module. The search-related pipelets as shipped with Enfinity Suite 6 are organized in the pipelet group SearchIndex (see Figure 12).

Figure 12: Enfinity Studio listing the search-related pipelets



Pipelets - in all cartr. (57 Items)

Filter (? = any character, * = any string, . = string end):

SearchIndex

Group

Name	Group	Cartridge
AddCustomAttributesToSearchIndex(	SearchIndex	bc_foundation
AddObjectsToSearchIndex	SearchIndex	bc_foundation
AddSortingCondition	SearchIndex	bc_foundation
AssignSearchIndexFeature	SearchIndex	bc_foundation
BackupSearchIndexFile	SearchIndex	bc_foundation
CreateFilterAttribute	SearchIndex	bc_foundation
CreateSearchIndexAttribute	SearchIndex	bc_foundation
CreateSearchIndexConfiguration	SearchIndex	bc_foundation
CreateSearchIndexIterator	SearchIndex	bc_foundation
CreateSearchIndexQuery	SearchIndex	bc_foundation
CreateSearchIndexSynonym	SearchIndex	bc_foundation
CreateSearchIndexTuning	SearchIndex	bc_foundation
DeleteSearchIndex	SearchIndex	bc_foundation
DeleteSearchIndexAttributes	SearchIndex	bc_foundation
FindSearchIndexAttributes	SearchIndex	bc_foundation
GetEnabledSearchFeaturesByDomain	SearchIndex	bc_foundation
GetFilterAttribute	SearchIndex	bc_foundation
GetIndexSearchParameters	SearchIndex	bc_foundation
GetOnlineSearchIndexByFeatureID	SearchIndex	bc_foundation
GetSearchFeatures	SearchIndex	bc_foundation
GetSearchIndex	SearchIndex	bc_foundation
GetSearchIndexFeature	SearchIndex	bc_foundation
GetSearchIndexImportHandler	SearchIndex	bc_foundation
InitializeSearchIndex	SearchIndex	bc_foundation
LoadOnlineSearchIndexes	SearchIndex	bc_foundation
MoveFilterAttributePosition	SearchIndex	bc_foundation
QuerySearchIndex	SearchIndex	bc_foundation
RemoveFilterAttributes	SearchIndex	bc_foundation
RemoveObjectsFromSearchIndex	SearchIndex	bc_foundation
ResolveObjectsFromSearchResult	SearchIndex	bc_foundation
RestoreSearchIndexFile	SearchIndex	bc_foundation
SaveSearchIndex	SearchIndex	bc_foundation
SendEventReloadSearchIndexes	SearchIndex	bc_foundation
UnassignSearchIndexFeature	SearchIndex	bc_foundation
UnloadSearchIndex	SearchIndex	bc_foundation
UpdateCustomXmlValuesObject	SearchIndex	bc_foundation
UpdateSearchIndexConfiguration	SearchIndex	bc_foundation
WebAppAdapter	SearchIndex	bc_foundation
AddProductsToSearchIndex	SearchIndex	bc_mvc
AddSearchCondition	SearchIndex	bc_mvc
CreateCategoryFilterAttribute	SearchIndex	bc_mvc
CreateProductIndexAttributesContai	SearchIndex	bc_mvc

You can use the Enfinity Studio pipelet view to get detailed information about these pipelets. Generally, Intershop recommends to use these pipelets to build custom search pipelines with your customized or new search index.

Template Integration

The storefront integration of the ASM provides several standard templates that use the provided standard interfaces to implement the storefront functionality. These templates are located in the storefront cartridges (*sld_ch_consumer_app*, *sld_ch_adv_corporate_app*) in the template sub-directory *default/searchindex/*.

NOTE: You can override these templates by a definition inside the search module using a search engine-specific template implementation.

Table 3 lists the standard storefront templates that provide ASM functionality.

Table 3: Standard ASM storefront templates

Template	Description
SearchResult.isml	Used to process general information about the search result such as index error status or a default message if no result is found. The main part includes several other templates that depend on the search index feature definition.
ProductSearchResult List.isml	Search result list related to storefront product search indexes (SFProductSearch).
ContentSearchResult List.isml	Search result list related to storefront content search indexes (SFContentSearch).
SearchResultTabs.isml	If both product and content search are used and the two indexes show search results, this template displays two links (tabs) to switch between the product and content search results.
SearchBread Crumbs.isml	Renders the breadcrumb on top of the result list.

Template	Description
SearchIndex Paging.isml	Displays the paging bar and a search sorting box.
FilterAttributes.isml	Implements the guided search in the left navigation using three iterators provided by the search pipeline: • `StandardCategoriesAsFilterAttribute`, provides the catalog categories of the search result as an iterator of filter attribute entries • `CategoryFilterAttributesIterator`, provides the category-specific filter attributes defined for the current category, along with the number of hits, etc. • `StandardFilterAttributesIterator`, provides the global filter attributes defined for the index

For `CategoryFilterAttributesIterator`, the correspondiong ISML code may look like this:

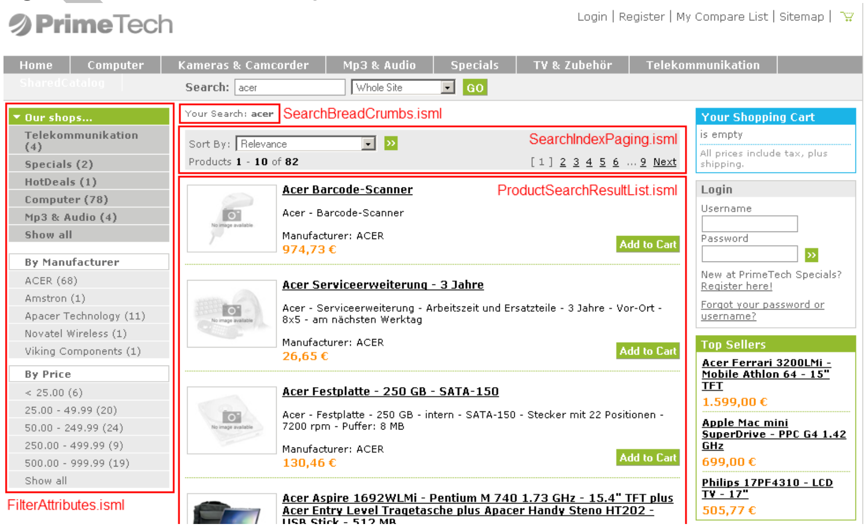
```
<isif condition="#isDefined(CategoryFilterAttributesIterator)#">
<isloop iterator="CategoryFilterAttributesIterator"
alias="catFilterAttribute">
```

For `StandardFilterAttributesIterator`, the correspondiong ISML code may look like this:

```
<isif condition="#isDefined(StandardFilterAttributesIterator)#">
<isloop iterator="StandardFilterAttributesIterator"
alias="stdFilterAttribute">
```

Figure 13 illustrates the template usage in the example storefront.

Figure 13: ASM storefront templates


PrimeTech
Login | Register | My Compare List | Sitemap |
Home | Computer | Kameras & Camcorder | Mp3 & Audio | Specials | TV & Zubehör | Telekommunikation
SharedCatalog
Search: acer | Whole Site | GO
Your Search: acer
SearchBreadCrumbs.isml
SearchIndexPaging.isml
ProductSearchResultList.isml
FilterAttributes.isml
Add to Cart
Your Shopping Cart
Login
Top Sellers

Our shops...
Telekommunikation (4)
Specials (2)
HotDeals (1)
Computer (78)
Mp3 & Audio (4)
Show all

By Manufacturer
ACER (68)
Amstron (1)
Apacer Technology (11)
Novatel Wireless (1)
Viking Components (1)

By Price
< 25.00 (6)
25.00 - 49.99 (20)
50.00 - 249.99 (24)
250.00 - 499.99 (9)
500.00 - 999.99 (19)
Show all

Sort By: Relevance
Products 1 - 10 of 82
[1] 2 3 4 5 6 ... 9 Next

Acer Barcode-Scanner
Acer - Barcode-Scanner
Manufacturer: ACER
974,73 €

Acer Serviceerweiterung - 3 Jahre
Acer - Serviceerweiterung - Arbeitszeit und Ersatzteile - 3 Jahre - Vor-Ort - 8x5 - am nächsten Werktag
Manufacturer: ACER
26,65 €

Acer Festplatte - 250 GB - SATA-150
Acer - Festplatte - 250 GB - intern - SATA-150 - Stecker mit 22 Positionen - 7200 rpm - Puffer: 8 MB
Manufacturer: ACER
130,46 €

Acer Aspire 1692WLMi - Pentium M 740 1.73 GHz - 15.4" TFT plus Acer Entry Level Tragetasche plus Apacer Handy Steno HT202 - USB Stick - 512 MB

Your Shopping Cart is empty
All prices include tax, plus shipping.

Login
Username
Password
New at PrimeTech Specials? Register here!
Forgot your password or username?

Top Sellers
Acer Ferrari 3200LMi - Mobile Athlon 64 - 15" TFT
1.599,00 €
Apple Mac mini SuperDrive - PPC G4 1.42 GHz
699,00 €
Philips 17PF4310 - LCD TV - 17"
505,77 €

In addition, an ASM implementation may provide additional templates to be included in order to support specific features provided by a given search engine. This may include:

- SearchResultErrorTemplate (no default template provided), can be used to display specific error information,
- SearchResultEmptyTemplate (no default template provided), to be displayed if there is no search result,
- SearchCampaignsTemplate (no default template provided), intended to be used for display of promotional content or additional information above the actual search result,
- SuggestSearchResultTemplate (*/SuggestResult.isml*), renders a suggest search result to display suggests in the storefront with the provided suggest script.

Data Replication

The common search indexes management includes mechanisms to support the replication of indexes. To this end, a staging group is defined in the *bc_foundation* cartridge, `FND_SearchIndexes`, which is processed by a dedicated *SearchIndexesStagingProcessor*. This processor uses a file system staging processor that is defined in the *staging.properties* file. This is the default configuration:

```
staging.processor.SearchIndexesStagingProcessor.className =
com.intershop.beehive.core.capi.staging.process.SimpleFileSystemStagingProcessor
staging.processor.SearchIndexesStagingProcessor.decorator.0 =
com.intershop.component.foundation.capi.replication.RefreshSearchIndexesDecorator
```

The file system staging processor for search indexes is configured to replicate the default location of search indexes and its configurations, *units/${UNIT}/indexes*.

The special decorator runs after a replication process to notify the Enfinity Suite 6 application servers and, depending on the implementation, the search engines about the search indexes to reload. The default *RefreshSearchIndexesDecorator* sends an `RefreshSearchIndexesEvent` processed by the event handler *SearchIndexMgr*. The default implementation will call the unload and load methods of the indexes.

The staging group `FND_SearchIndexes` is added to the data replication groups `PRODUCTS`, `PRODUCTPRICES` and `CATALOGS` in the *ProcessReplicationGroupAssignment_52* pipeline. When assigning these replication groups to a data replication task, the search indexes will also be replicated.

ASM-Related Jobs

In order to trigger an update of all search indexes in the system, there is the dedicated schedule *UpdateSearchIndexes* defined in the *SLDSystem* site. This schedule is disabled by default.

The schedule *UpdateSearchIndexes* calls the *ProcessSearchIndex* pipeline of the *sld_ch_base* cartridge to process all search indexes of a domain. This pipeline can also be parametrized in custom schedules to rebuild or update only a specified search index.

The *sld_ch_base* cartridge and its pipelines are available in the *SLDSystem* site. The schedule *UpdateSearchIndexes* must therefore be defined in this site or in a unit that belongs to this site, i.e., for the demo scenario, for example, in *PrimeTech-Site* or in *PrimeTech*.

Table 4 lists the parameters that can be specified as schedule attributes.

Table 4: ProcessSearchIndex-CompleteBuildIndex Parameters

Parameter Name	Description	Notes
SearchIndexID	The search index identifer as specified in the search index back office.	required
UnitName	The unit name, e.g., PrimeTech-PrimeTechSpecials.	required
SearchIndex ProcessAction	The action to be executed, either `Update` to update only changed objects or `Rebuild` to re-create the index.	required

ASM Implementation and Customization

This section describes advanced search module implementation with respect to the indexing process and the index types. It outlines how to extend the ASM in order to create a customized search engine integration that can be plugged into Enfinity Suite 6. Doing so assumes a sound knowledge of the Enfinity Suite 6 development processes.

The Enfinity Suite 6 ASM is able to manage search implementations for different search index types and with different search engines. Typically, an ASM implementation is deployed via an additional cartridge that implements the ASM interfaces or extends the base classes of the advanced search module and changes the standard back office at specific extension points.

A central part of the ASM is included with the *bc_foundation* cartridge in the package *com.intershop.component.foundation.searchindex*. The existing search index manager implementation provides the basic methods to handle the specific implementations of ASM.

Search Index Types

The available search index types supported by the advanced search module and their properties are controlled by the following configuration files:

- *searchindexfeatures.properties*
- *<indexTypeID>[.<engineID>].xml*
- *<indexTypeID>[.<engineID>]_<locale>.properties*

searchindexfeatures.properties

These cartridge-specific files (individually located in *<cartridge>/javasource/resources/<cartridge>/searchindex*) determine the available search index types of the system. Each cartridge that is listed after the foundation cartridge in the *cartridgelist.properties* may provide an additional *searchindexfeatures.properties* file in order to extend or customize the available search index types. Generally, these files determine the individual search index type identifiers and the Java classes used to represent the index. The search index manager of the foundation cartridge scans these files to find the available search index types as well as the assigned index classes to instantiate the index representations of the search indexes available at the server.

The class name that is specified as the property value in the *searchindexfeatures.properties* file must implement the `com.intershop.component.foundation.capi.searchindex.SearchIndex` or can extend the base implementation class `com.intershop.component.foundation.internal.searchindex.SearchIndexBaseImpl`.

The example illustrates the *searchindexfeatures.properties* of the *ac_search_factfinder* cartridge:

```
SFProductSearch =
com.intershop.adapter.search_factfinder.internal.FactfinderProductIndex
SFProductSearch.factfinder-ws =
com.intershop.adapter.search_factfinder.internal.webservice.FactfinderIndex
```

<FeatureID>[.<EngineID>].xml

The search index type definition file (individually located in *<cartridge>/javasource/resources/<cartridge>/searchindex*) determines the properties of a search index type and available static attributes that will always be added or that can be added to the index via the back office.

Intershop recommends to qualify the search index type using an additional search engine ID, e.g., *SFProductSearch.factfinder-ws.xml*. This is necessary if the system includes different implementations for the same index type. Using the search engine qualification provides the ability to have individual index implementations for different channels.

NOTE: Only one implementation for a search index type can be active for an index type at one channel.

In addition to some descriptive information and the available (static) attributes that are defined for the index type, there are some additional elements of the search index type definition that are used when the index is built or updated.

Table 5: Search index type elements

Element	Description
description	Descriptive information specifying the index' purpose, used in the back office preferences page.
searchEngineName	Search engine name, used as display name in the back office.
importHandlerClass	The class to be instantiated when the index is built or updated, the import handler takes the data of the objects that will be in the index and writes this object data to the index.
objectsResultClass	Provides a class to re-instantiate Enfinity Suite 6 objects from the search result, used in *ResolveObjectsFromSearchResult*.
objectsToIndexQueryName	Name for a query that retrieves the objects that should be part of the index; the query is used for building and updating the index.
objectsToRemoveQueryName	Name for a query that retrieves objects to be removed from the index during an update.

Element	Description
dataProviders	List of data providers used to retrieve data from Enfinity that is to be passed to the search index import handler.
defaultAttribute	Default attribute definition used when adding an attribute in the back office that specifies only a name and description. The default attribute defines the additional values that are used to add this attribute to the index.
templates	Defines search index type-specific templates that should be used for displaying results of the search index type.
attributes	The list of index attribute elements that will be added automatically at index creation (=standard attributes) and the business attributes that are available for selection in the back office.

The element type attribute specifies the initial properties of an index attribute. The element basically maps directly to the class `com.intershop.component.foundation.capi.searchindex.Attribute` (for more details, see the JavaDoc).

The standard attributes and the selected business attributes are used to create the index configuration file (*ISH-config.xml*) and the search engine-specific index configuration file (with FACT-Finder, for example, *config.xml*). The elements of the attribute can be mapped to the initial engine-specific field configuration elements via the corresponding ASM implementations.

Table 6: Attribute elements

Element	Type	Description
cluster	string	Specifies a clustering of grouped attributes.
dataProviderName	string	Maps to a data provider definition name, which defines the data provider that is responsible to get the index data for this attribute.
dataType	int	Specifies the data type (0=string, 1=int, 2=double).
description	string	Specifies a description for the attribute (used in the back office).
displayName	string	Specifies a displayname for the attribute.
filterAttribute	boolean	Specifies whether this attribute is to be used for filter group creation.

Element	Type	Description
group	string	Specifies a group name.
groupSort	int	Specifies a possible sorting.
name	string	Specifies the unique name for this attribute.
position	int	Specifies the position of a filter (currently unused).
searchable	boolean	Specifies whether the attribute can be searched.
sortable	boolean	Specifies whether the attribute is used for sorting.
standard	boolean	Specifies whether the attribute is added by default (`true` = attribute is added to the index at index creation, `false` = business attribute is available in the back office for selection).
unit	string	Specifies an optional unit.
weight	int	Optionally, can specify a weighting of the attribute (search engine specific).

<FeatureID>[.<EngineID>]_<locale>.properties

The defined attribute display names and descriptions can optionally be localized using these additional property files (located as well in *<cartridge>/javasource/resources/<cartridge>/searchindex*). These localizations are used within the back office to display the selectable attributes with respect to the user's locale.

NOTE: If there is a localization, the attributes are added to the *ISH-config.xml* with the localized names and descriptions of the respective index locale, but shown in the back office with the current backoffice user's locale.

Indexing Process

This section outlines the indexing process as performed by the advanced search module in Enfinity Suite 6.

Configuring the Index

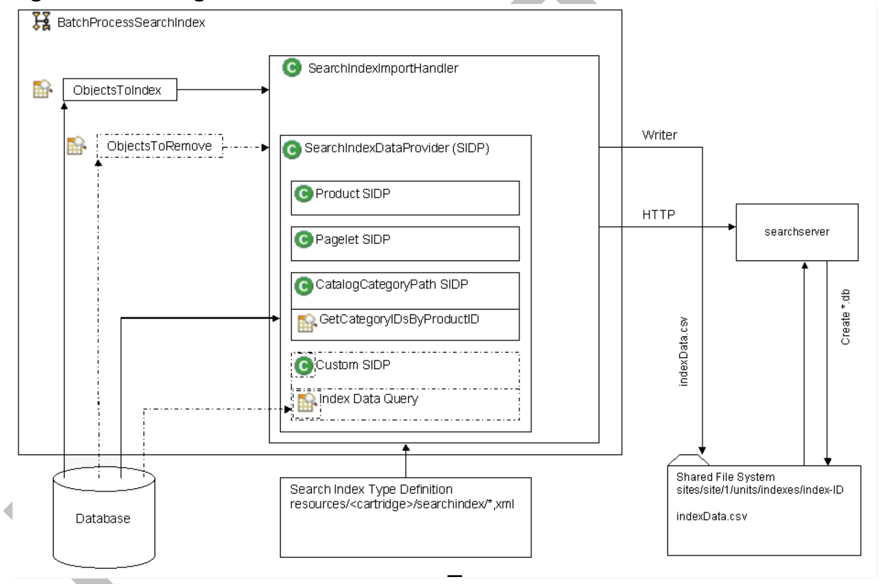
Upon configuring the index in the back office, the search index and its configuration is instantiated. If changes are applied to the configuration, the save method of the configuration is called to persist the changes.

Building the Index

Triggering an index build process via the back office starts a batch process (job). During the batch process initialization, the index import handler is instantiated. If there are any data providers defined for the search index type, they are instantiated as well and the respective attributes are registered to its providers. The query for indexing objects (`objectsToIndexQueryName`) is executed to retrieve the objects. These objects are passed to the search index import handler, with the default base implementation to each data provider. The data providers can now determine the data to be indexed and return this data to the index handler. The index handler is now responsible for writing this data to the index.

Figure 14 illustrates this process.

Figure 14: Building a search index



Updating the Index

Basically, the same process is executed upon updating the index. As opposed to the index build, the `objectsToIndex` query gets an additional parameter, `LastIndexStartTime`, which is used as a condition to determine only the new or updated objects. An additional query is executed to determine objects that need to be removed from the index. These objects are then passed to the search index import handler, which then removes them from the index.

Data Providers

Data providers are intended to retrieve the required data from Enfinity Suite 6 (or other sources) and to pass the data to the index handler. Thus, it separates the step of retrieving the data from the specific index organization and the specifics of a certain search index implementation.

Additional data providers can also be used to extend and customize already existing index types through providing additional or different data to these index types.

Since most of the Enfinity Suite 6 data reside in the database, there are some base implementations that utilize the query framework to retrieve the data and to provide some basic mapping methods, which map the retrieved query data to the search index attributes.

Example Cartridge *ac_search_mysearch*

Based on an example cartridge, the following sections describe how to create a search implementation. For the purpose of the example, we assume to implement the imaginary search engine “MySearch”.

NOTE: The example cartridge does not implement any search or filter functionality. Its mere purpose is to illustrate the basic integration steps into the ASM.

Building and installing the example cartridge requires the usual Enfinity Suite 6 development environment. Basically, this includes the following steps:

- Building the cartridge using Ant,
- Adding the cartridge *ac_search_mysearch* to the *cartridgelist.properties* in *<IS.INSTANCE.SHARE>/system/cartridges*,
- Restarting the Enfinity Suite 6 application server.

Implementing a Search Index

The *ac_search_mysearch* cartridge illustrates how to use the advanced search module for defining a new search index. Doing so involves the following steps:

- Defining a search index type implementing the SFProductSearch index type,
- Writing the search index type definition,
- Reusing existing queries to retrieve the data to be indexed.

This section also outlines which base classes or abstract classes are to be extended when implementing a new search index.

NOTE: You may use the *bc_foundation* base implementations to create a new search index type that shows up in the back office and that is configurable using the standard features such as selecting indexed attributes or adding stop words and synonyms. However, the base implementations of *bc_foundation* do not implement indexing, search and search result functionality. This must be done using search index type-specific implementations.

Search Index Type

To add a new search index feature definition, add the configuration file *searchindexfeatures.properties* to the example cartridge. In this file, define the search index type as well as the class used to represent this index.

Content of the example *searchindexfeatures.properties*:

```
#Index-type.EngineID=IndexClassName
SFProductSearch.mysearch=com.intershop.adapter.search_mysearch.Index
```

Search Index Type Definition

In the next step, describe the search index type using the search index type definition file. The following example illustrates some sections of the search index type definition. Refer to the file *SFProductSearch.mysearch.xml* provided with the example cartridge to see a complete definition with additional comments.

Once completed the type definition and the index classes (mentioned below) you can build the cartridge and restart the server. The new search index type should appear in the Search Index Preferences page of the back office available for activation. You should also be able to create a new search index configuration applying the new type as well as to add index attributes.

Search Index

The search index object represents an index that usually resides as a file (or files) on the disk. It implements methods to query the search index as well as methods to support the index lifecycle in the server (e.g., `load` or `unload`). Note that there is a single instance created for each search index defined in the system. The search index object is held in memory by the search index manager.

SearchIndexBaseImpl constitutes the base implementation of a search index (in *bc_foundation*), which is to be extended or overridden upon customization.

Search Index Configuration

The search index configuration is used to store information about, for example, the selected indexed attributes for a search index. The search index configuration is associated to a specific search index implementation and to an implementing search index object. The configuration implementation is responsible for creating and updating appropriate search index configuration files with respect to the implemented index.

NOTE: The specific configurations may derive its specific configuration files from the common configuration.

The class *SearchIndexConfigurationImpl* maintains the search index configuration stored in the *ISH-config.xml* file.

Search Index Import Handler

The search index import handler is responsible for creating and updating an index. The handler is instantiated upon starting the batch process for building and updating indexes (once for each indexing process). This handler is always search engine-specific.

AbstractSearchIndexImportHandler constitutes the abstract base class (in *bc_foundation*) that instantiates and uses data providers given in a search index type definition.

Search Result

This object is a common representation of the search result retrieved from an index. Generally, it provides methods for retrieving the actual hits from the query as well as helper methods for paging the search results. The search result is also used to resolve the hits of the current page back to objects that can be used to display the results.

In addition, the search result must provide implementations for the FilterAttribute/FilterAttributeEntry interfaces to enable the guided navigation as well as a list of SearchItem implementations to provide a breadcrumb trail of the search result in the storefront.

Data Retrieval Queries

The Enfinity Suite 6 query framework is used to obtain the store data from the database. This requires a query that delivers the objects to be indexed and, additionally, a query that retrieves the custom attributes for these objects.

In the example cartridge *ac_search_mysearch*, the already existing query *product/GeProductsToIndex* is used to get the products to be indexed. The used query is defined in the *SFProducts.mysearch.xml* as `objectsToIndexQueryName`.

To remove deleted data from the index, there is a query `objectsToRemoveQueryName`, which is executed when the index is updated. The objects retrieved from the removal query are handed over to the method `removeObjectsfromSearchIndex` of the index handler.

Data Provider Classes

Search index data providers are classes that can be plugged-in to the search indexing process to retrieve data for the search index from the Enfinity Suite 6 database or other data sources that are related to the indexed objects. By default, there are data providers to retrieve catalog data (category data and product data) and a data provider to retrieve pagelet content. Data providers are specified in the search index type definition.

NOTE: The query framework lookup for queries relies on the assigned cartridges for a site. The search cartridge is assigned automatically to the backoffice and storefront site when activating the search index feature.

There are a number of basic abstract classes that help defining and implementing additional data providers.

Table 7: Data provider classes in *bc_foundation*

Class	Description
capi.searchindex.dataprovider. AbstractSearchIndexDataProvider	Handles the registration of attributes only
internal.searchindex.dataprovider. AbstractExtractorDataProvider	Handles, in addition, the configuration and instantiation of content extractors as defined by the properties as well as methods to get and call the content extractors
internal.searchindex.dataprovider. AbstractQueryDataProvider	Base class for loading, parameterizing and executing database queries based on the query framework
mvc.internal.searchindex.dataprovider. ProductPODataProvider	Retrieves data from the product table
mvc.internal.searchindex.dataprovider. ProductAVDataProvider	Retrieves additional product data stored into custom attributes of the product
mvc.internal.searchindex.dataprovider. CatalogFilterDataProvider	Retrieves data about catalog views. Returns the catalog filter uuids for wich a product will be visible
mvc.internal.searchindex.dataprovider. CatalogCategoryPathDataProvider	Retrieves data of assigned categories for a product
mvc.internal.searchindex.dataprovider. ProductListPriceDataProvider	Retrieves product list price data
pmc.internal.searchindex. PageletPODataProvider	Retrieves data from commerce site editing pages and components

NOTE: Some of the base classes are kept in the internal package. They are, however, intended to be extended or overridden depending on the needs for new search index implementations.

Content Extractors

A content extractor is intended to additionally process a specific index attribute value. It is used, for example, to process HTML content in the pagelet configuration parameters. The basic interface for content extractors included with *bc_foundation* is *ContentExtractor*.

Content extractors can retrieve different types of content. If there are, for example, product attchements defined that contain links to documents located in the file system, a content extractor may take the file reference and retrieve content out of the attached documents for indexing. The *AbstractExtractorDataprovider* already provides an implementation to define and instantiate content extractors. These content extractors are specified in the properties of the data provider, like, for example

```
#example properties
#for a specific extractor for an attribute that contain html values
contentextractor.HTMLAttribute=html.xml
# specify the type of extractor (regex|xml|full class name)
html.xml.type=xml
#ignore content inside of these tags
html.xml.ignore=script,style
#list of tags to index separately into title attribute
html.xml.tag.target.title=h1
#add content of specified attributes from the following tags
html.xml.attributes=img:alt,img:title,meta:content
```

There are two base implementations to extract text from HTML. They can be configured by specifiying `regex` or `html`. To instantiate your own implementations, specifiy a fully qualified class name. The parameters that are defined with the given prefix `html.xml` are passed to the `init` method of your content extractor.

To execute the content extracting, call the content extractor in your implementation of the `getData` method. The following code snippet illustrates the use of the methods provided with the *AbstractExtractorDataProvider*. It assumes that there is an extractor definition for the `attributeName` and that the value used for extracting is in the `attributeValue` object. If there is no content extractor for the attribute, the value is directly passed to the index data.

```
ContentExtractor ce = getContentExtractor(attributeName);
  if(ce != null)
    {
      processContentExtractor(attributeValue, attributeName, ce, data);
    }
  else
    {
      data.put(attributeName, attributeValue);
    }
```

For another example of the property definitions of the provided extractors, refer to the *SFContentSearch.factfinder-ws.xml* located in *resources/ ac_search_factfinder/searchindex*.

Back-Office Templates

Different search index types may require different configuration pages in the back office. To ease the integration of configuration pages without changing the original back office, you can use a search index-specific include to specify different tabs for the Search Index Detail view, like

/searchindex/inc/[<FeatureID>].[<EngineID>_]_Tabs.isml. The General tab is always available since it is used to display the general information about the search index, which is required for all search index types.

The tab include uses the following lookup:

1. a fully qualified tab include
2. a defined include for a specific engine
3. an include for the specific feature.

If there is no specific tab include available, the system displays the standard tabs as defined in the template *inc/SearchIndexTabs.isml* of the *sld_ch_base* cartridge.

For the ac_search_mysearch example cartridge, you can create a template *SFProductSearch.mysearch_Tabs.isml* in searchindex/inc, as well as a corresponding pipeline *ViewMySearchConfig*.

Storefront Templates

The ASM provides standard templates that use the ASM interface to render a search result list, breadcrumbs, filter attribtues and the paging bar. The results of a search index search may, however, deliver different result objects for different index types. Therefore the ASM provides an include mechanism for templates to enable ASM implementation cartridges to replace or override the used standard templates. This mechanism can also be used for the search engine-specific customization of storefront templates.

To specify a template for use with a specific search index type, there is a section in the search index type definition file. The templates that are defined in this section are dynamically included if a specified template exists.

The currently used working template for displaying search results defines a number of template names that can be used to map to physical templates. The ASM and the default *SearchResult.isml* working template uses the following name keys to include additional templates:

Table 8: ASM template name keys

Name	Description	Default Template
WorkingTemplate	The working template used in pipeline *ViewParametricSearchBySearchIndex*.	*searchindex/SearchResult.isml*
LeftPanelWorkingTemplate	The template included to render the left panel.	*searchindex/FilterAttributes.isml*
SearchBreadCrumbsTemplate	The breadcrumb section of the search result display.	*searchindex/SearchBreadCrumbs.isml*

Name	Description	Default Template
SearchResultTabs Template	An optional tab template to display tabs for multiple search result. The default implementation can switch between a SFProductSearch and SFContentSearch search result.	*searchindex/ SearchResultTabs.isml*
SearchIndexPaging Template	The paging bar including the drop-down box for sorting.	*searchindex/ SearchIndexPaging.isml*
SearchCampaigns Template	Optional template included on top of search result.	
SearchResultList Template	The result list.	*searchindex/Product SearchResultList.isml*
SearchResultError Template	Optional error template included if there was an error during search.	
SearchResultEmpty Template	Optional template used if the search result contains no items to display in the search result list.	
SuggestSearchResult Template	The template used to render a suggest search result. Used with the pipeline *ViewParametric SearchBySearchIndex-Suggest*.	

Search Suggestions

The ASM provides the opportunity to optionally implement suggestions for product searches while typing a search term in the search input box.

To implement suggestions, the search index class needs to implement a SearchResult `getSuggestResult(String)` method, which returns the suggest items as simple Java bean classes. The default SuggestResult template will render suggest result items implementing the following bean properties:

Table 9: Suggest result properties

Property	Description
Query	The suggestion string, to be displayed as suggestion and used as search string when a user selects a suggestions.
Type	A type of suggestion, uses one of content, category, brand, unspecified.
HitCount	Specifies the number of possible hits.

Property	Description
SearchCount	Specifies a number of searches (“How often was this term used in search?”).
ImageURL	An optional image URL that can be used to display an image for a suggestion, SuggestResult.isml assumes a Content URL format.

The use of suggestions is configurable by the search index configuration. Suggestions will be used if the `isSuggestEnabled` method of the search index configuration class returns true.

The *ac_search_mysearch* example demonstrates the use of suggestions. It will return a static suggest result if a user enters “su” as query string in the search index box. A real implementation must implement suggest results for real search terms.

Search Engine Preferences

The advanced search module is prepared to support additional configuration settings that may be required for search engine integrations.

To define search index preferences for a specific search engine, you can add preference definitions to the search index preference group (`SearchIndexPreferences`). These search engine preferences are then displayed on the search index preferences page in the back office. To associate the preferences to a specific search engine, make sure that the corresponding search engine ID precedes the actual preference identifier. The following example illustrates the preference definition of the *ac_search_mysearch* example cartridge.

- ***PreferenceDefinitions.properties***

```
# mysearch example preference
mysearch.ExamplePreference = SearchIndexPreferences;3;;true;
```

- ***PreferenceDefinitionInformation.properties***

```
# example preference description
mysearch.ExamplePreference = A sample preference showing the usage of
preferences to configure search engine specific configuration values.
```