FAST INSTREAM

version 4.1

CONFIGURATION GUIDE

Copyright

Trademarks

Restricted Rights Legend

Contents

FAST Support

Website

Please visit us at:

```
http://www.fastsearch.com/
```

Contacting FAST

Fast Search & Transfer, Inc.
Cutler Lake Corporate Center
117 Kendrick Street, Suite 100
Needham, MA 02492 USA
Tel: +1 (781) 304-2400 (8:30am - 5:30pm EST)
Fax: +1 (781) 304-2410

Technical Support and Licensing Procedures

Technical Support for customers with active FAST Maintenance and Support agreements,

E-mail: `tech-support@fastsearch.com`

For obtaining FAST licenses or software, contact your FAST Account Manager or

E-mail: `customerservice@fastsearch.com`

For evaluations, contact your FAST Sales Representative or FAST Sales Engineer.

Product Training

E-mail: `fastuniversity@fastsearch.com`

Sales

E-mail: `sales@fastsearch.com`

About this Guide

Purpose of this Guide

This guide describes how to configure FAST InStream 4.1. The chapters in this guide provide task-oriented configuration procedures; the appendices contain all parameter descriptions and usage information necessary for proper configuration format.

FAST InStream is a derivative of FAST Data Search for OEM deployments.

Chapter 1 provides basic setup information that you need in order to create at least one collection using default parameters provided by FAST InStream. Creating at least one collection is necessary in order to run FAST InStream.

The remaining chapters in this Guide describe advanced setup procedures should you decide to expand or diversify your search.

Note! For information about other guides included in the FAST InStream documentation set refer to *Chapter 1 The FAST InStream Documentation Set* in the *FAST InStream Product Overview Guide*.

Audience

This guide provides information for several types of users:

- ***Managers*** and ***supervisors***, who need to understand how FAST InStream functions in order to plan the system design, configuration, implementation, and operation.
- ***System administrators*** and ***operators***, who need reference information about individual modules of FAST InStream, such as configuration options or logging.

Conventions

This guide uses the following textual conventions:

- Terminal output, contents of plaintext ASCII files will be represented using the following format:

  ```
  Answer yes to place the node in the known_hosts file.
  ```

- Terminal input from operators will be in the same but bold format:

  ```
  chmod 755 $HOME
  ```

- Input of some logic meaning, will be enclosed in <> brackets:

  ```
  setup_<OS>.tar.gz
  ```

  where `<OS>` represents a specific operating system that must be entered.

- URLs, directory paths, commands and the names of files, tags, and fields in paragraphs appear in the following format:

  The default home directory is the *C:\DataSearch* directory.

- User Interface page/window texts, buttons, and lists appear in the following format:

  Click **Next** and the **License Agreement** screen is displayed.

- *$FASTSEARCH* (UNIX) or *%FASTSEARCH%* (Windows) refer to an environment variable set to the directory where FAST InStream is installed.

Chapter 1

Basic Setup

This chapter explains the procedures you need to follow to make your first documents searchable in FAST InStream. It assumes that you:

- followed the procedures in the *Installation Guide* and have properly installed FAST InStream on your system.
- understand the basic concepts of FAST InStream as explained in the *Product Overview Guide*.

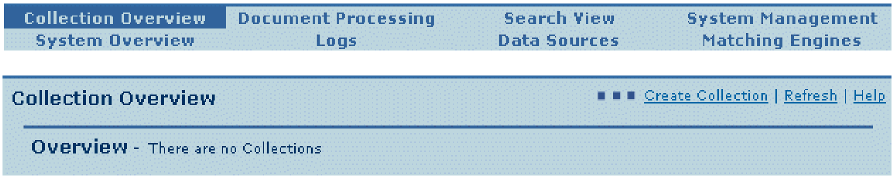
This chapter includes the following topics:

- FAST InStream Administrator Interface Navigation Bar and Symbols
- Creating a Basic Web Collection
- Edit Collection Status Boxes
- Deleting a Collection

FAST InStream Administrator Interface Navigation Bar and Symbols

After installing FAST InStream, you will find the administrator interface at address *http://yourhost:portnumber/admin*, where *portnumber* is base port + 3000. Base port is set at installation time (default *portnumber* is 16000).

The following page is displayed:



Collection Overview

Overview - There are no Collections

Navigation Bar

The navigation bar is located across the top of all pages in the FAST InStream administrator interface. The following table lists the available selections on the navigation bar.

Table 1-1 Navigation Bar Selections

Selection	Description
Collection Overview	This selection allows you to view the status of any collections (web, file, database) running on FAST InStream. This selection also allows you to create, edit or remove a collection from your search installation.
Document Processing	This selection provides an overview of pipelines and stages available for configuration. This selection displays the statistics (host, port, status, stages, pipelines) for each stage of the pipeline. You can view, create, add, edit, or remove stages of the pipeline through this selection
Search View	This selection allows you to use the FAST Query Toolkit.
System Management	This selection provides status information for any node controllers that are configured in the system. Information includes the node name, when created, general system information such as the name of the home directory and disk status, and a list of all installed modules and available modules. This selection allows you to stop or restart any or all of the installed modules as well as add or remove an available processor server or crawler. You can also stop an entire node from the System Management page.
System Overview	This selection provides a total overview of the FAST InStream system as well as status information for any modules that are configured in the system. Information includes the module name, host, port number, and status as well as the option to view detailed information for a specific module.
Logs	This selection allows you to view all system generated log files by file (module) name, category, module or collection. Log entries include the time the entry was generated, the log level, the module, host and port where the activity occurred, collection, and a text message of the activity. Archived logs can also be accessed from this selection.
Data Sources	This selection provides a list of the available data sources as well as the ability to view statistics and configuration on any associated collections.
Matching Engines	This selection provides the hostname for the Search Engine, its port number and type.

Administrator Interface Symbols

There are common icons or symbols throughout the administrator interface. The following table lists these symbols and a description of what action is taken if you click on the symbol.

Table 1-2 Administrator Interface Symbols

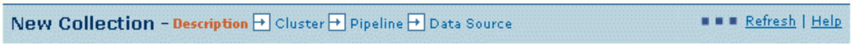
Symbol	Name	Action
next ▶	**Next**	Click on the Next icon to accept any new information and proceed to the next screen.
✖ cancel	**Cancel**	Click on the Cancel icon to clear any changes you made to the values on the screen and return to the previous screen.
✓ ok	**OK**	Click on the OK icon if you agree with and accept the information displayed on the screen.
submit ▶	**Submit**	Click on the Submit icon to accept any new information and proceed to the next screen
◀ back	**Back**	Click on the Back icon to return to the previous screen. Any information that was entered will be lost.
🔍	**View** or **Details**	Click on the View or Details icon for details of a particular entry.
📝	**Edit**	Click on the Edit icon to modify existing information in a particular entry.
🗑	**Delete**	Click on the Delete icon to remove existing information in a particular entry.
⏸	**Stop Process**	Click on the Stop Process icon to stop a module process.
↻	**Restart Process**	Click on the Restart Process icon to restart a module process.
+	**Add** or **Create**	Click on the Add or Create icon to add the related component to the existing configuration.
−	**Remove**	Click on the Remove icon to delete the related component from the existing configuration.
Refresh	**Refresh**	Click on the Refresh icon to receive screen updates about changes made to the system since screen was first displayed.
Help	**Help**	Click on the Help icon to display online help screens.

Creating a Basic Web Collection

A collection is a logical group of documents. Collections are set up in order to group documents based on selected criteria such as semantics (for example, similar types of documents) and/or document processing (for example, through pipeline configuration). This way of marking sites allows you to treat your data from different sites or sources in different ways. In order for FAST InStream to run, you need to set up at least one collection.

To get started with FAST InStream, set up a basic web collection. This will enable you to set up a content index for one or more web domains. In order to do this, you will need to indicate one or more web domain addresses, assign these to a collection and create a sample search page.

Complete this subsection to add at least one web collection using available selections and run FAST InStream. Adding a collection involves the following:

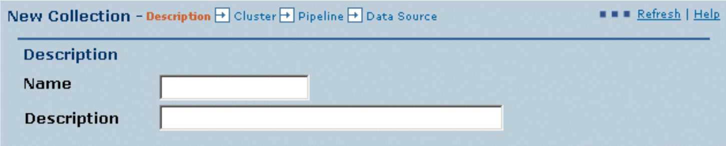


- Description (naming the new collection),
- Cluster configuration (select which cluster to use)
- Pipeline (configuring the Document Processing Pipeline)
- Data Source (identifying a data source, in this case, the web crawler).

Description

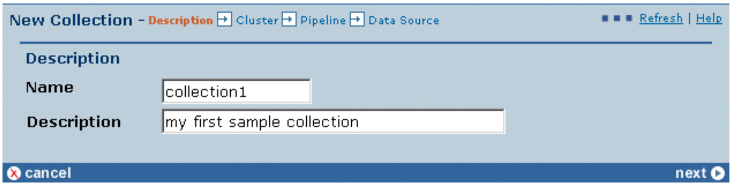
1 Select **Collection Overview** on the navigation bar and the Collection Overview screen is displayed.

2 Select **Create Collection** and the New Collection **Description** screen is displayed.



3 Enter values for the following fields to setup the new collection:

- **Name** to identify the name of the new collection. The name should be no more than 16 characters long. Make sure the name you choose contains only alphanumeric characters (a-z and 0-9) and contains no spaces. The following example creates a collection named *collection1*.
- **Description** to briefly describe the new collection. The following example describes *collection1* as *my first sample collection*.



Description

Name: collection1

Description: my first sample collection

4 Enter **Next** to accept the *collection1* Name and Description and proceed to the next screen.

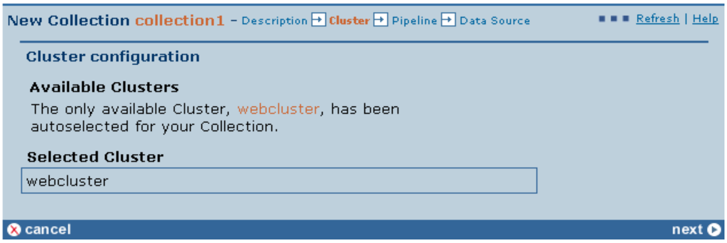
Cluster

The indexer organizes documents according to how you configure clusters to make data searchable. When you select a cluster, you select in which set of search nodes the document will reside. Each cluster has a number of collections.

Refer to the *Query Integration Guide* for details on creation of a custom search front-end.

Complete this procedure to configure clusters for the new collection.

1 After naming the collection, the New Collection *collection1*, the **Cluster Configuration** screen is displayed. The following example automatically selects *webcluster* cluster, the default cluster created during the FAST InStream installation


New Collection collection1 - Description Cluster Pipeline Data Source
Refresh | Help
cancel
next

Cluster configuration

Available Clusters

The only available Cluster, webcluster, has been autoselected for your Collection.

Selected Cluster

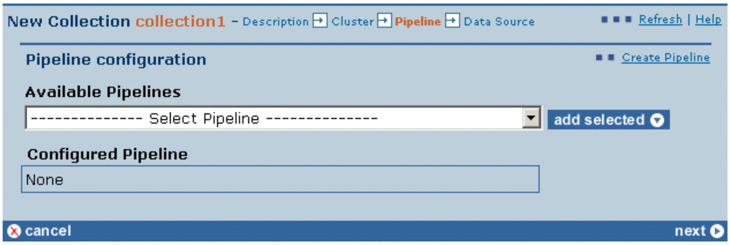
webcluster

2 Click **Next**.

Pipeline

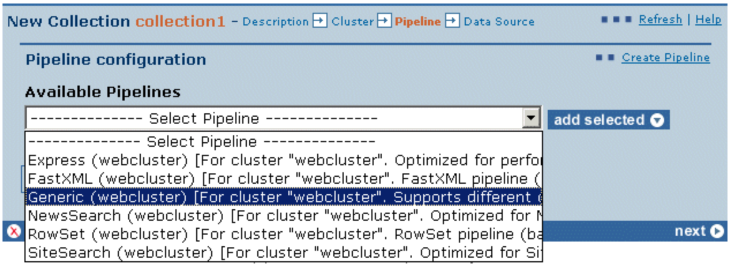
A pipeline defines a configured set of processing applied to a collection prior to indexing. Complete this procedure to configure a pipeline for the new collection using an available pipeline.

1 After selecting the cluster *webcluster*, the **Pipeline Configuration** screen is displayed.


New Collection collection1 - Description Cluster Pipeline Data Source
Refresh | Help
Pipeline configuration
Create Pipeline
Available Pipelines
-------------- Select Pipeline --------------
add selected
cancel
next

Configured Pipeline: None

2 To set up the pipeline for the new collection using an available pipeline, click on the pull down arrow in the **Available Pipelines**, and choose *Generic*, the default pipeline.


New Collection collection1 - Description Cluster Pipeline Data Source
Refresh | Help
Pipeline configuration
Create Pipeline
Available Pipelines
-------------- Select Pipeline --------------
add selected
-------------- Select Pipeline --------------
Express (webcluster) [For cluster "webcluster". Optimized for perfo
FastXML (webcluster) [For cluster "webcluster". FastXML pipeline (
Generic (webcluster) [For cluster "webcluster". Supports different
NewsSearch (webcluster) [For cluster "webcluster". Optimized for N
RowSet (webcluster) [For cluster "webcluster". RowSet pipeline (ba
SiteSearch (webcluster) [For cluster "webcluster". Optimized for Si
next

3 Click **add selected** and the configured pipeline is displayed.

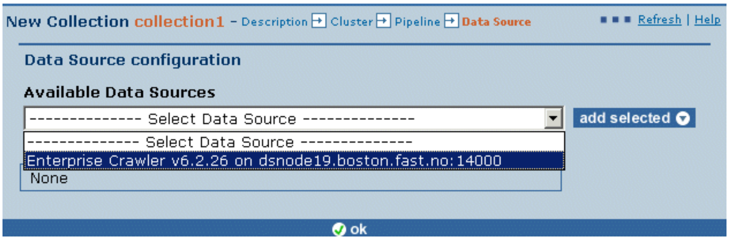
4 Click **Next**.

Data Source

Complete this procedure to identify a crawler as a data source.

1 After configuring the pipeline, the New Collection *collection1* **Data Source Configuration** screen is displayed.

2 Select the *Enterprise Crawler* from the **Available Data Sources**. This connects the collection to a web crawler for content retrieval.


New Collection collection1 – Description → Cluster → Pipeline → Data Source
Refresh | Help
add selected
ok

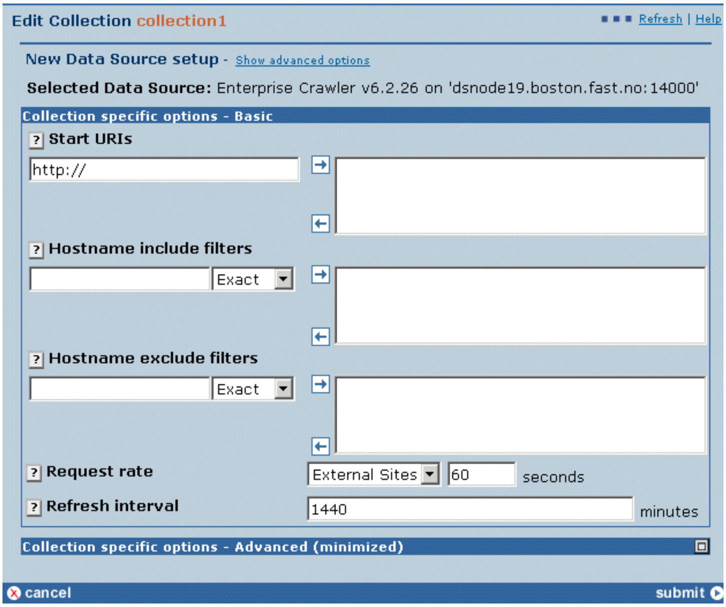
Data Source configuration

Available Data Sources

-------------- Select Data Source --------------

-------------- Select Data Source --------------
Enterprise Crawler v6.2.26 on dsnode19.boston.fast.no:14000
None

3 Click **add selected** and the Edit Collection *collection1* **New Data Source Setup** screen is displayed.



New Data Source setup -

Selected Data Source: Enterprise Crawler v6.2.26 on 'dsnode19.boston.fast.no:14000'

Collection specific options - Basic

? Start URIs: http://

? Hostname include filters: Exact

? Hostname exclude filters: Exact

? Request rate: External Sites 60 seconds

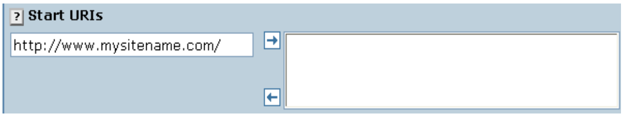
? Refresh interval: 1440 minutes

Collection specific options - Advanced (minimized)

Note! **Request rate** and **Refresh interval** define how often the crawler visits the web servers of the indicated web domain. This example assumes the default values.

4 The Start URI field allows you to create a set of URIs from which to start crawling. At least one URI (or a file containing a list of URIs) must be defined in order for the system to crawl. Make sure the URI ends with a slash (/); you cannot add the same URI twice.

Enter a start URI in the **Start URIs** box as shown. This example adds the `http://www.mysitename.com/` URI.



Start URIs

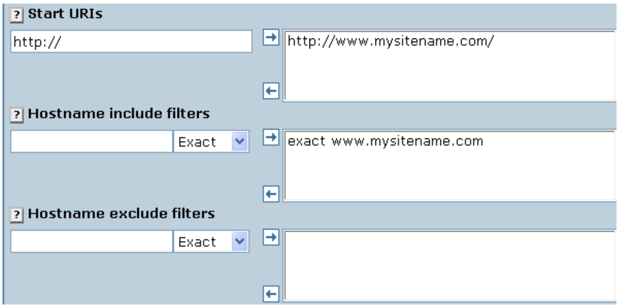
http://www.mysitename.com/

Click the → add arrow and the URI is added to the set of URIs in the text box on the right.

To remove a start URI from the list, highlight the URI and click the ← remove arrow.

At the same time, an exact hostname include filter is by default added to the list of allowed hosts in the **Hostname include filters** field. All servers in the host `www.mysitename.com` will be crawled.

After adding this URI the setup should look like this:



Start URIs

http://

http://www.mysitename.com/

Hostname include filters

Exact

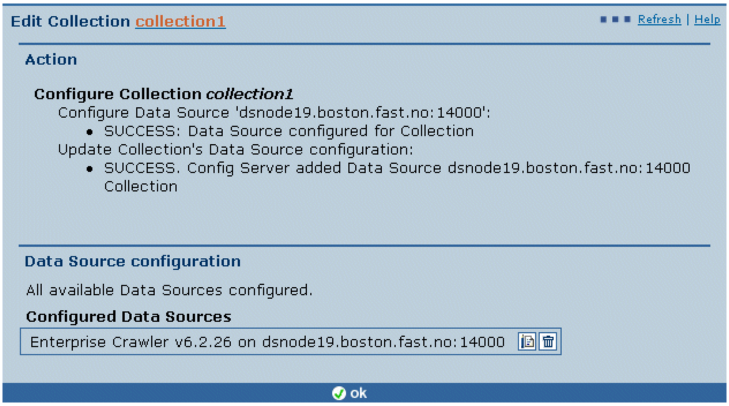
exact www.mysitename.com

Hostname exclude filters

Exact

Note! The **Hostname excludes filter** field is empty by default. The **Request rate** and the **Refresh interval** default rates are also used in this example.

5 Click **Submit** and the Edit Collection *collection1* **Action** screen is displayed. The Data Source is now installed.



Edit Collection collection1

Action

Configure Collection *collection1*
Configure Data Source 'dsnode19.boston.fast.no:14000':
- SUCCESS: Data Source configured for Collection

Update Collection's Data Source configuration:
- SUCCESS. Config Server added Data Source dsnode19.boston.fast.no:14000 Collection

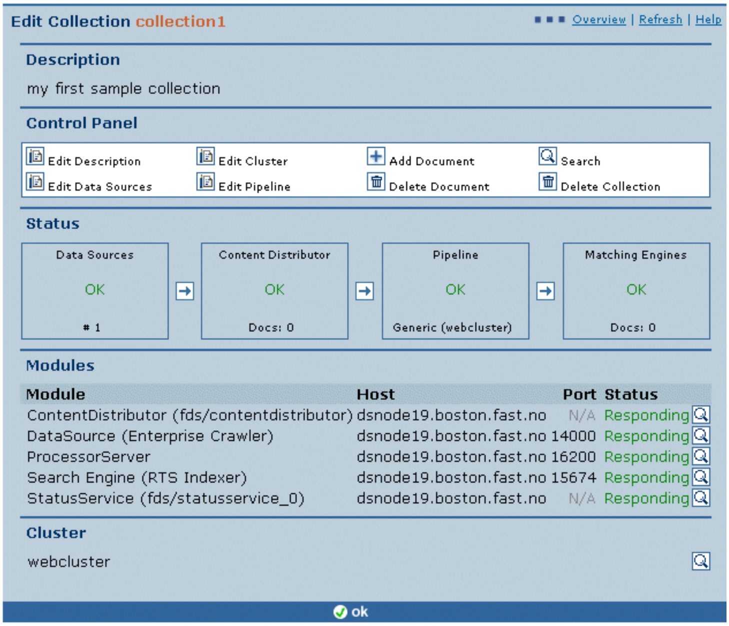
Data Source configuration

All available Data Sources configured.

Configured Data Sources

Enterprise Crawler v6.2.26 on dsnode19.boston.fast.no:14000

6 Click **ok** and the configuration is now complete. The Edit Collection *collection1* screen displays a summary of your new collection. This screen lists the name, description, pipeline, modules and clusters you have configured for *collection1*. In addition, this screen provides status information of where documents are in the process.



Edit Collection collection1

Description

my first sample collection

Control Panel

Status

Data Sources	Content Distributor	Pipeline	Matching Engines
OK	OK	OK	OK
# 1	Docs: 0	Generic (webcluster)	Docs: 0

Modules

Module	Host	Port	Status
ContentDistributor (fds/contentdistributor)	dsnode19.boston.fast.no	N/A	Responding
DataSource (Enterprise Crawler)	dsnode19.boston.fast.no	14000	Responding
ProcessorServer	dsnode19.boston.fast.no	16200	Responding
Search Engine (RTS Indexer)	dsnode19.boston.fast.no	15674	Responding
StatusService (fds/statusservice_0)	dsnode19.boston.fast.no	N/A	Responding

Cluster

webcluster

Edit Collection Status Boxes

The **Status** section in the **Edit Collection** screen displays the status of the modules in the different data flow areas of FAST InStream. The Status boxes are described in the following table.

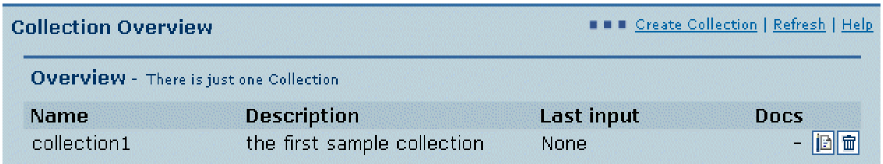
Table 1-3 Edit Collection Status Boxes

Status Box	Description
Data Sources	This box gives the status of the data sources that are relevant for the current collection. Possible messages are: **OK**: All relevant data sources are responding. **ERROR**: At least one of the relevant data sources is not responding. **N/A**: No data source is configured. If at least one data source is configured, then the bottom line contains the number of relevant data sources.
Content Distributor	This box gives the status of the Content Distributor, which distributes received content to collections and pipelines: Possible messages are: **OK**: The Content Distributor forwards information on last input and number of documents. **ERROR**: The Content Distributor forwards content that is not understandable. **N/A**: The FAST Configuration Server is unable to tell whether and where the Content Distributor is running. The bottom line contains the number of documents sent to the Content Distributor. **Last input** time is displayed when the cursor is placed over the **Docs:** document count number.
Pipeline	This box gives the status of the pipelines. Possible messages are: **OK**: A pipeline is selected for this collection. **N/A**: No pipeline is selected for this collection. The bottom line contains the name of the pipeline, if configured.

Table 1-3 Edit Collection Status Boxes

Status Box	Description
Matching Engines	This box gives the status of the Search Engines connected to the current collection. Possible messages are: **OK**: At least one Search Engine is used by the collection and is responding properly. **ERROR**: At least one of the Search Engines applied by the current collection is not responding. **N/A**: No Search Engine is applied. The bottom line **Docs:** contains the number of documents the collection has indexed in the Search Engine.

7 To return to the **Collection Overview** screen, click **ok**.

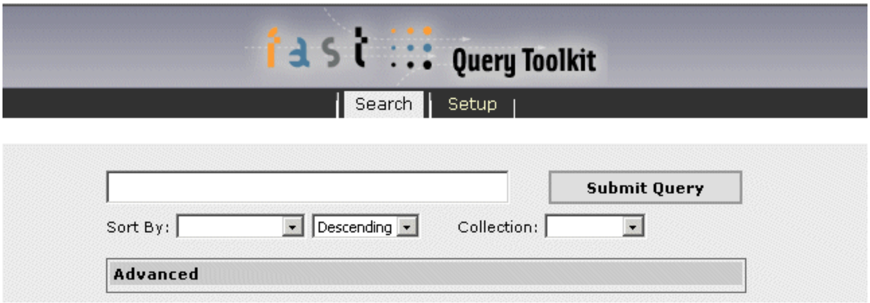


Collection Overview

Overview - There is just one Collection

Name	Description	Last input	Docs
collection1	the first sample collection	None	-

8 To access the FAST Query Toolkit, select **Search View** from the navigation bar.



Now you can perform a query against your collection.

Note that the generated search front-end does not expose all query features available in FAST InStream. Refer to the *Query Integration Guide* for details on how to create a custom search front-end.

Deleting a Collection

Before you delete a collection, note the following:

- A delete collection operation involves deletion from data sources (crawler), outputs (search engine), and may also involve operations in process server stages. The collection is not completely deleted until all these modules have reported the collection as deleted, thus it might take a while before a collection can be created again.
- Content submitted to the system has to be processed before the collection can be deleted; deleting a collection can take some time.

To delete a collection:

1. From the **Collection Overview** screen, click on the 🗑 Delete icon to the right of the collection to be deleted.
2. The **Collection Deletion** screen is displayed that asks *“Are you sure you want to delete collection <collection name>?”*. Click on the **Yes** button.
3. Click **Submit**.

Note! After starting the deletion process, do not stop or restart FAST InStream before the process has been completed. The collection is successfully deleted when the message “*Status Service signaled completion of document deletion from collection '<Collection Name>*'" is displayed.

4. Click **ok** and you are returned to the **Collection Overview** screen. Verify that the collection is now removed from the collection list.
5. Select **Logs** from the administrator interface to verify in the Logfile that the deletion has occurred.

If, after deleting a collection, you decide to create a new collection using the same name as the one that was deleted (as in the case of a *Cold Update*):

- shut down FAST InStream using *$FASTSEARCH/bin/nctrl* (UNIX) or *%FASTSEARCH%\bin\nctrl.exe* (Windows).
- In the *etc* directory, locate all files containing your original collection name.

  A UNIX example of this step: `grep myCollectionName etc/*`.

Note! Make sure to back up these files before altering them.

Go to each of these files and remove any references to your collection. If they are XML files, you need to delete the entire tag, not just the name.

- If you have used the crawler with this collection it may be necessary to delete crawled data by deleting the directories:

  $FASTSEARCH/data/crawler/dsqueues/collection.queues/<collection>

  $FASTSEARCH/data/crawler/config/<collection>

  $FASTSEARCH/data/crawler/store/<collection>

  $FASTSEARCH/var/log/crawler/PP/<collection>

  $FASTSEARCH/var/log/crawler/fetch/<collection>

  $FASTSEARCH/var/log/crawler/stats/<collection>

After you have finished, restart the system using `bin/nctrl start`. If you continue to have problems, restore your system to its normal state (by replacing edited files with the backup copies you made) and contact FAST technical support.

Chapter 2

Configuring the FAST Document Processing Engine

The FAST Document Processing Engine provides document processing on incoming documents for format conversion and document relevancy.

This chapter explains how to customize the way content is treated during document processing by configuring the Document Processing Engine. This chapter includes:

- Pipeline Templates
- Creating a Custom Pipeline With Customized Stages
- Document Processing Stages Overview
- Basic Element Manipulation
- Advanced Content Processing
- Element Format Handling
- Interfacing External Document Processors
- XML Content Processing
- Custom Indexing of HTML Documents
- Modifying a Pipeline When the System is Running

Modifying document processing is an advanced feature that can be performed through the document processing screens in FAST InStream.

Note! Be careful when modifying document processing pipelines. Removing or adding processors, changing the sequence of processors and modifying processing parameters may cause undesired side effects that may be difficult to predict.

Each pipeline template consists of a set of fixed document processing stages and a set of configurable document processing stages.

The Document Processing view within the FAST InStream Administration Interface includes a **Normal** (default) and **Advanced** edit mode. The **Normal** edit mode allows you to create custom pipelines based on templates, create certain selected custom stages and use these stages in the custom pipelines. This mode also allows you to reorder custom stages within custom pipelines.

Note! The **Advanced** mode allows you to modify, add or remove all available document processing stages. Changing, re-ordering or removing other pipeline stages than those exposed in the **Normal** edit mode may have undesired side-effects and should not be performed without contacting FAST Professional Services.

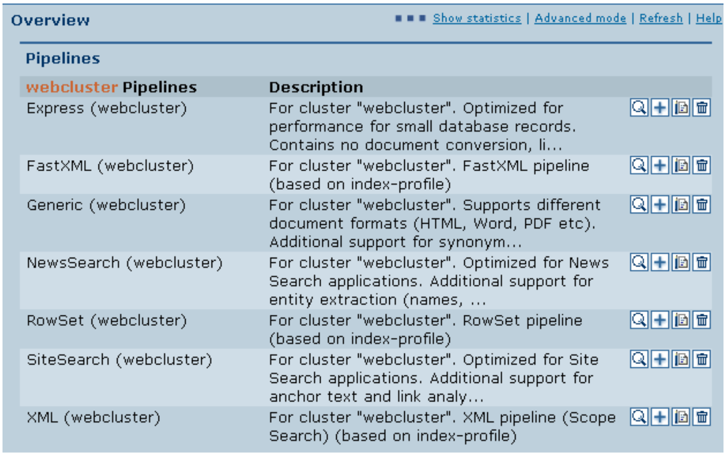
Pipeline Templates

When defining a collection you need to select a document processing pipeline as a basis for your collection. You will have two main choices:

- Select one of the available pipeline templates that corresponds to your Search Engine Cluster, also known as *Search Service* in the Installer. Search Service is provided by a Search Engine Cluster, which in turn consists of a set of search engine rows and columns. The characteristics of each pipeline template are described in more detail in the following subsections.
- Create a custom document processing pipeline based on one of the available pipeline templates that corresponds to your Search Engine Cluster. This is further described in *Creating a Custom Pipeline With Customized Stages* on page 28.

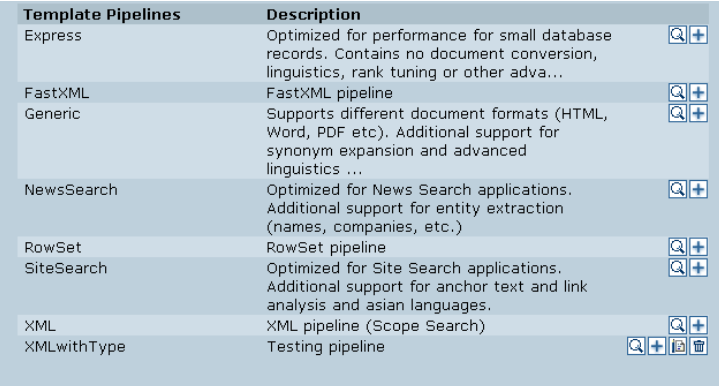
Note! Certain field updates in the index profile implies automatic changes of the default Document Processing Pipeline Templates. This includes field type setting (datetime, float, double) and changing the field attributes tokenize, lemmatize and vectorize. All templates except the *Express* template are subject to such automatic configuration.

The following screens list the default *webcluster* pipelines and pipeline templates; subsections that follow describe the pipeline templates.



Pipelines

webcluster Pipelines	Description
Express (webcluster)	For cluster "webcluster". Optimized for performance for small database records. Contains no document conversion, li...
FastXML (webcluster)	For cluster "webcluster". FastXML pipeline (based on index-profile)
Generic (webcluster)	For cluster "webcluster". Supports different document formats (HTML, Word, PDF etc). Additional support for synonym...
NewsSearch (webcluster)	For cluster "webcluster". Optimized for News Search applications. Additional support for entity extraction (names, ...
RowSet (webcluster)	For cluster "webcluster". RowSet pipeline (based on index-profile)
SiteSearch (webcluster)	For cluster "webcluster". Optimized for Site Search applications. Additional support for anchor text and link analy...
XML (webcluster)	For cluster "webcluster". XML pipeline (Scope Search) (based on index-profile)



Template Pipelines	Description
Express	Optimized for performance for small database records. Contains no document conversion, linguistics, rank tuning or other adva...
FastXML	FastXML pipeline
Generic	Supports different document formats (HTML, Word, PDF etc). Additional support for synonym expansion and advanced linguistics ...
NewsSearch	Optimized for News Search applications. Additional support for entity extraction (names, companies, etc.)
RowSet	RowSet pipeline
SiteSearch	Optimized for Site Search applications. Additional support for anchor text and link analysis and asian languages.
XML	XML pipeline (Scope Search)
XMLwithType	Testing pipeline

Web/Document Pipeline Templates

These pipelines support web content (HTML) and different external document formats such as Word or PDF. The pipelines are able to detect and convert 225+ document formats (refer to the *Product Overview Guide* for a list of supported documents and formats) to a similar element format that can be processed further on as for HTML documents.

The difference between the available pipeline templates are the level of relevance functionality included.

Below is listed a number of use cases where one of these pipeline templates should be used:

- Indexing unstructured content extracted from web servers and file servers using the FAST Enterprise Crawler, FAST File Traverser or any client application using the Content API. The same FAST InStream collection may contain different types of content, such as web pages, word processing documents, various presentation formats and spread sheets.
- Indexing content extracted from a commercial Content Management System (CMS) such as Documentum or Microsoft CMS. FAST provides dedicated Content Connectors for this purpose. In this case the content source will provide the actual document such as PDF and Word, accompanied with metadata from the CMS system.
- Indexing database content that also may include document attachments.

The available pipeline templates are:

- *Generic* - Web content and documents
- *SiteSearch* - Site Search applications
- *NewsSearch* - News Search applications

Generic

This is a generic pipeline template for web content (HTML) and different document formats such as Word or PDF. Refer to *Product Overview Guide* for a list of supported documents and formats. This template also supports synonym expansion and advanced linguistics features such as lemmatization.

Pipeline pre-processing

The following document processing is performed in the pre-processing part of the document processing pipeline. Any custom document processing stages may operate on the elements created by the processing described below.

- Retrieve external document. If the *data* element is set, the external document is retrieved from this element. Alternatively the external document is retrieved from the URL supplied in the *getpath* element.
- Parse HTML or external document (PDF, Word).
    - The HTML parser extracts visible content and initialize the *body* and *title* elements. It also extracts metadata from HTML header and place those in corresponding meta.* elements.
    - The document converter recognizes document types by MIME type and extracts textual content and properties-type metadata from the documents. Extracted content and metadata will appear in the same elements as for HTML content.
- Initialize *url* and URL related elements.
- Set *size* element to the size of the external document.

Customizable document processing

The following configurable document processing is performed after the initial document processing stages as described above.

- *TeaserGenerator:* This document processing stage creates a query independent document summary in the *teaser* element. The stage can be customized with respect to size of the generated teaser text string.

Additional custom document processing stages may be added in this part of the processing pipeline.

Pipeline post-processing

The following document processing is performed in the post-processing part of the document processing pipeline. Any custom document processing stages will be included prior to these document processing stages:

- Perform tokenization as defined in index profile.
- Perform lemmatization (support for query match with grammatical forms of words) as defined in index profile.
- Perform vectorization (preparing for Find Similar and Unsupervised Clustering) as defined in index profile.
- Convert and process date/time elements as defined in the index profile.

- Perform rank tuning processing for Business Manager’s Control Panel (BMCP).
- Perform element/field type mapping and conversion according to the index profile field configuration.
- Submit processed document for indexing.

SiteSearch

This pipeline template is optimized for Site Search applications. Compared to the *Generic* pipeline template this template adds support for authority based ranking.

Pipeline pre-processing

Refer to the pipeline template description for *Generic* on page 19.

Customizable document processing

Refer to the pipeline template description for *Generic* on page 19.

Pipeline post-processing

The pipeline post-processing is equal to the *Generic* pipeline template, but also includes support for the following functionality:

- *Anchor text extraction*. Anchor text is the textual component of the hyperlinks to this page. Either the content of text links, or text in the *alt* attribute of image links. Anchor texts associated with a web document will be indexed separately. Complete or partial query match in anchor text will imply rank boost according to the *authority* setting in the *rank-profile* of the index profile.
- Supports anchor text and link analysis and Asian languages.

NewsSearch

This pipeline template is optimized for News Search applications. Compared to the *SiteSearch* pipeline template this template adds support for *entity extraction* (names, companies).

Pipeline pre-processing

The pre-processing is equal to the *Generic* pipeline template, but also includes support for the following functionality:

- Extract common entities from documents based on dictionaries supplied with FAST InStream. This includes geographical entities, person names and company names.

The names for *geographical entities* mainly consist of English and German names for cities and countries. For very popular cities, such as *Rome*, the name in the language of the respective country may also for example, be given as *Roma*.

The *person names* consist of first and last names, for example, *John Smith*.

The *company names* include names for the most popular companies world wide and smaller companies in USA and Germany.

The complete list of entities consists of approximately 1.5 million entries. The selection of the entities was done in such a manner as to give the best search results for general news articles written in English.

However, different entities might be more suitable for a particular application, especially if the documents are in another language or refer to a certain geographical region. In this case, contact FAST Professional Services to find out whether a customized version of the list of entities is available.

The extracted content may be used as basis for Dynamic Drill-Down. The entities are extracted to the following elements:

  - Geographical entities mapped to the element *locations*
  - Person names mapped to the element *personnames*
  - Company names mapped to the element *companies*

- Extract e-mail, host and US location entities from content based on regular expression analysis:
  - e-mail addresses mapped to the element *emails*
  - e-mail addresses (formatted as Email/<state>/<city>) mapped to the element *taxonomy*
  - US locations (formatted as Location/<domain>/<name>) mapped to the element *taxonomy*
  - Host names (e.g. *www.mysitename.com*) mapped to the element *host*

Customizable document processing

Refer to the pipeline template description for *Generic* on page 19.

Pipeline post-processing

Refer to the pipeline template description for *SiteSearch* on page 21.

Database/XML Pipeline Templates

These pipelines support content retrieved from relational databases and XML. The content is either received as individual elements or XML from a Database Connector or the Content API.

An XML document is passed in the *data* element, and the processing pipelines will extract individual elements from the XML.

The difference between the available pipeline templates are the level of relevance functionality included.

Note! If your content also includes external documents as attachment, it may be more convenient to use one of the Web/document pipeline templates as a basis for your collections.

The available pipeline templates are:

- *Express* - for small database records
- *FastXML* - for API and FastXML formatted content
- *RowSet* - for RowSet XML formatted content

Express

This is a simple pipeline template optimized for performance for small database records. It contains no document conversion, linguistics, rank tuning or other advanced features. This pipeline template can be used as a basis for:

- Structured data submitted directly via the API
- Content retrieved by a database connector
- Custom XML formats submitted via the API or the file traverser

Pipeline pre-processing

The following document processing is performed in the pre-processing part of the document processing pipeline. Any custom document processing stages may operate on the elements created by the processing described below.

- Initialize *url* element (set equal to the Document ID unless explicit set by the content source)

Customizable document processing

Custom document processing stages may be added in this part of the processing pipeline, and will be executed after the initial document processing stages as described above.

The custom processing may operate on the elements as submitted by the content source.

Pipeline post-processing

The following document processing is performed in the post-processing part of the document processing pipeline. Any custom document processing stages will be included prior to these document processing stages:

- Process date/time elements for freshness boost as defined in index profile
- Perform element/field type mapping and conversion according to the index profile field configuration
- Submit processed document for indexing

XML

This pipeline template accepts XML content that is mapped to scopes within the FAST InStream index. Refer to section *XML Content Processing* on page 47 for more details.

Pipeline pre-processing

The following document processing is performed in the pre-processing part of the document processing pipeline. Any custom document processing stages may operate on the elements created by the processing described below.

- If the *data* element is present, it is assumed to contain FastXML and will be extracted to individual elements.
- Set *size* element to the size of the XML document

Customizable document processing

The following configurable document processing is performed after the initial document processing stages as described above.

- *XMLParser:* Parse XML to a DOM tree. Refer to section *XMLParser* on page 47 for details.
- *XMLExtract:* Extract relevant part of XML to a document title. This is a custom stage based on the *XMLMapper* stage, and is included as an example. In most cases you will have to customize this for your application needs. Refer to section *XMLMapper* on page 54 for details.

- *XMLScopifier:* Map the DOM tree to a scope structure. Refer to section *Scopifier(<clustername>* on page 47 for details.

Additional custom document processing stages may be added in this part of the processing pipeline.

Pipeline post-processing

The following document processing is performed in the post-processing part of the document processing pipeline. Any custom document processing stages will be included prior to these document processing stages:

- Perform tokenization if defined in index profile
- Perform lemmatization (support for query match with grammatical forms of words) if defined in index profile. May be applicable for unstructured text elements
- Perform vectorization (preparing for Find Similar and Unsupervised Clustering) if defined in index profile
- Convert and process date/time elements if defined in the index profile
- Perform element/field type mapping and conversion according to the index profile field configuration
- Submit processed document for indexing

FastXML

This pipeline template accepts XML content formatted according to the FastXML syntax.

This pipeline template can also be used when submitting structured data via the Content API and you are building the document elements directly using the API document factory methods. In this case the *FastXML* parsing stage will do nothing, as the document elements are already created by the API. In this case the difference between *Express* and this pipeline template is the support for optional linguistics processing on the content.

Pipeline pre-processing

The following document processing is performed in the pre-processing part of the document processing pipeline. Any custom document processing stages may operate on the elements created by the processing described below.

- Initialize *url* element (set equal to the Document ID unless explicit set by the content source)
- Set *size* element to the size of the XML document

Customizable document processing

Custom document processing stages may be added in this part of the processing pipeline, and will be executed after the initial document processing stages as described above.

The stage *FastXMLReaderData* extracts FastXML from the *data* element into individual elements. This stage may be removed if you do not use FastXML.

The custom processing will operate on the elements as submitted by the content source (either as individual elements or elements extracted from the FastXML content).

Pipeline post-processing

The following document processing is performed in the post-processing part of the document processing pipeline. Any custom document processing stages will be included prior to these document processing stages:

- Perform tokenization if defined in index profile.
- Perform lemmatization (support for query match with grammatical forms of words) if defined in index profile. May be applicable for unstructured text elements.
- Perform vectorization (preparing for Find Similar and Unsupervised Clustering) if defined in index profile.
- Convert and process date/time elements if defined in the index profile.
- Perform rank tuning processing for Business Manager’s Control Panel (BMCP).
- Perform element/field type mapping and conversion according to the index profile field configuration.
- Submit processed document for indexing.

RowSet

This pipeline template accepts XML documents formatted according to RowSet XML format, such as database content, using a built-in XSL style sheet. Each row is mapped to a corresponding element within the document processing pipeline.

Pipeline pre-processing

The following document processing is performed in the pre-processing part of the document processing pipeline. Any custom document processing stages may operate on the elements created by the processing described below.

- If the *data* element is present, it is assumed to contain RowSet XML and will be extracted to individual elements.

- Initialize *url* element (set equal to the Document ID unless explicit set by the content source).
- Set *size* element to the size of the XML document.

Customizable document processing

With this pipeline you can add optional custom processing stages that will be applied after the XML processing/parsing (those processing stages may operate on the elements as submitted by the content source either as individual elements or elements defined within the RowSet XML content).

Note! The supplied style sheet for RowSet transformation includes XSL for transforming the <ROWSET> element (which contains multiple rows in the RowSet XML). Note that this part of the style sheet is not used in FAST InStream, as the File Traverser parses the main XML structure and submits individual rows as documents to FAST InStream.

Pipeline post-processing

Refer to the pipeline template description for *FastXML* on page 25.

Creating a Custom Pipeline With Customized Stages

Complete this procedure if you want to create a Custom Pipeline based on one of the available document processing pipeline templates.

The characteristics of the available document processing pipeline templates are described in *Pipeline Templates* on page 17. This section describes the pre-processing and post-processing performed within the pipeline. Custom stages may be added as shown in Figure 2-1, using the *FastXML* pipeline template as an example.

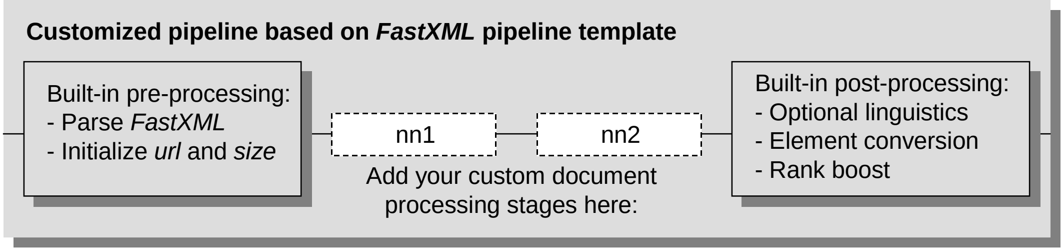

Customized pipeline based on FastXML pipeline template
Built-in pre-processing:
- Parse FastXML
- Initialize url and size
nn1
nn2
Add your custom document processing stages here:
Built-in post-processing:
- Optional linguistics
- Element conversion
- Rank boost

Figure 2-1 Customizing *FastXML* pipeline

Note! Certain field updates in the index profile imply automatic changes of the default Document Processing Pipeline Templates. This includes field type setting (datetime, float, double) and changing the field attributes tokenize, lemmatize and vectorize. It is important that you name your custom pipeline according to the following description. Otherwise the automatic update will not take place when changing the index profile later.

Creating a Customized Document Processing Stage

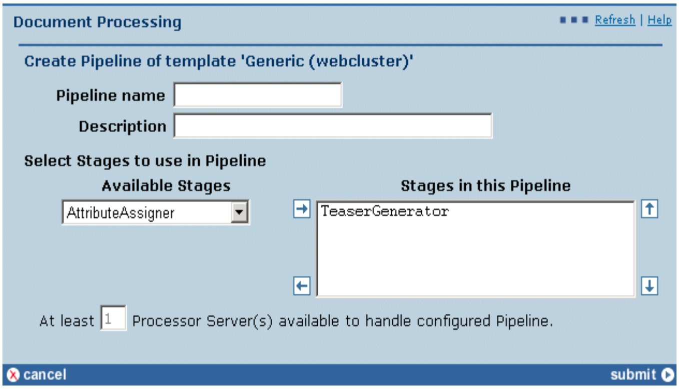
First you need to create one or more Custom Document Processing Stages that you want to add into your pipeline.

Creating a Custom Pipeline

Now you must create a custom pipeline that includes the custom stages you have created.

1 Select the **Document Processing** screen, and locate the section associated with the Search Engine Cluster where you want to use the custom pipeline.

2 From the **Document Processing** screen, click on the ⊞ **Create** button for the pipeline you want to modify. Select the pipeline template to be modified. Pipelines are named using the format *<Template name> (<clustername>)*. This example selects the *Generic (webcluster)* pipeline template.



Document Processing

Create Pipeline of template 'Generic (webcluster)'

Pipeline name

Description

Select Stages to use in Pipeline

Available Stages: AttributeAssigner

Stages in this Pipeline: TeaserGenerator

At least 1 Processor Server(s) available to handle configured Pipeline.

a Enter the new **Pipeline Name**. The name must be defined as follows:

<Your pipeline name> (<clustername>)

Example: If you want to name your pipeline *MyPipeline*, and the pipeline will be associated with the cluster *webcluster*, the name should be:

MyPipeline (webcluster)

Note! Make sure that you format the custom pipeline name correctly. There should be a single space before the parenthesis.
You must use this naming convention to ensure that the pipeline is correctly updated when changing the index profile configuration later on.

b Enter a brief **Description**.

c The screen shows the available stages for the *Generic* pipeline as well as the stages already configured. If you want to replace an existing stage (the *TeaserGenerator*), you need to remove the stage to be modified first. To do this, highlight the stage and click the ← delete button.

d Select the newly created custom stage from the **Available Stages** selection box. Highlight the stage that you created and add it to the pipeline by clicking the → add button. This stage is added to the bottom of the list.

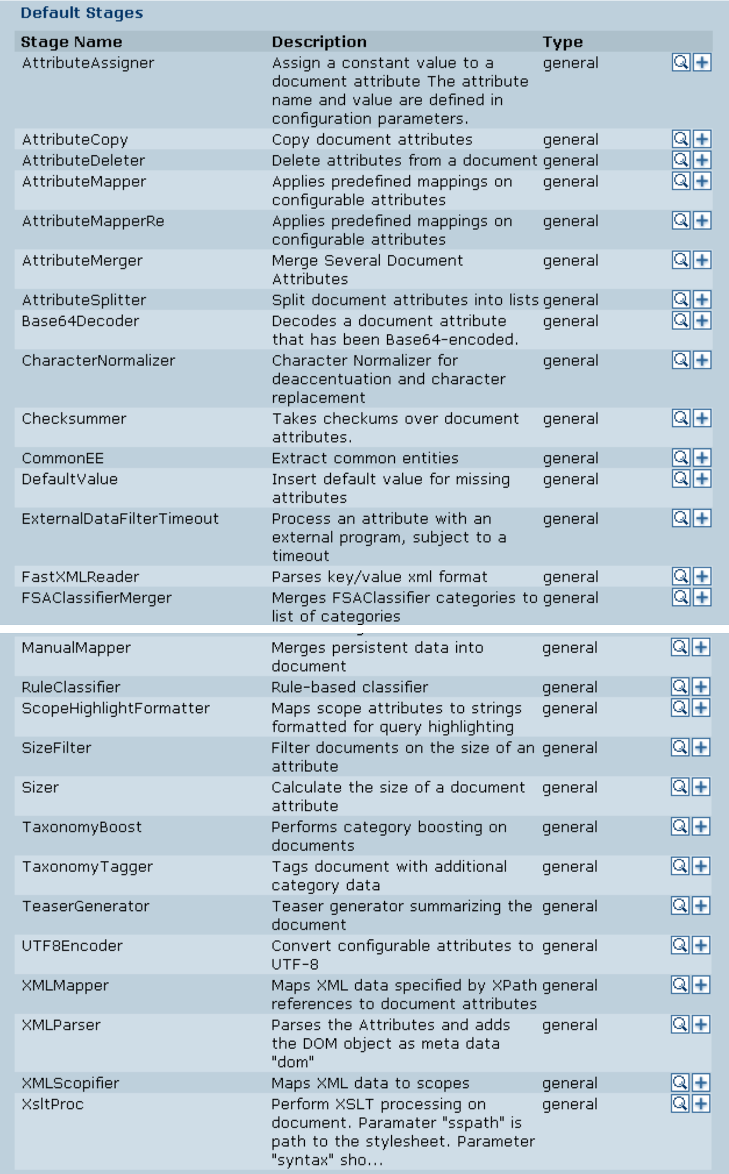
e If you add multiple custom stages that are dependant on each other (use/update the same elements) you must ensure that the stages are added in the correct order. Use the up/down arrows to place the custom stages in the correct order.

f The **Processor Server(s)** displays the number of servers that will be able to handle the pipeline.

3 Upon completion, click **Submit**.

Document Processing Stages Overview

The following screen displays the default document processors supplied with FAST InStream as listed in the **Normal mode** view of the FAST InStream Administrator interface **Document Processing** screen. These stages may be customized from the FAST InStream administrator interface in order to manipulate document elements (attributes).



Default Stages

Stage Name	Description	Type	
AttributeAssigner	Assign a constant value to a document attribute The attribute name and value are defined in configuration parameters.	general	🔍 +
AttributeCopy	Copy document attributes	general	🔍 +
AttributeDeleter	Delete attributes from a document	general	🔍 +
AttributeMapper	Applies predefined mappings on configurable attributes	general	🔍 +
AttributeMapperRe	Applies predefined mappings on configurable attributes	general	🔍 +
AttributeMerger	Merge Several Document Attributes	general	🔍 +
AttributeSplitter	Split document attributes into lists	general	🔍 +
Base64Decoder	Decodes a document attribute that has been Base64-encoded.	general	🔍 +
CharacterNormalizer	Character Normalizer for deaccentuation and character replacement	general	🔍 +
Checksummer	Takes checkums over document attributes.	general	🔍 +
CommonEE	Extract common entities	general	🔍 +
DefaultValue	Insert default value for missing attributes	general	🔍 +
ExternalDataFilterTimeout	Process an attribute with an external program, subject to a timeout	general	🔍 +
FastXMLReader	Parses key/value xml format	general	🔍 +
FSAClassifierMerger	Merges FSAClassifier categories to list of categories	general	🔍 +
ManualMapper	Merges persistent data into document	general	🔍 +
RuleClassifier	Rule-based classifier	general	🔍 +
ScopeHighlightFormatter	Maps scope attributes to strings formatted for query highlighting	general	🔍 +
SizeFilter	Filter documents on the size of an attribute	general	🔍 +
Sizer	Calculate the size of a document attribute	general	🔍 +
TaxonomyBoost	Performs category boosting on documents	general	🔍 +
TaxonomyTagger	Tags document with additional category data	general	🔍 +
TeaserGenerator	Teaser generator summarizing the document	general	🔍 +
UTF8Encoder	Convert configurable attributes to UTF-8	general	🔍 +
XMLMapper	Maps XML data specified by XPath references to document attributes	general	🔍 +
XMLParser	Parses the Attributes and adds the DOM object as meta data "dom"	general	🔍 +
XMLScopifier	Maps XML data to scopes	general	🔍 +
XsltProc	Perform XSLT processing on document. Paramater "sspath" is path to the stylesheet. Parameter "syntax" sho...	general	🔍 +

Make sure that the index profile is updated accordingly when adding new elements through custom stages.

The actual procedure for modifying a stage is described in the *Creating a Custom Pipeline With Customized Stages* on page 28.

Refer to Chapter 5 *Configuring the Taxonomy Toolkit*, section *Document Processing Stages for Taxonomy* for information about the *ManualMapper, RuleClassifier*, *Taxonomy-Boost* and *TaxonomyTagger* stages.

Basic Element Manipulation

The following processing stages enable you to perform basic operations on elements such as copy, split, merge and assign values. These stages does not perform any analysis of the actual element content.

All operations are performed on textual elements.

The following stages are available:

- *AttributeAssigner* - Assigns a constant value to a document element
- *AttributeCopy* - Copies document elements
- *AttributeDeleter* - Deletes elements from a document
- *AttributeMerger* - Merges several document elements
- *AttributeSplitter* - Splits document elements into lists
- *DefaultValue* - Inserts default values for missing elements
- *Checksummer* - Creates checksums for given document elements
- *SizeFilter* - Filters documents based on the size of an element
- *BodySizeFilter* - A pre-defined custom stage based on *SizeFilter*
- *Sizer* - Calculates the size of a document element

AttributeAssigner

Assign a constant string value to a document element. Existing element values are overwritten.

Table 2-1 *AttributeAssigner* Parameter Values

Parameter	Description
Attribute	Name of document element
Value	Value that will be assigned to the element (text string)

AttributeCopy

Copy document elements to other elements.

The document element named by the Input configuration parameter is copied to the element named by the Output parameter. Any existing value for the output element will be overwritten. If the input element does not exist, the copy operation is ignored.

Table 2-2 *AttributeCopy* Parameter Values

Parameter	**Description**
Input	Name of input document element (copy from)
Attributes	Several elements can be copied using the *Attributes* parameter. It has the form "a:b c:d", where a is copied to b and c is copied to d. Any number of such mappings can be configured. The element copy operations defined using the *Attributes* parameter are performed first, then the element copy from *Input* to *Output*
Output	Name of output document element (copy to)

AttributeDeleter

Delete elements from a document. This stage may be used to delete large elements after being processed in order to reduce memory usage by subsequent stages.

Table 2-3 *AttributeDeleter* Parameter Values

Parameter	**Description**
Attributes	Space separated list of element names. The named elements are deleted from the document, ignoring missing elements

AttributeMerger

Merge Several Document Elements. The Input document elements are merged together and placed in the Output element. The values of the input elements are merged together in the specified order and are separated with a single space.

Table 2-4 *AttributeMerger* Parameter Values

Parameter	**Description**
Input	Space-separated list of input text elements. The *Input* elements are concatenated and placed in the *Output* element

Table 2-4 *AttributeMerger* Parameter Values

Parameter	Description
Output	Name of the output element. The values of the individual input elements are merged together in the specified order and are separated with a single space

AttributeSplitter

Split document elements into lists. This enables conversion of text string separated by a configurable delimiter character to a list of strings. The list element can then be assigned to a text field with multiple string support in the index. For such a text field, you can apply Dynamic Drill-Down on string values, and also perform exact boundary match queries against the strings.

The index profile also supports mapping of multiple strings that is separated with a configurable delimiter character. Using this stage may provide more flexibility as individual stages may be configured with different delimiter characters.

The text from the input elements will be split at each separator character and copied to the output element as a list. The output element will be formatted as a Python list (Python is the programming language that is basis for the document processing framework).

Table 2-5 *AttributeSplitter* Parameter Values

Parameter	Description
Attributes	A space separated list of element pairs where each pair has the format: input:output[:type] *input* is the source element (text string) *output* is the destination element. *input* and *output* may be the same element. In this case the element itself is converted. *type* is an optional data type conversion indicator. *type* can be *int*, *long*, *float* or *str*. Default type is *str* (string). Using type conversion is for advanced usage and is only applicable when used in association with custom created document processors that expects a list of values of a given type. The stage will perform an independent element mapping for each indicated element pair.
Separator	Separator character that separates the individual strings in the input element(s). *Default: ;*

Example

```
Attributes       duplicates:duplicates
Separator        ;
```

This stage will convert the *duplicates* text element containing a number of strings separated with ‘;’ to a list of strings (in this case still in the same *duplicates* element).

DefaultValue

Insert default values for missing elements. This stage operates on text elements.

Check that all elements indicated in the *Attributes* configuration parameter have a value. Missing or empty elements are assigned the text string as indicated in the *Value* configuration parameter.

Table 2-6 *DefaultValue* Parameter Values

Parameter	Description
Attributes	A space separated list of elements
Value	Default value that will be assigned to non-existing or empty elements.

Checksummer

Creates checksums for given document elements. An MD5 checksum is created based on the input element content and assigned to the output element as a text string.

This stage is useful for more efficient dynamic duplicate removal based on large text fields. The dynamic duplicate removal feature enables removal of perceived duplicates within a result set based on identical value for given fields. This is very expensive to perform on large text fields. The alternative is to create a new element containing the checksum of the original element, and assign this to a separate field. This field can then be used as a basis for the duplicate removal.

Table 2-7 *Checksummer* Parameter Values

Parameter	Description
Attributes	A space separated list of element pairs where each pair has the format: input:output *input* is the source element (text string) *output* is the destination element (text string) that will contain the MD5 check sum.

SizeFilter

This document processing stage allows you to filter documents on the size, in bytes, of an element (*Input*). The processor checks the size of the *Input* element with respect to minimum (*MinLimit*) and maximum (*MaxLimit*) limits. If the size of the element is smaller or larger than the respective limits, then the document is discarded. Negative limits are ignored. If the input element does not exist, then the document passes.

Table 2-8 *SizeFilter* Parameter Values

Parameter	Description
Input	The name of the document input element to be filtered by size. *Default: data* (If this field is empty, then the document passes).
MinLimit	The minimum size in bytes for the document to pass. Valid Values: any non-negative number is the number of bytes. A negative number means the limit is not used. *Default: -1*
MaxLimit	The maximum size in bytes for the document to pass. Valid Values: any non-negative number is the number of bytes. A negative number means the limit is not used. *Default: -1*

BodySizeFilter

This stage is equal to the *BodySize* stage except that the input element is by default set to *body*.

Sizer

Calculate the size of a document element.

Table 2-9 *Sizer* Parameter Values

Parameter	Description
Input	Input element (text)
Output	Output element (text). The size of the input element is assigned to this output element

Advanced Content Processing

The following processing stages enable you to perform operations on elements or assign new elements based on analysis of the content itself.

All operations are performed on textual elements.

The following stages are available:

- *AttributeMapper* - Performs a conditional assignment of a target element based on the full content of a given source element
- *AttributeMapperRe* - Performs a conditional assignment of a target element based on the result of given regular expressions applied to a given source element
- *TeaserGenerator* - Creates a static (query independent) document summary (teaser)

All pipeline templates except *Express* also includes support for optional linguistic processing on given elements. This includes:

- Tokenization. Configurable tokenization on given elements/fields can be defined in the index profile. Refer to the *tokenize* attribute described in Table D-3 of *Index Profile Elements*, section *Field Sub-Elements and Attributes*.
- Lemmatization. Configurable lemmatization (grammatical normalization) on given elements/fields can be defined in the index profile. Refer to the *lemmatize* attribute described in Table D-3 of *Index Profile Elements,* section *Field Sub-Elements and Attributes*.
- Entity Extraction. Refer to *NewsSearch* on page 21 for a description on how to apply this functionality using the *NewsSearch* pipeline template.
- Creating document similarity vectors. Configurable vectorization on given elements/fields can be defined in the index profile. Refer to the *vectorize* attribute described in Table D-4 of *Index Profile Elements,* section *vectorize Sub-element*. You may also configure certain dictionaries for the vectorization process. Refer to *Configuring Similarity Vector Creation* on page 143.

AttributeMapper

This document processing stage performs a conditional assignment of a target element based on the full content of a given source element. It enables a custom mapping of elements from one element to another element. This to/from mapping requires a match with the entire input element, which can be a word or a string. Mapping parameters are stored in an external XML configuration file.

This mapping requires three parameters: the name of the element to perform the match on (*FromAttribute*), the name of the element to assign a value to (*ToAttribute*), and the file that has all the conditions and assignment values (*MappingFileUrl*).

The *AttributeMapper* looks up the value in the *FromAttribute* and matches it against the condition(s) in the XML configuration file (*from*). When the values match, the assignment value in the configuration file (*to*) is used and assigned to the *ToAttribute*.Table 2-10 describes these parameters.

If the output element indicated is already set in the document processing pipeline, then it will not be overwritten by the *AttributeMapper.*

One simple application of this document processor is *mapping from URL to category.* This is an alternative to using the *ManualMapper* provided by the Taxonomy Toolkit, and may provide a similar result. Another example is mapping from the value of a given element such as a product name, to a unique integer that in turn may be used for the *Field Collapsing*.

Refer to *AttributeMapperRe* on page 39 for details on error handling and update considerations.

Table 2-10 *AttributeMapper* Parameter Values

Parameter	**Description**
ToAttribute	This value identifies the name of the element for the conditional value assignment. *Example: taxonomy (this assigns category information as a result)* *Default:* none

MappingFileUrl	This value is the name of the mapping configuration file. The configuration file must be located on the FAST Configuration Server in the directory:
	UNIX: *$FASTSEARCH/etc/config_data/AttributeMapper/*
	Windows: *%FASTSEARCH%\etc\config_data\AttributeMapper\*
	A path/name specification may also be used, where the path must be relative to the directory mentioned above.
	The mapping configuration file is an XML file which contains an `<attribute-mapping>` element containing several `<map>` elements with a `from` and a `to` attribute. For example:
	`<attribute-mapping>`
	`  <map from="penguin.com" to="zoo"/>`
	`  <map from="penguin.com" to="birds"/>`
	`</attribute-mapping>`
	Example: AttributeMapperConfig.xml
	(which identifies the configuration file to be $FASTSEARCH/etc/config_data/AttributeMapper/AttributeMapperConfig.xml)
	Default: none

FromAttribute	This value identifies the name of the element to perform the match on (according to the <map> statements in the configuration file) *Example: taxonomy* *Default:* none

AttributeMapperRe

This document processing stage performs a conditional assignment of a target element based on the result of given regular expressions applied to a given source element.

Each regular expression is applied to the input document element. If the regular expression (*from*) matches the text (*to*), then the output document element is appended. Mapping parameters are stored in an external XML configuration file.

This mapping requires four parameters: the name of the element to perform the regular expression match on (*FromAttribute*), the name of the element to assign a value to (*ToAttribute*), the file that has all the conditions and assignment values (*MappingFileUrl*), and a delimiter character to be used between multiple output values to separate multiple category values.

If the output element indicated is already set in the document processing pipeline, then it will not be overwritten by the *AttributeMapperRe* processor.

One simple application of this document processor is *mapping from URL to category*. This is an alternative to using the rule-based mapping (*RuleClassifier*) provided by the Taxonomy Toolkit. Another example is regular expression based mapping from the value of a given element (for example, product code series, URL sub-domains) to a unique integer that in turn may be used for *Field Collapsing*.

The *AttributeMapperRe* processor can also handle simple mappings as defined for *AttributeMapper*, as long as the from text does not contain any regular expression characters not escaped using ‘\’.

Table 2-11 *AttributeMapperRe* Parameter Values

Parameter	**Description**
ToAttribute	This value identifies the name of the element for the conditional value assignment. *Example: taxonomy (this assigns category information as a result)* *Default:* none
Separator	If more than one value is present in the source element for this string field (multi-value field), then the individual values should be separated by this character. This is used by the navigators for dynamic drill-down and for boundary match queries on multi-value string fields. *Example: #* *(‘#’ is used when assigning category info, as this is the defined way to separate categories in the ‘taxonomy’ field)* *Default: “ ”*

Table 2-11   *AttributeMapperRe* Parameter Values

Parameter	Description

MappingFileUrl	This value is the name of the mapping configuration file. The configuration file must be located on the FAST Configuration Server in the directory:
	UNIX: *$FASTSEARCH/etc/config_data/AttributeMapperRe/*
	Windows: *%FASTSEARCH%\etc\config_data\AttributeMapperRe\*
	A path/name specification may also be used, where the path must be relative to the directory mentioned above.
	The mapping configuration file is an XML file which contains an `<attribute-mapping>` element containing several `<map>` elements with a `from` and a `to` attribute. For example:
	`<attribute-mapping>`
	` <map from=".*/a/path/.*" to="topcategory/subcategory1"/>`
	` <map from=".*\.mycomp.com" to="topcategory/subcategory2"/>`
	`</attribute-mapping>`
	Example: AttributeMapperConfig.xml
	(which identifies the configuration file to be $FASTSEARCH/etc/config_data/AttributeMapperRe/AttributeMapperConfig.xml)
	Default: none

FromAttribute	This value identifies the name of the element used to match the regular expression. (regular expression format) *Example: url* *Default:* none

Note! *AttributeMapperRe* does not support default value assignment (if no regular expression match). A way to facilitate this is to add a *DefaultValue* stage after the *AttributeMapperRe* stage.

Errors in the XML configuration file will lead to an error message of the following type:

```
'Creating processor "<name of custom stage>" failed: SAXParseException:
<unknown>:3:12: not well-formed (invalid token)'
```

- This error message indicates an error on line 3, column 12 in the XML file.
- This error message will appear in the system log when creating the custom stage.

You must restart all document processing engines in your system (**System Management** tab) if you change the XML configuration file without changing document processing pipelines or stages. Refer to *Modifying a Pipeline When the System is Running* on page 62 for details on a safe restart of the document processing engines.

Refer to Chapter 6 *Configuring Linguistic Processing*, section *Re-Processing the Documents After Configuration Change* for details on how to re-process documents after a document processing configuration change.

TeaserGenerator

This document processing stage configures the minimum and maximum lengths of a teaser or document summary.

Note that it may be more relevant to use the dynamic teaser in the search result. Using the default index profile, the dynamic teaser is created from the body of the document at query result processing, trying to detect the section of the document that provides the best match for the query. If a dynamic teaser can be produced, the static teaser is discarded. If the match is outside the document body, the dynamic teaser cannot be created. In this case the static teaser created by the *TeaserGenerator* is returned instead. Refer to the *Dynamic Teasers* subsection in Chapter 5 *Index Profile Management* for details.

Table 2-12 *TeaserGenerator* Parameter Values

Parameter	**Description**
MaxLen	This parameter defines the maximum length of a teaser. A teaser is chopped on a word boundary if the text to be used for the teaser exceeds this maximum length size. *Default: 300*
MinLen	This parameter defines the minimum length of a teaser. This means that the generator will wait until a section of the document has this as a minimum length before this part is used as the teaser. *Default: 75*

Element Format Handling

The following stages apply the standard format conversion on given elements:

- *UTF8Encoder* - Converts configurable elements to UTF-8
- *Base64Decoder* - Decodes a document element (attribute) that has been Base64-encoded.

UTF8Encoder

Convert configurable elements to UTF-8 character encoding. This stage may be used if you are using one of the Database/XML pipelines and your content is not according to UTF-8. FAST InStream requires that elements are encoded using UTF-8.

The encoding per source element is retrieved from one of the following sources in priority order and should be coded according to ISO 8859:

1. The 'encoding' metadata for the document element. This is for advanced use only, normally only used when coding your own document processors.
2. If not set use the element named in the *encodingattribute* configuration parameter.
3. If not set use the fallback encoding in the *encoding* configuration parameter.

Table 2-13 *UTF8Encoder* Parameter Values

Parameter	Description
attributes	A space separated list of element pairs where each pair has the format: input:output *input* is the source element (text string) *output* is the destination element that will contain the format converted element. Existing content of the destination elements will be overwritten. Element that is not yet defined in the pipeline will be ignored.
encodingattribute	Name of element that indicates the encoding of the source element.
encoding	Fallback encoding for the source element if the element specified by the *encodingattribute* parameter is not defined or empty.

Base64Decoder

This document processing stage decodes a document element (attribute) that has been Base64-encoded. This may be used by certain Content Connectors (consult the connector documentation for details) and when passing Base64 encoded content in XML. Such an element may be an attachment in a database record.

Table 2-14 *Base64Decoder* Parameter Values

Parameter	Description
src	This is the name of source element (attribute) that is base64 encoded. *Default: base64data*
dst	This the name of the document element (attribute) that will contain the converted result. This parameter should be set to the *data* element if the content is to be processed by one of the standard pipelines. This is typically the case if the element contained an external binary coded document (PDF, word, etc.) that is to be processed an indexed. *Default: data*

Interfacing External Document Processors

The *ExternalDataFilterTimeout* stage enables a document processing pipeline to access external resources outside FAST InStream for document processing of elements.

ExternalDataFilterTimeout

This document processing stage invokes an external program that processes the content of an element and assigns the returned value string to another element. In this way you can create a simple custom processor and assign a timeout for the external *stdin/stdout* program.

The external program is executed for each document, using the element named in the *Input* parameter value as input, and placing the program output in the element named in the *Output* parameter value. The external program must complete within the number of seconds configured in the *Timeout* parameter value or the program and pipeline are terminated.

Errors written to *stderr* will be picked up and logged if the process exits with an unsuccessful (non-zero) exit code.

Table 2-15 *ExternalDataFilterTimeout* Parameter Values

Parameter	**Description**
Input	This parameter determines which document input element is used as input to the external program. *Default: data*

Table 2-15 *ExternalDataFilterTimeout* Parameter Values

Parameter	**Description**
Command	Input to the external program will be provided through a temporary file on disk. Similarly, output must be written to a temporary file which will be read by the *ExternalDataFilterTimeout* stage and placed in the attribute named in the *Output* parameter. The formatting codes %(input)s and %(output)s must be present in this value, and will be substituted with the names of the input and output temporary files. If the external program acts like a standard UNIX shell application, reading input from *stdin* and writing output to *stdout*, then standard UNIX shell input and output redirection characters can be used to channel data as necessary. If the external program requires a temporary working directory, one will be created and passed to the program if the %(tmpdir)s formatting code is present in the *Command* configuration parameter. *Example 1: Replace all occurrences of "foo" with "bar" using the standard UNIX tool "sed". Shows input from file and output to stdout.* *sed -e 's|foo|bar|' %(input)s > %(output)s* *Example 2: Sort all lines alphabetically using the standard UNIX tool "sort". Shows temporary directory usage and input/output from files.* *sort -T %(tmpdir)s -o %(output)s %(input)s* *Example 3: Translate all lowercase characters to uppercase using the standard UNIX tool "tr". Shows how to redirect input and output for a pure stdin/stdout application.* *tr [:lower:] [:upper:] < %(input)s > %(output)s* *Default:* none (This value must be set manually.)
Timeout	This parameter determines the maximum number of seconds that the external program is allowed to run. If the timeout is exceeded, the program and pipeline are terminated. If the pipeline is terminated, then the current document is dropped and a WARNING is logged as a DocID failure. No other documents are affected and only the document that timed out is discarded. *Valid values: 1 to 1800 seconds* *Default: 300 seconds*
Output	This parameter determines which document attribute that the external program output is placed in. The *Output* attribute can be set to the same as the *Input* attribute if you want to replace the contents. *Default: data*

XML Content Processing

The following stages perform configurable parsing of XML documents. In order to use these stages you should create a custom document processing pipeline based on one of the *Express, XML* or *FastXML* templates.

XMLParser

The XMLParser stage performs a mapping from XML contained in the ‘data’ element to a DOM tree that can be processed by the XMLScopifier to create a scope field.

Table 2-16 *XMLParser* Parameter Values

Parameter	Description
Attributes	Input element containing XML. The Content API and File Traverser will by default use the data document element to carry the XML for this pipeline. *Default: xml*

Scopifier(<clustername>

The *Scopifier(<clustername>)* stage performs a configurable mapping from an XML DOM tree as generated by *XMLParser* to a Scope structure that can be indexed.

This stage is automatically configured from the Index Profile configuration of the scope field(s). This is indicated by the name of the stage. In the XML pipeline for the default Search Cluster (named *webcluster*) the stage will have the name *Scopifier(webcluster)*.

Note! Be careful when changing the configuration of this stage, as certain parameters are automatically configured from the Index Profile. See further details in Table 2-17.

Table 2-17 *Scopifier(<clustername>)* Parameter Values

Parameter	Description
TypeAttribute	This parameter configures the parsing of the XML, and is not set from the Index Profile. If set, indicates the name of an optional XML attribute that can be used for explicit typing of XML elements. Allowed type values for the specified attribute are: *string, int32, float, double,* and *datetime*. Other type values may be used as defined with the *TypeMap* configuration parameter (see below). *Valid values:* Any valid XML attribute name. *Default: empty*
TypeMap	This parameter configures the parsing of the XML, and is not set from the Index Profile. If *TypeAttribute* is used, *TypeMap* can be used in order to map application specific data type notations to FAST InStream data type names (*string, datetime, int32, float, double*). If *TypeMap* is used, you can only use the type values defined in *from* below in the XML, not the built-in FAST InStream data types (string, datetime, int32, float, double). Indicated as a space separated list of *from:to* mappings. *from*: Data type name as used in the source XML. *to*: Corresponding data type name in FAST InStream *Example*: `real:float int:int32` The source XML use the attribute value *real* to indicate that the element are a floating point value that shall be mapped to the FAST type *float*. Correspondingly, *int* is mapped to *int32*. *Default: empty* - This means 1:1 mapping to FAST InStream data type names.

Table 2-17 *Scopifier(<clustername>)* Parameter Values

Parameter	Description
Mapping	This parameter configures the mapping from one or more document elements containing XML to one or more document elements containing scope structured information to be indexed in scope fields. The parameter is automatically set when updating/deploying an Index Profile, reflecting the <scope-field> setting. You should not change the parameter value unless you want to test new configuration without updating the Index Profile. Note that your changes will then be overwritten when you deploy a new Index Profile later. The parameter value is a space separated list with *input:output* mapping of input XML element to output document element that will be indexed as a scope field. *Format (each mapping entry):* *<input>[<selection>]:<output>[<result>]* <*input*>: Source document element containing a DOM tree as generated by *XMLParser*. Refer to section *XMLParser* on page 47. <*selection*>: Optional parameter defining selection criteria within the XML by use of *xpath*. You can include the xpath expression as part of the *element-name* attribute to the scope field in the Index Profile. Detailed syntax for this parameter when set in the Index Profile is described in section *Scope-field Attributes* on page 206. <*output*>: Destination document element to be indexed as a Scope Field (rootscope). <*result*>: Optional parameter for internal use only. The parameter is set when using *result="dynamic"* for the Scope Field in the Index Profile. If you are configuring multiple scope fields for the same collection the parameter will include a space separated list of mapping entries as described above. *Example*: *data:xml(result:resxml) data[//title]:xtitle(result:resxtitle)* In addition to the default scope field xml this specification will also map to a scope field named *xtitle*. Only the XML that match the xpath expression (//title) will be mapped to the *xtitle* scope field. Note that the square brackets are part of the parameter syntax, not the xpath syntax. *Default: data:xml* This means that the *data* element contains the source XML and the DOM tree structure as generated by *XMLParser*. The content of *data* is mapped to the scope field named *xml*.

The mapping from XML to scopes introduce an important distinction between *typed elements, non-typed elements* and *XML attributes*.

Non-typed XML elements

Non-typed XML elements means that there is not an attribute indicating the *type* for the element (as described in Table 2-17). In this case the XML element structure maps to a corresponding scope structure in the FAST InStream index. This means that sub-elements of a non-typed XML element will be represented as sub-scopes.

The content of a non-typed XML element is mapped to a non-typed scope within the FAST InStream index. This is equal to a scope of type *string* except for support of boundary match.

Non-typed XML elements are typically used for structuring the content into hierarchies, e.g. a book-chapter-paragraph structure. You can for example be able to search for a set of words that must appear in the same paragraph within a chapter.

The XML standard says that sub tags will be concatenated to their parent. E.g. the content of <a> in

```
<a>T<b>EST</b></a>
```

becomes ‘TEST’ (from a search perspective).

This is not the case when mapping non-typed XML elements to scope fields. In this case the sub-element (<b></b>) will split ‘TEST’ into two tokens ‘T’ and ‘EST’. Hence, you can match this content with the queries:

```
rootscope:a:and(“T”, “EST”)
rootscope:a:and(“T”, b:“EST”)
```

but not with the query:

```
rootscope:a:test
```

Sub-elements like <b></b> may also impose phrase breaks. If you index the XML:

```
<a><b>TEST</b> PROCEDURE</a>
```

you can match this with the query:

```
rootscope:a:string(“test procedure”, mode=”and”)
```

but not with the query:

```
rootscope:a:string(“test procedure”, mode=”phrase”)
```

because the mapping of non-typed XML elements to scope fields implies a phrase break token on a scope boundary. Note that this works differently for typed XML elements (see below).

Note! Boundary matching (e.g. search for a scope that starts with a given word or is exactly the sequence of words) can not be applied to the scopes resulting from non-typed XML elements.

Typed XML text (string) elements

Typed XML text elements means that there is an attribute indicating the *type* for the element (as described in Table 2-17) indicating that the element is of type *string*.

A *typed XML text element* may include non-typed XML sub-elements. In this case the resulting scope will contain the text of the element and all sub-elements. Consider the XML fragment:

```
<book>
  <p type=”string”>hello <b>world<b></p>
</book>
```

The resulting scope named *‘p’* will contain the text ‘hello world’. Note that this is different from a non-typed XML element, where <b> would have been mapped to a sub-scope. In the case of a typed text element you will also be able to query the scope content as a phrase even if the text contains formatting sub-elements as in the XML above:

```
rootscope:a:string(“hello world”, mode=”phrase”)
```

It is not allowed to have typed sub-elements to a typed XML text element. Attributes will appear as sub-scopes (refer to *Mapping of XML Attributes* on page 52).

Boundary matching (e.g. search for a scope that starts with a given word or contains exactly a given set of words) is supported for typed XML text elements.

Typed numeric XML elements

Typed numeric XML elements means that there is an attribute indicating the *type* for the element (as described in Table 2-17). The XML element type may be any of the built-in FAST InStream numeric data types (int32, float, double, datetime). The element will be mapped to a scope with the corresponding type.

A typed numeric XML element must not have XML sub-elements. It may have XML attributes.

It is also possible to map numeric XML content that is not represented as a typed XML element from the content source. This can be done by adding an XSL transformation stage in the document processing pipeline. Refer to section *XSLT* on page 57.

Note! You must know the data type for the scope you are querying. If there is a mismatch between data type in query term and scope, you will not get match, or in some cases wrong match.

Mapping of XML Attributes

An XML *attribute* will be represented as a sub-scope, where the scope name equals the attribute name with the prefix '@'. An XML *attribute* will always be represented as a scope node of type *string*.

Boundary matching (e.g. search for a scope that starts with a given word or contains exactly a given set of words) is supported for attributes.

The following attributes have special meaning:

- The *type* attribute as configured within *FAST InStream* document processing. Refer to Table 2-17.

  This attribute will impact how the associated XML element is typed, but the type attribute itself will not be indexed as a sub-scope.

- The *xml:lang* attribute controls the lemmatization of the associated element. The *xml:lang* attribute itself will not be indexed as a sub-scope.

Example

The following examples show how XML can be mapped to scopes, and how to query the content. The examples assumes that the scope field is named *xml*:

```
<book>
   ...
   <chapter>
      <num type=int32>5</id>
      <heading type=”string”>Tigger and Winnie</heading>
      <sentence>Tigger came out of the forest</sentence>
      <sentence>Winnie the Pooh was waiting for him</sentence>
   </chapter>
   ...
</book>
```

Assuming that the scope field is named ‘xml’, this will be mapped to the following scope structure:

```
book:                                                  (non-typed)
   chapter:                                            (non-typed)
      num:5                                            (type=int32)
      heading:”Chapter 5”                              (type=string)
      sentence:”Tigger came out of the forest”         (non-typed)
      sentence:”Winnie the Pooh was waiting for him”   (non-typed)
```

In this example the chapter number is mapped to a numeric scope (*int32*). You may for instance use that to search within a range of chapters.

The following alternative XML representation may also be used, where the chapter number is represented as an attribute instead:

```
<book>
   ...
   <chapter num=”5”>
      <heading type=”string”>Tigger and Winnie</heading>
      <sentence>Tigger came out of the forest</sentence>
      <sentence>Winnie the Pooh was waiting for him</sentence>
   </chapter>
   ...
<book>
```

In this case you can still search for content within a specific chapter (searching for the chapter number as a string), but you will not be able, as in the previous example, to search within a range of chapters.

XMLMapper

The XMLMapper processor is designed to fulfill the need to extract content from XML documents and map it to FAST InStream document elements.

XMLMapper may be used in a scope search setting to extract parts of the XML content to individual fields, such as a title, etc. In this case the XMLMapper stage should be used within a pipeline based on the XML pipeline template. This template includes a default custom stage based on XMLMapper that extracts a title from the XML content.

The XML document processing pipeline template includes a custom stage based on *XMLMapper* that extracts a title from XML documents. This custom stage is named *XMLExtract* and is mainly intended as an example. You will in most cases need to customize the configuration file to suit your XML content.

When using XMLMapper to map XML to non-scope fields only, use the FastXML pipeline template as a basis, and replace the FastXMLReaderData stage with the XMLMapper stage.

The processing is based on based on the *libxml2 DOM* and *XPath 1.0* interfaces. Because of this, much of the documentation exists in the form of W3C recommendations.

In order to use this stage you should create a custom document processing pipeline based on the *Express* or *FastXML* template and include the *XMLMapper* stage.

Configuration

Initial configuration of the XMLMapper processing stage defines the source document element and the configuration file.

Table 2-18 *XMLMapper* Parameter Values

Parameter	**Description**
Attribute	The name of the document element (attribute) where the mapper will look for XML content to map. You may use any element for the XML content. It is recommended to use the *data* element for this purpose. The File Traverser will output XML to the *data* element by default, and this is also the case for the dedicated XML methods in the Content API. *Default: xml*
ConfigFile	The name of the configuration file to use. This file must reside in the *$FASTSEARCH/etc/config_data/XMLMapper* directory on the FAST Configuration Server node within the FAST InStream installation. *Default:* empty

Mappings

There are three tags that allow you to define mappings from XML data into FAST InStream document attributes, *SubTree*, *Mapping* and *MappingGroup*. The following tables define these tags and describe how they can be used.

Table 2-19 *XMLMapper SubTree* Tag Values

Value	Description	
base-path	*base-path* defines what tag(s) or attribute(s) will be used as the base node for all SubTree, Mapping, and MappingGroup tags contained within this SubTree tag. This base-path is an XPath 1.0 path defining a single tag in the source XML document. ***Note!*** It is very important that this maps to only one instance of a tag within a document.	Required

Table 2-20 *XMLMapper Mapping* Tag Values

Value	Description	
attr	*attr* defines the name of the document attribute where content will be placed. Content will be placed into this field based on the mode selected in the mode tag. ***Note!*** Note: This tag is not available when a Mapping tag is within a MappingGroup tag.	Required
path	*path* defines what tag(s) or attribute(s) will be extracted. This path is an XPath 1.0 path defining a set of tags or attributes in the source XML document.	Required
sep-str	*sep-str* allows you to define the separation character that is used when the path matches more than one XML node and they must be joined together.	Single Space Character
pre-str	*pre-str* is a string that will be prepended to the beginning of the content extracted by this mapping.	Empty String
post-str	*post-str* is a string that will be appended to the end of the content extracted by this mapping.	Empty String

Table 2-20 *XMLMapper Mapping* Tag Values

Value	Description	
ignore-whitespace	*ignore-whitespace* defines how whitespace is treated when it is extracted from the XML document. The value of ignore-whitespace can be 'yes', in which case consecutive whitespace characters will replaced by a single space, or 'no' which will leave all whitespace characters in the content.	yes
mode	*mode* defines how this mapping tag will interact with data that is already in the document attribute defined by *attr*. The options for this field are 'append', which will append the content from this mapping to the document attribute, 'overwrite' which will overwrite the contents of the attribute, and 'prepend' which will prepend the content to the document attribute. ***Note!*** Note: This tag is not available when a Mapping tag is within a MappingGroup tag.	overwrite

Table 2-21 *XMLMapper MappingGroup* Tag Values

Value	Description	
attr	*attr* defines the name of the document attribute where content will be placed. Content will be placed into this field based on the mode selected in the mode tag.	Required
base-path	*base-path* defines the base tag(s) that the Mapping tags within this group will be relative to. This field is optional and if not specified, all Mapping tags within will be relative to the XML document root.	Required
sep-str	*sep-str* allows you to define the separation character that is used when the path matches more than one XML node and they must be joined together.	Single Space Character
pre-str	*pre-str* is a string that will be prepended to the beginning of the content extracted by this mapping.	Empty String
post-str	*post-str* is a string that will be appended to the end of the content extracted by this mapping.	Empty String
rec-sep-str	*rec-sep-str* allows you to define the separation character that is used when joining the contents of the Mapping tags within this group.	Single Space Character

Table 2-21 *XMLMapper MappingGroup* Tag Values

Value	Description	
rec-pre-str	*rec-pre-str* is a string that will be prepended to the beginning of the content extracted from the Mapping tags within this group.	Empty String
rec-post-str	*rec-post-str* is a string that will be appended to the end of the content extracted from the Mapping tags within this group.	Empty String
select	*select* allows you to determine which Mapping tag(s) within this group will be used when extracting content. The values for select are 'merge', which will merge the contents of the Mapping tags using the rec-sep-str, rec-pre-str, and rec-post-str parameters; 'first' which will select only the content from the first Mapping tag that matches; 'longest' which will select the content from the Mapping tag that produces the most content.	merge
mode	*mode* defines how this mapping tag will interact with data that is already in the document attribute defined by *attr*. The options for this field are 'append', which will append the content from this mapping to the document attribute, 'overwrite' which will overwrite the contents of the attribute, and 'prepend' which will prepend the content to the document attribute.	overwrite

XSLT

This document processing stage performs XSLT processing on the XML as processed by the *XMLParser* stage. This is applicable in cases where you would need to do some pre-processing of the XML prior to generating a *Scope Field*. The processing stage does not operate on the source XML, but applies modifications to the DOM tree as generated by *XMLParser*, based on the indicated style sheet.

Table 2-22 *XSLT* Parameter Values

Parameter	Description
Attribute	Indicates the source document element. *Valid values:* Any valid document element that includes XML processed by the *XMLParser* stage. *Default: data*

Table 2-22 *XSLT* Parameter Values

Parameter	Description
Stylesheet	This is the path to the XSL style sheet, relative to the following directory on the FAST InStream admin node: *$FASTSEARCH/etc/config_data/DocumentProcessor* *Default: no style sheet (empty)* *Example:* *XSLT/myXSLT.xml* In this case the style sheet must be located in *$FASTSEARCH/etc/config_data/DocumentProcessor/XSLT/myXSLT.xml*

XsltProc

This document processing stage performs XSLT processing on an XML document.

Note! This document processing stage is deprecated in this version of FAST InStream and is only included for backwards compatibility.

When using XSLTProc, select the FastXML pipeline template, replace the FastXMLReaderData with FastXMLReader and add the XSLTProc stage in front of the FastXML-Reader stage.

Note! For performance reasons it is recommended to use XMLMapper instead of XSLTProc for custom XML transformations.

The Style Sheet should transform custom XML into FastXML, which is then processed using the *FastXMLReader* stage.

Table 2-23 *XsltProc* Parameter Values

Parameter	Description
sspath	This is the path to the XSL style sheet, relative to the FAST InStream installation directory ($FASTSEARCH). An example XML style sheet that transforms RowSet XML to FastXML is provided: *$FASTSEARCH/etc/RowSetXSLT.xml* *Default: etc/parsehtml.xml*

Table 2-23 *XsltProc* Parameter Values

Parameter	Description
attrib	This the name of the document element (attribute) that contains the XML or HTML document to be transformed. This parameter should be set to the *data* element. *Default: data*
syntax	This parameter should be either *html* or *xml*. Note that HTML documents cannot use strict XML parsing. *Default: html*

Note! The style sheet needs to be distributed to every machine running the document processor.

FastXMLReader / FastXMLReaderData

This document processing stage parses elements encoded according to FastXML syntax and initializes the individual elements as defined within the FastXML body.

The default *FastXMLReader* stage reads FastXML from the *xml* element. This stage is used in association with XSLT transformation and extraction of *RowSet XM*L, as the Row Set extractor extracts XML from the *data* element and outputs this as FastXML in the *xml* element.

The *FastXMLReaderData* stage parses FAST XML from the *data* element. This is similar to *FastXMLReader* except that another source element is used.

You may customize the standard processing stage *FastXMLReader* if you want to use a non-standard (different from *data*) document element for the FastXML content. This is configured using the *XMLSource* configuration parameter.

Normally the *data* element is used to pass XML to FAST InStream. The custom document processing stage *FastXMLReaderData* is pre-configured to be used in this case.

If you use the *FastXML* pipeline template for other types of content than *FastXML*, the *FastXMLReaderData* document processing stage may be removed from the pipeline.

When using the *XSLTProc* stage, ensure that the *FastXMLReader* stage is included after this stage.

Custom Indexing of HTML Documents

FAST InStream is able to identify custom tags in HTML documents that indicate sections of the document that will not be indexed. Common menu structures can span across multiple pages and may mislead search. HTML tagging can be used to avoid this indexing of auto-generated content and can be added manually or automatically by content management or web publishing applications.

The tag format is based on the `DIV` or `SPAN` HTML attributes. The following example uses the `DIV` attribute:

```
<HTML>
...
<BODY>
...this is searchable html...
<DIV class="noindex">
...this html will not be indexed by FDS...
</DIV>
...this is searchable html...
</BODY>
</HTML>
```

Notes:

- The `<DIV>` tag and `<SPAN>` tag are standard HTML tags.
- The semantics of `"noindex"` is specific to FAST (for example, `<DIV class="noindex">` or `<SPAN class="noindex">`).
- The `<DIV>` tag may impact the rendering of HTML in a browser if used inside a paragraph. This tag should only be used to enclose structural blocks of HTML that you do not want to index, such as an HTML table or complete HTML paragraphs. The `<SPAN>` tag does not impact the rendering of HTML and may be used instead.
- This feature is supported by the web pipelines: *Generic*, *SiteSearch* and *NewsSearch*.

Modifying a Pipeline When the System is Running

If you want to modify a pipeline while FAST InStream is running, do the following:

1. All data sources must be stopped before you can safely change the pipeline for a collection. From the **System Management** screen, stop modules by clicking on the ⏸ **Stop** icon to the right of the applicable entries.
2. Check the number of documents queued by the Content Distributor. From the **System Overview** screen, locate the **ContentDistributor** entry in the Module List. Click on the 🔍 **Details** icon to the right of the **ContentDistributor** entry.
3. Verify that the value of the **Pipeline Queue Size** is equal to **0**. The size must be equal to 0 before it is safe to modify or remove any pipelines.
4. Follow the procedures in *Creating a Custom Pipeline With Customized Stages* on page 84 to modify a pipeline.
5. You may also re-process the content for one or more collections in order to re-process the documents using the new configuration. Refer to Chapter 6 *Configuring Linguistic Processing*, section *Re-Processing the Documents After Configuration Change* for details.

Chapter 3

Index Profile Features Management

The index profile defines the index structure and manages what features are enabled. This process is similar to the process of defining a schema or table structure for a database. The index profile is defined through an XML based configuration file. After editing, this configuration file is uploaded and deployed using the administrator interface of FAST InStream. The index profile identifies:

- Mapping from document processing *Elements* to searchable *Fields* in the index.
- *Field* attributes such as *type* (text/integer), *search attributes* (substring, lemmatization), *rank tuning* parameters and *result presentation*.
- Grouping of fields and creation of *result views*.

Refer to the *Product Overview Guide* for concepts and terminology related to the index profile.

Refer to Appendix C *Configuration Files and DTD* for the complete FAST InStream default index profile and DTD configuration files; refer to Appendix D *Index Profile Elements* for detailed element and attribute descriptions.

This chapter describes the features that can be configured via the index profile as well as detailed procedures on how to set up the index profile. The following features can be enabled by configuring the index profile:

- Dynamic Teasers
- Field Importance Level
- Substring Query Support
- Dynamic Duplicate Removal
- Categorization
- Result Clustering
- Proximity Rank Boost

- Dynamic Drill-Down (Navigators)
- Field-Collapsing
- Text Sorting
- Relevance Tuning Using Rank-Profile
- Relevance Tuning for Scope Fields

Default Index Profiles

The default index profiles provided with FAST InStream provide a suitable configuration for a web-based index. However, you may decide to customize the index profile to suit the particular needs of your application.

Appendix C *Configuration Files and DTD* contains the default index profile and DTD files; Appendix D *Index Profile Elements* describes all index profile elements and attributes.

Setup at Installation

The FAST InStream installation program provided you with three different choices for the index profile: *Web Profile*, *Web profile with lemmatization*, and *Custom Profile*. If you chose:

- *Web Profile*, then you chose the default index profile for a web search application that does not utilize the lemmatization feature. This selection uses the *datasearch-4.x-default.xml* file.
- *Web profile with lemmatization*, then you chose the default index profile for a web search application that utilizes an advanced linguistic feature. This selection uses the *datasearch-4.x-lemmatization.xml* file.
- *Custom Profile*, then you chose to enable a customized index profile. You must edit and save the customized index profile prior to installation.

The FAST InStream distribution includes a *datasearch-4.x-default.xml* file, a *datasearch-4.x-lemmatization.xml file* and a *datasearch-4.x-taxonomy.xml* file (this file is identical to the *default.xml* file except that it is also configured for use with the Taxonomy Toolkit). These files can be used as a basis when creating a *Custom Profile*.

Dynamic Teasers

A dynamic teaser for a document is a summary of the document that presents the sections within the documents with the most relevant query match, and highlights the query terms. The dynamic teaser is normally created on the basis of the *body* of the document.

Within the index profile a field of type *string* or a *scope-field* may be configured as *dynamic*. This implies that this particular field (typically the *body*) will be used as the basis for creating the dynamic teaser.

Dynamic Teaser Examples

`result=“dynamic”`

The following example configures the *body* field to be the basis for the dynamic teaser. The *body* field is defined by `index=“no”`. This implies that the field itself is not indexed. However, in this example, the *body* field is defined as part of a *composite-field* in the index, and will therefore be searchable within the *composite-field*.

`result=“dynamic”` implies that this field will be returned as a dynamic teaser rather than returned as such in the search result.

`fallback-ref=“teaser`

For a web index the default pipelines also create a static (not query dependent) summary of the document. This may be configured as a fallback in case it is not possible to create the dynamic teaser (e.g. the query match content in another field). The attribute *fallback-ref* can be used to indicate another field to be used as fallback.

`fallback-ref=“teaser”` implies that the *teaser* field will be the fallback:

```
<field name="body" tokenize="auto" max-result-size="1024"
   fallback-ref="teaser" result="dynamic" index="no">
```

For non-web applications it may be preferred to use another text field as the fallback, or create a custom document processor in order to create a fallback *teaser*.

`element-name="productdescription"`

In some cases it may also be desired to use the same field as basis for the dynamic teaser and as a fallback. Example: The index profile configuration below will ensure that the *productdescription* field in the result contains the dynamic teaser (if possible to create), otherwise it will contain the actual content of the *productdescription* field:

```
<field name="teaser" element-name="productdescription" index="no" />

<field name="productdescription" tokenize="auto" fallback-ref="teaser"
       result="dynamic" />
```

`fallback-ref="teaser"` indicates that the *teaser* field is used as the fallback. The definition of this extra field *teaser* is required due to the fact that it is not possible to define the field itself as fallback. You can limit the size of this fallback field if required, using the *max-result-size* attribute (in order to limit the text returned from the query).

Refer to *Appendix D Index Profile Elements* for details on index profile syntax.

Field Importance Level

This is an advanced feature that is applicable for large indices and high query rates. FAST InStream is able to compute static and dynamic rank for a large number of query hits within each query. However, if the total number of hits to any individual term in a query is very large, this may not be feasible for performance reasons. To overcome this limitation, the *Field Importance Level* support provides a means to automatically drill-down in the result set by narrowing the field scope for the query. This feature functions on a per-term, not per-query basis.

The *Field Importance Level* feature applies to *composite-fields* and utilizes the *level* attribute. The initial scope for the query matching is all fields as defined within the *composite-field*. If the number of matching entries is too large, a new attempt is applied using all the fields with *level >1*. This process is continued (*level > 2*, then *level >3*, and so on) until a manageable number of hits can be retrieved as basis for the rank computation. After narrowing the scope using *level*, the matching entries are ranked by static/dynamic rank. The total number of hits is still presented to the user, but the remaining hits (excluded using the *field importance level*) will be presented after the ranked documents.

Note! When using the *level* attribute, make sure that the *composite-field* includes *field-refs* with all levels up to the maximum level. This means that if the maximum *level* is 4, a *field-ref* with *level* equal 1, 2 and 3 must also exist in the *composite-field*.

Field Importance Level Example

Refer to the *<composite-field> Example* on page 210 for a detailed example.

Substring Query Support

FAST InStream can support substring search. This feature focuses on Asian (CJK) content which does not have a strict word based tokenization. The feature enables wildcard queries that do not take into consideration word boundaries. It may also be used on non-CJK content, but in these cases normal wildcard support is usually preferred. By defining a *field* or *composite-field* in the index profile to be a *substring* field, you can search for substrings of arbitrary lengths and at arbitrary positions inside the indexed content.

A *field* or *composite-field* definition includes an optional attribute `substring=n`. The value `n` determines usage, performance and index size, and must be in the range 0 through 31, 33 through 63.

Substring Query Example

```
<composite-field name="content" rank="yes" default="yes" substring="36">
   <field-ref … />
   …
</composite-field>
```

If `n` is in the range 1-31, all white space characters are ignored when the documents are indexed, and the search engine has no knowledge of at what character each word begins or ends. This has the effect that wildcard before and after each search term is implied; `*term`, `term*`, `*term*` and `term` are equivalent in the query string.

If `n` is equal to or greater than 33, a flag specifying that word boundaries are to be detected is set. Specifying this allows the use of the wildcard character `*` before and/or after the search term to be treated as one would normally expect.

What is the most suitable choice for `n` depends on the following factors:

- Whether white space is usually used to separate words.

  If the majority of documents to be indexed are in languages where spaces are not used to separate the words, such as Japanese and Chinese, it is normally best to strip away information about word boundaries. This is beneficial with regard to index size, and is also better when searching for phrases. Use `n` in the range 1-31 in this case.

  If the majority of the documents to be indexed are in languages where spaces are used as word separators, it is normally best to be aware of the word boundaries. Use `n` in the range 33-64 in this case.

- The size of the character set(s) for the language(s) of the documents to be indexed. This affects which part of the range `n` should be within: 1-31 or 33-64. When the number of characters is very high, use a low number within the range (such as 2 or 34). The value 34 is equivalent with the value 2, except that word boundaries are indexed for n as 34. When there are fewer characters available, use a somewhat higher n value. The higher end of the ranges is rarely if ever, needed.

A high range of values is recommended when indexing code, such as DNA sequences, 'GCTCAGTTCTGACCTCTTTAGCAACGTACA...', where very few different characters are used.

Recommended values for n are:

36	- used mainly for western language documents
2	- used mainly for Chinese documents
3	- used mainly for Japanese documents
35	- used mainly for Korean documents

Word boundaries are not detected with substring fields having an n value that is less than 33. This means that a query **ecar** would also match the text *white carpet*. This type of substring search is not always applicable for normal text documents of reasonable size, as the probability for such undesired matches across word boundaries increase with document size. For structured data and Asian language encoded data this may not be a problem.

Note that this feature is not supported when used in conjunction with the dynamic teaser and proximity features. Due to this limitation, it may in some cases be desirable to have a *composite-field* searchable both in *normal* mode and *substring* mode. In this case the *composite-field* needs to be duplicated in the index, and a query front-end will have to select between the *normal* and *substring composite-field* for the queries, depending on the use of wildcards.

This can be achieved by creating a custom document processing pipeline that copies the desired elements into new elements; creating a duplicate of all elements that is used in the *composite-field*. This can be performed using the document processor *AttributeCopy.*

For example, if you want to define a *composite-field* consisting of *name* and *address*. In the custom processing pipeline *name* is copied to *namesub* and *address* is copied to *addressub* using *AttributeCopy.* In the index profile two *composite-fields* are defined, *person* and *personsub*. *person* contains the fields *name* and *address*, and *personsub* contains the fields *namesub* and *addressub*. In this case the query front-end should direct normal queries to the *person composite-field*, and wildcard queries to the *personsub composite-field*.

Dynamic Duplicate Removal

A result-side duplicate removal filter can be applied using the *result-filter* element within the index profile *result-specification* element.

The *duplicate* filter applies to a specific field. Only documents with different values for this field will be returned for the query. If the referenced field has an empty value for a specific result and the field has a *fallback-ref* specified, then duplicate filtering will be attempted using the fallback.

The duplicate removal function will ensure that the entry with highest rank among the duplicates is returned in the result set. The duplicate removal function is supported on *text* fields only.

In the following example a duplicate removal result filter is applied on the *url* field. This ensures that only one document is returned with the same URI.

Normally such duplicate removal based on URI is performed prior to indexing. However, in certain cases the same document (the same URI) may appear in different collections within FAST InStream. As queries may be applied to selected collections only, it is not possible to detect or remove such duplications prior to indexing.

Result-Side Duplicate Removal Example

```
<result-specification>
   <result-filter name="duplicateurlremover" type="duplicate">
      <field-ref name="url"/>
   </result-filter>
   …
</result-specification>
```

Note! When you enable dynamic duplicate removal, the total hit count for the result will be an estimate of the total hit count. The estimate may be different if querying with a different value for the query parameter *offset*.

Note that this feature may be processing-expensive if there are many fields configured to be part of the result set, and the probability to find duplicates are large. It may be suitable in order to detect unique duplicates as in the example above, but a duplicate removal filter on a field such as the body is normally not recommended due to the processing overhead.

A maximum of two duplicate removal specifications are supported.

Note that FAST InStream also provides other features with related functionality:

- *Result Clustering*. This feature enables clustering of similar documents in the result set, which may enable similar end-user functionality.
- *Dynamic Drill-Down*. This feature enables grouping of results based on the value of individual fields.

Categorization

Categorization fields in FAST InStream provide index level categorization and are used for *Supervised Clustering*, *Search within category* and *Taxonomy Index*. The main category field has to be defined in the index profile using a *result-specification* element. The *categorization result-specification* element defines a category field and ensures that this field is configured properly for the *Result Clustering* and the *Taxonomy* Index.

Categorization Example

The default index profile defines the field *“taxonomy”* as the main category field:

```
<result-specification>
   <categorization name=“default” sort-by=“label”>
      <field-ref name=“taxonomy”/>
   </categorization>
   ...
</result-specification>
```

The *categorization result-specification* element assigns the field *“taxonomy”* to the built-in categorization support. A prerequisite for the Taxonomy Index to work is that a corresponding taxonomy has been created using the Taxonomy Toolkit or by defining the appropriate taxonomy XML (TaXML) files.

Refer to the *Categorization Element* on page 214 for details on index profile syntax.

Result Clustering

Result Clustering is used to cluster results according to a document similarity metric. Clustering is more dynamic than categorization in that the clusters generated are generated automatically based on the actual results. Refer to Chapter 5 *Configuring the Taxonomy Toolkit* for details on the *Result Clustering* feature.

In Result Clustering, the document similarity vector generation can be configured using the vectorize attribute (or sub-element) to the field specification.

Clustering is configured in the *result-specification* in the index profile. Note that the clustering feature is linked to the reserved *docvector* field. The *result-specification* does not contain any explicit field reference.

Result Clustering Example

```
<result-specification>
   <clustering threshold="0.40"/>
   ...
</result-specification>
```

This example defines the clustering algorithm to use the non-standard similarity threshold *0.40*.

Refer to the *Clustering Element* on page 215 for details on index profile syntax.

Note that the clustering feature can be controlled on a per query basis using the Query API parameter *CLUSTERING*.

Proximity Rank Boost

Proximity relevance implies that the distance between query terms in matching documents impacts the query results. The proximity computation contributes a portion of the overall rank of a document within a result set.

Proximity affects only queries with multiple words. The more terms that are present in the query, the higher the impact will be on the result-set.

Explicit Proximity implies using the proximity query operators NEAR and ONEAR. Refer to the *Query Integration Guide* for details.

Implicit Proximity refers to the proximity component of the rank value. *Implicit Proximity* does not change the total number of hits for a query, but impacts the relevance sorting of the results.

The basic proximity support (*Implicit*/*Explicit*) in FAST InStream is controlled via the *positions* attribute of a *composite-field*. This must be set to *positions="yes"* in order to apply the necessary indexing of position information.

Implicit Proximity ranking is controlled via the *rank-profile*; refer to *Relevance Tuning Using Rank-Profile* on page 79 for more information.

For backwards compatibility, FAST InStream also supports two proximity approximation features that can be uses instead of the standard index-side proximity support:

- *Result-side Proximity Boost* implies that proximity rank boost and explicit proximity are applied to the result-side on a limited part of the result set. Result-side proximity is controlled via the *result-proximity* element in the index profile.
- *Bigram Proximity Boost* implies that query term pairs appearing as bigrams will imply rank boost. *Bigram Proximity Boost* is controlled via the *bigram-boost* element in the index profile.

Dynamic Drill-Down (Navigators)

The index profile defines a *navigator* element to manage the *dynamic drill-down* feature.

Note! The default index profiles supplied with FAST InStream enable the Navigation attribute for some fields. This is added as an dynamic drill-down feature example. If you do not intend to use the feature, it is recommended to remove this configuration from your index profile.

Refer to *Result-proximity Element* on page 218 for details on index profile syntax.

Navigator Example

```
<field-list>
   ...
   <field name=”artist” type=”string”/>
   <field name=”length” type=”string”/>
   ...
</field-list>

<result-specification>
   <string-navigator name=”artistnavigator” display=”Artist”>
      <field-ref name=”artist”/>
   </string-navigator>

   <numeric-navigator name=”lengthnavigator” display=”Length”
     unit=”sec” resolution=”60” algorithm=”rangedivision”>
      <field-ref=”length”/>
      <range-label type="first" format="Less than %g sec” offset="1"/>
      <range-label type="middle" format="%g sec - %g sec” />
      <range-label type="last" format="More than %g sec” />
   </numeric-navigator>
</result-specification>
```

This configuration example provides *dynamic drill-down* support for the two fields named *artist* and *length*. *Artist* is configured as a *string-navigator* and the search results will in that case include a navigator section that has a drill-down modifier for each value in the field *artist*.

Length is a *numeric-navigator* and this will return a list of modifiers that each represent a range. For the field *length* one can typically get a list of buckets such as *less than 120 sec*, *120-179 sec*, *180-299 sec* and *more than 300 sec*. Since we specified a resolution of *60* for the navigator, each bucket will have a size that is a multiple of 60.

Note that there is no absolute connection between the field type specified in the *field* and the *navigator* type. It would be possible to specify a string navigator for the *length* field, but that would yield a modifier/drill-down option for each discrete value in that field. This may be useful for other integer fields like *cpu-speed* or *memory* where it is desirable to display all the possible values instead of displaying ranges like *128-255 MB* and *256-383 MB* for the *memory* field. In such a case it would probably be better to use a string navigator and display *128* and *256* as drill-down options. In this case, use the *auto* or *number* sort type for the string navigator.

Note that navigators cannot be defined for scope fields. If you want to drill down on scope content, it is required to extract relevant drill-down content from the scope search content into separate non-scope fields. For XML source, this may be done using the XMLMapper document processor.

Navigator Relevancy Performance

It is possible to limit navigators to only return the most relevant values, called *buckets*. For example, for a navigator on person names, only those names that occur most frequently are returned as navigator values. Even with such cutoffs, the system will return correct values for the total number of values even though they are not all returned.

String navigator *cutoffs* can be specified in the index profile. Available attributes in the string-navigator element are *cutoff-maxbuckets*, *cutoff-frequency*, and *cutoff-frequency-exception-minbuckets*. Refer to Table D-22 in Appendix D *Index Profile Elements* for detailed descriptions of these attributes.

Updating navigator cutoffs in the index profile does not require a *Cold Update*.

Note that all values restrict the number of navigator values returned from individual search partitions. There are usually several search partitions per column in a system. The total number of unique navigator values will be a product of the number of search engine columns and number of partitions per column.

Field-Collapsing

Field-collapsing allows a folding of results with the identical value for a given result field.This can be used in order to collapse results with given attributes.

Examples of usage are:

- *Site collapse similar* as found on popular web search engines. All results from the same domain may in this way be collapsed into one result, typically presenting the result from this domain with the highest relevance score (rank).
- *Product field collapse* or collapsing all results with the same product code in the result set.

A *site collapse* type feature requires that an ID is assigned to every document with similar origin. In an Enterprise setting this may not necessarily be the domain, but may be sub-domains or other origin information associated with the document URI. Assignment of such unique URIs may be performed using the *AttributeMapperRe* document processor.

Collapsing is specified query-time using the collapse parameter.

Refer to *Index Profile Elements,* section *Field Element and Attributes* for details on index profile syntax.

Note! This feature is not a full collapse but rather a re-sort that will show the top ranked results for each value of the collapse field before other search results.

Field-collapsing Example

```
<field-list>
   <field name="mysitenameid" type="int32" sort="no"/>
   ...
</field-list>
<result-specification>
   <field-collapsing name="sitecollapse">
      <field-ref name="mysitenameid"/>
   </field-collapsing>
   ...
</result-specification>
```

This profile defines that field-collapsing should be possible on the `"mysitenameid"` field. Only one field may be field-collapsing enabled, and only *int32* fields are supported.

Text Sorting

The default text sorting configuration files are located in *$FASTSEARCH/etc/alphasort* (UNIX) or *%FASTSEARCH%\etc\alphasort* (Windows). If you intend to create a custom sorting profile, complete the following:

1 Create a sub-directory in the indicated directory, where the sub-directory name will be the name of the sort profile.

2 Create a main configuration file named *AlphaSortMasterFile.xml* in this sub-directory. Use the default file as a basis.

3 Create a set of language specific configuration files as described below, depending on your application need. A number of sample files are supplied that can be used as a basis.

4 Create a *fullsort* sub-element to the field specification, indicating the sort profile with the attribute *profile*.

Note that the default alphanumeric sorting is based on ascii sort order. A case-insensitive sort *profile* is also available. Refer to the *Index Profile Elements*, section *fullsort Sub-element* for format details.

AlphaSortMasterFile.xml Example

```
<?xml version="1.0"?>
<!DOCTYPE masterFile SYSTEM "SortMaster1_0.dtd">
<masterFile>
   <sortValueWidth> 8 </sortValueWidth>
   <importFiles>
      <importFile>Space.xml</importFile>
      <importFile>Numbers.xml</importFile>
      <importFile>Latin.xml</importFile>
      <importFile>Arabic.xml</importFile>
      <importFile>Hebrew.xml</importFile>
      <importFile>Cyrillic.xml</importFile>
      <importFile>Greek.xml</importFile>
      <importFile>Kana.xml</importFile>
   </importFiles>
</masterFile>
```

The `<sortValueWidth>` field defines the number of bits per character (8, 16, or 32) that will be used to create the rank value for sorting text. For example, if you select 8 bits per character, then 4 characters per rank value will be allowed since the maximum length of a rank value equals 32 bits. On the other hand, 8 bits per character will only support 256 sort levels which is not enough to support sorting of text in languages such as Chinese and Korean. In this case at least 16 bits per character would be needed for sorting.

The `<importFiles>` field defines the sort order between the different alphabets and syllabaries. By default FAST InStream sorts characters from different character sets in the following order:

1 Space

2 Numbers

3 Latin

4 Arabic

5 Hebrew

6 Cyrillic

7 Greek

8 Hiragana and Katakana

The internal sort order within a character set is defined by a separate file per alphabet/syllabary. The master file only contains references to these files. The structure of the character set files are shown below using the six characters "a", "A", "b", "B", "c" and "C" as in the following example:

```
<?xml version="1.0"?>
<!DOCTYPE sortOrder SYSTEM "AlphaSort1_0.dtd">
<sortOrder>
   <sortValue>
      <unicode>
         <utf8> a </utf8>
      </unicode>
      <unicode>
         <utf8> A </utf8>
      </unicode>
    </sortValue>
    <sortValue>
      <unicode>
         <hex> 42 </hex>
      </unicode>
      <unicode>
         <hex> 62 </hex>
      </unicode>
    </sortValue>
    <sortValue>
      <unicode>
         <dec> 67 </dec>
      </unicode>
      <unicode>
         <dec> 99 </dec>
      </unicode>
    </sortValue>
</sortOrder>
```

Note the following:

- Two characters are sorted in the same manner if they are listed within the same `<sortValue>` tag (for instance ‘`a`’ and ‘`A`’ in the previous example).
- A character is identified by its UNICODE code point value (‘`B`’ for instance has UNICODE code point 0x62).
- The UNICODE code point value can be given using utf8, hex or decimal notation (if tagged accordingly).

Relevance Tuning Using Rank-Profile

This section describes how to tune relevance (rank) using the *rank-profile* sub-element within a *composite-field*. This is used to tune a set of individual ranking parameters that determine how query results can be sorted by relevance.

Refer to *rank-profile Sub-element* on page 205 for details on index profile syntax.

Refer to the chapter *Concepts of Relevancy* in the *Product Overview Guide* for more details on how these relevance tuning parameters work and an overview of other relevance tuning capabilities in FAST InStream.

Each *rank-profile* includes the following:

- Reference to a basic Rank Model (*rank-model* attribute). FAST defines a set of pre-defined rank models optimized for Site Search, News Search, etc.
- The *authority* configuration element relates to the automatic connectivity analysis provided by FAST InStream. Query match in anchor texts of links to this page/ document is used to compute an authority rank component
- The *freshness* configuration element defines to what extent document freshness will imply higher rank. The date of a document is converted to a 'freshness' parameter that reflects the age of the document from the time of a query. The age is scaled to reflect the perceived importance of age (the difference between 1 and 5 days age may reflect the same relevance difference as between 1 and 12 months age).
- The *proximity* configuration element defines to what extent the distance between query terms in the documents is taken into consideration, and applies more relevance when query terms are found close to each other. Furthermore, matching query terms that occur early in the document imply a higher ranking value, and implicit phrase match may be boosted
- The *context* configuration element allows you to modify dynamic ranking on a per field basis. Each field can be assigned a *Field Weight*, which represents the relative importance of a match within this field compared to other fields. As an example, the weights between the fields can ensure that words contained in the *title* field are more relevant than those in the *body* field.

The rank boost according to *rank-profile* is applied as follows:

- This rank boost is one of several rank components, which includes static rank and manually modified rank. The result set will then be sorted based on the resulting rank value.
- If you have defined only one rank-profile for a *composite-field*, the corresponding rank boost will automatically apply when you search within this *composite-field*.
- If multiple rank-profiles are defined for a *composite-field*, the SORT_BY query parameter must be used to select the appropriate *rank-profile* context for each query.
- If no *rank-profile* is selected for a query, the appropriate *rank-profile* for the query will be the first listed *rank-profile* in the *composite-field* selected for the query (explicit or default *composite-field*).
- If no *composite-field* is defined, only query boosting via the BMCP is supported.

Note! Tuning of these parameters must be tested on a reasonably large and representative query set for your application in order to verify expected relevance improvements.

Relevance Tuning for Scope Fields

Composite fields and rank profiles may also be used to apply relevance tuning for scope fields. This is applied as follows:

- Create a dedicated *composite-field* for this purpose. In the default index profiles this composite field is called *subscopes*. This can be used as a basis for your implementation. Ensure that you have set the attributes *rank="yes"* and *default="no"* for this composite field. Also ensure that the *query-tokenize* attribute is set to the same value as for the scope fields defined in the index profile.
- The composite field must include one or more rank profiles. The rank profile must include references to the desired scope fields (<*scope-weight*> element) within the <*context*> element.
- The composite field may include a set of field-ref's for text fields that contains superior content extracted from the scope fields. In the default index profile three fields are defined for this purpose, named *scopeboostlow, scopeboostmedium* and *scopeboosthigh*. You may configure the *XMLMapper* document processor to extract content to these document elements in order to achieve different context boost to different sub-scopes. The default pipeline template *xml* includes a configurable stage XMLExtract that performs such an extraction. Note, however, that the configuration file for this stage should only be seen as an example. You will need to define your own extraction based on your XML content.
- Select the desired rank profile using the SORT_BY query parameter.

Note that this is a special way of using a composite field. You will normally not use this composite field as a search target, it is only used in order to define rank profiles for scope queries.

If you mix terms against scope fields and non-scope fields within the same query, it is normally not possible to apply dynamic ranking to both parts of the query. Dynamic ranking will only be applied to the fields (scope fields or non-scope fields) that is referred within the <*context*> element of the used rank profile. This may, however, be solved with help from FAST Professional Services.

Updating the Index Profile

Certain index profile changes require a full re-index of the content within the affected search engine, while others only require that the new index profile be installed.

Caution! In specific cases it is possible to upload a new index profile even when there are collections available. One such case is when a new field has been added to the profile.

New documents processed after the index profile is updated will be included in the update. Existing documents will not automatically be re-indexed, so the new field(s) will not be searchable for these documents until they are re-processed.

A cluster update can either be a *Hot Update* or *Cold Update*, depending on the changes you have applied to the index profile. If you perform a *Hot Update*, then existing collections are not affected. If you perform a *Cold Update*, then you will need to remove all collections from the search engine prior to the update.

Index profile changes that require re-indexing (*Cold Update*):

- Changes to the `<field>` block in the index profile, except if *result="no"* and *type="string"*
- Changes to the `<composite-field>` block, except the `rank-profile` changes
- Changes to *navigator* specifications with *deep="yes"* or *deep="<number>"*

Index profile changes that do not require re-indexing (*Hot Update*):

- Changes to relevance tuning in the `<composite-field>` `rank-profile` fields
- Changes to the `<result-specification>` block in the index profile, except *navigators*
- *Navigator* configurations with *deep="no"*

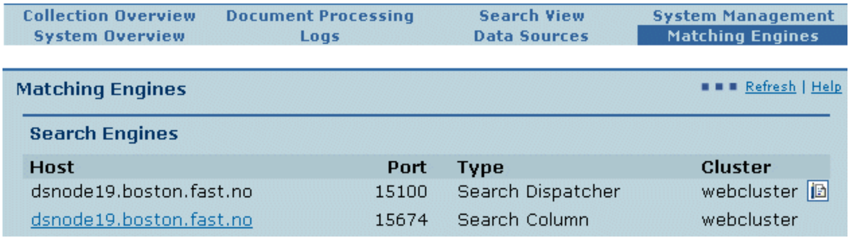
The following steps describe how to update and deploy the new index profile:

1 Determine if the index profile changes require re-indexing (*Cold Update*). If no, go to step 3.

2 Delete all collections within this search engine. Remember to log all the collection configuration data.

3 When running on Windows, stop all **RTS Search** modules from the administrator interface. If this is not done, the system will not be able to delete the old index.

4 Select **Matching Engines** on the FAST InStream administrator interface navigation bar.



Matching Engines

Search Engines

Host	Port	Type	Cluster
dsnode19.boston.fast.no	15100	Search Dispatcher	webcluster
dsnode19.boston.fast.no	15674	Search Column	webcluster

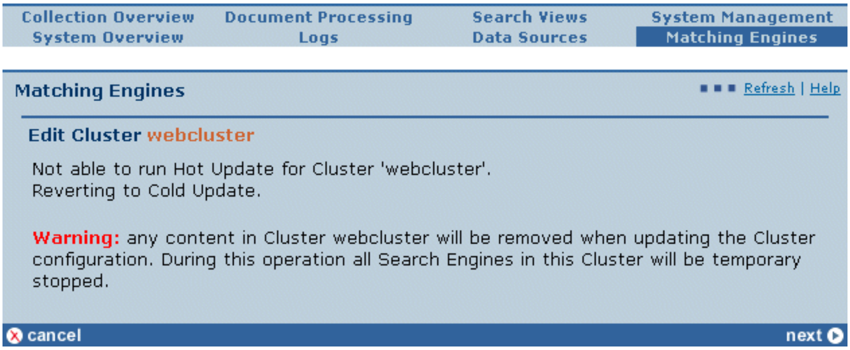
5 Locate the Cluster (for example *webcluster*) you want to modify and click the **Edit** icon to the right of the entry.

6 Enter the location of the index profile you want associated with the cluster.

If you would like to view the index profile to be edited, select **View current index-profile** and the file is displayed.

Click the **Browse** button to select the new index profile to create. Note that the file has to be accessible from the same machine as the web browser. Click **Next**.

7 Depending on the changes you have applied to the index profile, a *Hot Update* or a *Cold Update* is performed on the cluster. In the case of a *Cold Update*, the indexed contents of the cluster will be removed. If you have not already removed all collections from the Cluster, you will need to do so. Click **Next** to confirm that you understand the warning.



Matching Engines

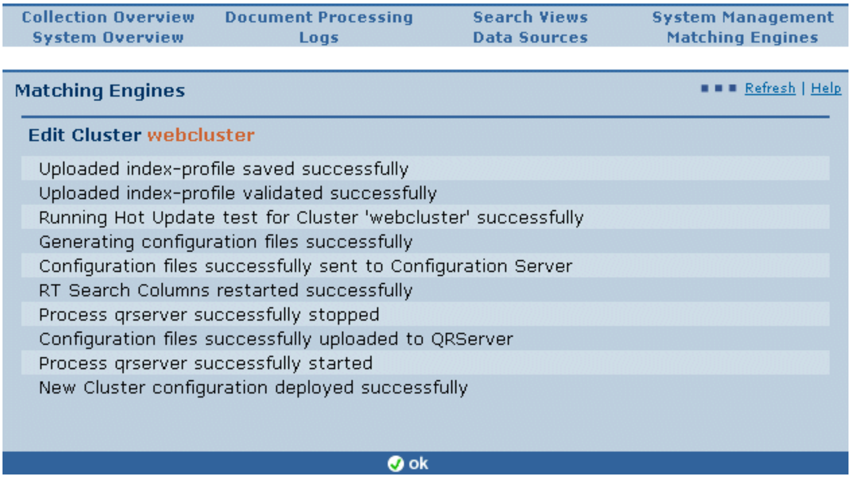
Edit Cluster webcluster

Not able to run Hot Update for Cluster 'webcluster'.
Reverting to Cold Update.

Warning: any content in Cluster webcluster will be removed when updating the Cluster configuration. During this operation all Search Engines in this Cluster will be temporary stopped.

If any errors occur or if the cluster edit is not successful, try clicking **Next** again. This operation can be timely and the browser may not have had enough time to complete the update.

8 The next screen verifies that the search engine modifications have been completed.



Edit Cluster webcluster

Uploaded index-profile saved successfully
Uploaded index-profile validated successfully
Running Hot Update test for Cluster 'webcluster' successfully
Generating configuration files successfully
Configuration files successfully sent to Configuration Server
RT Search Columns restarted successfully
Process qrserver successfully stopped
Configuration files successfully uploaded to QRServer
Process qrserver successfully started
New Cluster configuration deployed successfully

9 Click **ok** and you are returned to the Matching Engines *Edit cluster* screen. If you would like to view the updated index profile, select **View current index-profile** on screen and the updated file is displayed.

Chapter 4

Configuring Access Security

Configuring IP Address Security

FAST InStream provides an optional component security feature that relies on verification of IP addresses. The feature is activated when the following configuration file is present:

```
$FASTSEARCH/etc/accesslist.default.txt
```

This configuration file holds incoming information from all IP addresses accepted by FAST components. Specific components access lists files can also be specified and placed in the same directory as the default access list.

Example Components:

FAST Query and Result Engine: `$FASTSEARCH/etc/accesslist.qrserver.txt`

FAST Configuration Server: `$FASTSEARCH/etc/accesslist.configserver.txt`

Node Controller: `$FASTSEARCH/etc/accesslist.nctrl.txt`

Components will first look for their specific access list. If the file is missing the component will rely on the default file instead. If that file is also missing, then FAST InStream will run without component security.

The access list can hold lists of IP addresses and/or IP netmasks. The latter will allow an administrator to include a trusted set of machines with a single mask.

The configuration files include one line for each IP address accepted. The format for these files is:

```
<IP address> <optional netmask>  # comment
```

where the `<optional netmask>` may be used to allow entire subnets.

Examples:

```
192.168.7.7                     # single machine only
192.168.100.0 0xffffff00        # entire subnet, (hex notation)
192.168.110.0 255.255.255.0     # entire subnet, (decimal notation)
```

IP `127.0.0.1` (localhost) is by default always allowed access and need not be added to the list.

All components subscribing to these files will check the files every five minutes and reload them if the files have changed (checked by time stamp). If the file(s) were not present at system start-up, then adding the files later requires a restart of the component affected or a full system restart.

Unauthorized access will result in network shutdown without any warning.

Securing API Access Using SSL Proxy

It is possible to configure FAST InStream to protect the Content and Query APIs via an SSL Proxy. In this way it is possible to authenticate remote API clients using client certificates and encryption.

Contact FAST Professional Services if you need to use this feature.

Chapter 5

Configuring the Taxonomy Toolkit

This chapter gives an overview of how to configure the FAST InStream Taxonomy Toolkit (hereinafter referred to as the *Taxonomy Toolkit*).The Taxonomy Toolkit provides a user interface to make it easier for you to create, edit and maintain XML configuration files that specify taxonomy structure and category mapping information.

This chapter describes the categorization document processing stages and taxonomy configuration files used to configure a taxonomy. It then explains how to start FAST InStream using the FAST Taxonomy Toolkit. The information in this subsection includes:

- Document Processing Stages for Taxonomy
- Index Profile Category Fields
- Taxonomy Toolkit Configuration File
- Taxonomy Deployment Configuration File
- Starting the FAST Taxonomy Toolkit Application
- Managing Taxonomy Using TaXML Files
- Using the FAST Taxonomy Toolkit
- Deploying the Taxonomy
- Starting the Taxonomy Toolkit on a Remote Server

Refer to the *Categorizing Content and Search Results* chapter in the *Product Overview Guide* for detailed concepts and terminology related to the various capabilities for categorization and taxonomy support in FAST InStream.

Document Processing Stages for Taxonomy

FAST InStream may be configured for automatic categorization during document processing. This is configured using the Taxonomy Toolkit. Two categorization methods exist:

- URI based categorization. The Taxonomy Toolkit allows you to base the mapping of documents to categories based on the document URIs.

  This is accomplished by a assigning a configured list of mappings from URI to category. Any number of documents may be assigned to a given category, and a document may be assigned to several categories.
- Rule-based categorization. Categorization is based on programmatic rules specified using the Python programming language.

Creating a Custom Pipeline for Taxonomy

A set of document processing stages is required in order to perform this automatic categorization. The document processing stages operate on category data that is stored in an internal database called the *archive server.* This database is updated when deploying the taxonomy information as defined using the Taxonomy Toolkit. This is described later in more detail in subsection *Deploying the Taxonomy* on page 108. If you want to configure any of the document processing stages with non-default configuration parameter values, you must define a custom stage for this processing stage.

To enable the taxonomy feature, create a custom pipeline based on an existing pipeline template such as the *Generic (webcluster)* template and add the required document processing stages. Required stages are: *ManualMapper*, *RuleClassifier*, *TaxonomyBoost* and *TaxonomyTagger* and are described in the following subsections.

The categorization document processing stages are applied in a fixed sequence. The general rule is that the category document element (*taxonomy*) is set only if it has not been set by a previous processing stage. If the *ManualMapper* stage has updated the category information, then Rule-based mapping will not update the category information.

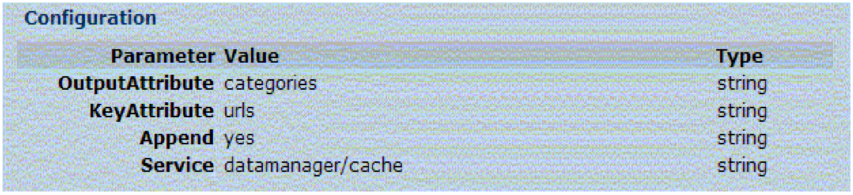
Refer to Chapter 2 *Configuring the FAST Document Processing Engine*, section *Creating a Custom Pipeline With Customized Stages* for more details about pipelines and stages.

ManualMapper

You can provide a mapping from URLs to a set of categories using the Taxonomy Toolkit. The *ManualMapper* document processing stage then uses the URL of an incoming document as a lookup key to consult this mapping, and annotates the document with the provided category membership information, if found.

Use this document processing stage if you know the classification of a (number of) page(s) and want to manually map them to categories using the Taxonomy Toolkit. Refer to *Using the FAST Taxonomy Toolkit* on page 104 for information on how to manually map documents.

ManualMapper document processing interface:



Configuration

Parameter	Value	Type
OutputAttribute	categories	string
KeyAttribute	urls	string
Append	yes	string
Service	datamanager/cache	string

Table 5-1 *ManualMapper* Parameter Values

Parameter	Description
OutputAttribute	This parameter specifies which document attribute will receive the document output. *Default: categories* (this value is shared by all the taxonomy toolkit processors and should not be changed in a standard configuration)
KeyAttribute	This parameter identifies the document attribute containing the URI to be used as a lookup key. Typically, this is a URL, but if you are feeding database content or from XML, enter the field that you use as the document ID when feeding. *Example: URL for web content; productID or similar for database/XML content* *Default: urls* (should be used unless you have applied custom document processing that uses another processing element for the URL)

Table 5-1 *ManualMapper* Parameter Values

Parameter	Description
Append	If *Append* is set to *yes* the processor extends the list of categories in the *OutputAttribute* of the document. If the *OutputAttribute* already contains categories, then categories from the *ManualMapper* processor are appended to the list) If *Append* is set to *no*, the processor does not extend the category list and skips processing if the *OutputAttribute* contains any categories. *Default: yes*
Service	This parameter specifies the name of the document storage server. *Default: datamanager/cache* (MUST be used in a standard configuration)

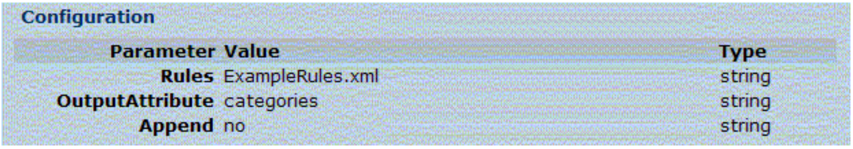
RuleClassifier

You can create rules for use in categorizing documents using the Taxonomy Toolkit and the Python programming language.

Use this document processing stage to classify sets of documents (based on their content) and not individual documents. Individual mappings (using the *ManualMapper* processing stage) override the *RuleClassifier*. Refer to *Using the FAST Taxonomy Toolkit* on page 104 in this chapter for information on how to create mapping rules.

Refer to *Add Categorization Rules* on page 106 for examples of categorization rules.

RuleClassifier document processing interface:



Configuration

Parameter	Value	Type
Rules	ExampleRules.xml	string
OutputAttribute	categories	string
Append	no	string

Table 5-2 *RuleClassifier* Parameter Values

Parameter	**Description**

Rules	The rules are loaded from an XML filename as defined in *Rules*, all matching rules are assigned to the same category.
	This filename must correspond to the `filename` used in the `mappingrulesfile` parameter in the *configuration.taxonomytoolkit.xml* file.
	Example of the XML file containing rules:
	`<?xml version="1.0" encoding="UTF-8"?>`
	` <mappingrules>`
	`  <rules topic="Top/Sports">`
	`  <rule priority="3001">`
	`  <![CDATA[((url.find("http://www.cnn.com/sports/"))`
	`     > -1)]]>`
	`  </rule>`
	` </rules>`
	`</mappingrules>`
	Default: ExampleRules.xml

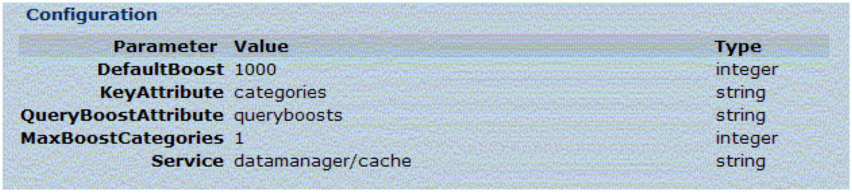
OutputAttribute	A rule is associated with a category. If the rule returns true (value != 0), the document attribute named in *OutputAttribute* is extended with the id of the category containing the rule. *Default: categories* (this value is shared by all the taxonomy toolkit processors and should not be changed in a standard configuration)
Append	If *Append* is set to *yes* the processor extends the list of categories in the *OutputAttribute* of the document. If the *OutputAttribute* already contains categories, then categories from the *ManualMapper* processor are appended to the list). If *Append* is set to *no*, the processor does not extend the category list and skips processing if the *OutputAttribute* contains any categories. *Default: no*

TaxonomyBoost

This document processing stage performs category boosting; this stage boosts all documents belonging to a specific topic or category.

This boosting is equivalent to that done with the boost module, only that it is done to every document in the category. For every document, a lookup (on the *KeyAttribute*) is performed to receive boosts from the category/categories the document has been classified as. If the document hasn't been classified, only the default boost is added to the document.

TaxonomyBoost document processing interface:



Configuration

Parameter	Value	Type
DefaultBoost	1000	integer
KeyAttribute	categories	string
QueryBoostAttribute	queryboosts	string
MaxBoostCategories	1	integer
Service	datamanager/cache	string

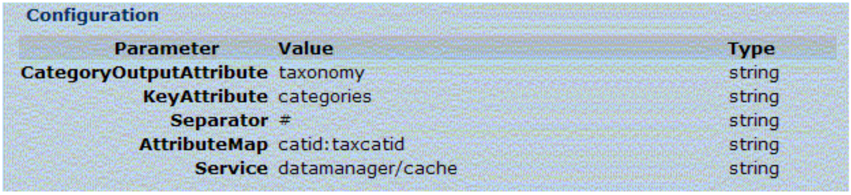
Table 5-3 *TaxonomyBoost* Parameter Values

Parameter	Description
DefaultBoost	All documents processed by this stage receive a default boost whether they have been boosted or not. *Default: 1000*
KeyAttribute	This parameter specifies the document attribute that lists the categories for document classification. *Default: categories* (this value is shared by all the taxonomy toolkit processors and should not be changed in a standard configuration)
QueryBoostAttribute	This parameter specifies which document attribute will receive the document query boost output. This attribute is read by the RankTuning processor to actually perform the query boosting of the document. *Default: queryboosts* (should be used as the RankTuning processor is configured to use the same attribute)
MaxBoostCategories	This parameter specifies the number of categories to provide boosts to the document. A value of *-1* indicates that the document should receive boosts from all categories it belongs to. Any other positive number will set the maximum number of categories to boost from. *Default: 1* (only receive boost from the first category the document has been classified into; the first category should be the primary and most significant category of the document.)
Service	This parameter specifies the name of the document storage server. *Default: datamanager/cache* (MUST be used in a standard configuration)

TaxonomyTagger

This document processing stage tags documents with supplementary taxonomy information as defined using the Taxonomy Toolkit. This processing stage tags documents with additional category meta-data such as *catid* (or *category ID*, represented as a string) in the default configuration.

TaxonomyTagger document processing interface:



Configuration

Parameter	Value	Type
CategoryOutputAttribute	taxonomy	string
KeyAttribute	categories	string
Separator	#	string
AttributeMap	catid:taxcatid	string
Service	datamanager/cache	string

Table 5-4 *TaxonomyTagger* Parameter Values

Parameter	Description
CategoryOutput Attribute	This parameter specifies the output attribute of the category string that combines the *KeyAttribute* and *Separator* parameters. *Default: taxonomy* (MUST be used in a standard configuration as this value corresponds to the default index profile configuration)
KeyAttribute	This parameter specifies the document attribute that lists the categories for document classification. *Default: categories* (this value is shared by all the taxonomy toolkit processors and should not be changed in a standard configuration)
Separator	This value assigns the delimiter character to be used between multiple output values. # is used when assigning category info, as this is the defined way to separate categories in the *taxonomy* field. *Default: #*
AttributeMap	This parameter specifies the mapping from taxonomy attributes to document attributes. Valid (left side) parts of the map are *catid*, *title*, *description*, *source*, *documenturl*, *parentcatids* and *children*. *Default: catid:taxcatid* (MUST be used in a standard configuration as this value corresponds to the default index profile configuration)
Service	This parameter specifies the name of the document storage server. *Default: datamanager/cache* (MUST be used in a standard configuration)

Index Profile Category Fields

Categorization fields in FAST InStream are used to implement the *Supervised Result Clustering*, *Search within Category* and *Taxonomy Index* features.

A prerequisite for the *Taxonomy Index* to work is that a corresponding taxonomy be created from either the Taxonomy Toolkit or by defining the appropriate TaXML files (refer to *Managing Taxonomy Using TaXML Files* on page 102 for more information).

The Taxonomy Toolkit and corresponding processing defines a set of document elements that will be mapped to the index. Categories are represented in the index as follows:

- As a text string representing the full category definition (for example, “Top/Computers/Internet”). This is the normal id of the category. This representation can be searched against as any other text string, and it is possible to search for words within the string (search for all documents that are included in any category that includes the term “Internet”). The limitation with this format is that you cannot always limit a query to a unique category. Assume that you want to search for documents within the category “a/b/c”, but there may also exist sub-categories below this category, “a/b/c/d”. Searching for documents within the category “a/b/c” using a field of this format will also return documents within sub-categories, “a/b/c/d”.

  The category string is tokenized (prior to indexing and on query side) so that “a/b/c” is indexed as “a b c” and “a/b/c/d” is indexed as “a b c d”. Hence, a query with a filter parameter indicating “taxonomy=a/b/c” will match all documents with category field containing the string “a b c”. This will then match both the “a/b/c” and “a/b/c/d” category.
- In order to enable search within distinct categories, each category has also been assigned a unique category id. Using this id, it is possible to search within a unique category without getting results from sub-categories as well.

  When using the Taxonomy Toolkit, this id will correspond to the leaf node of the category (for example, the category “Top/Business” will have the unique id ‘Business’). If ‘Business’ appears several places in the category tree, each unique category that has ‘Business’ as the leaf node will get the unique id ‘Business1’, ‘Business2’.

This index profile configuration (*datasearch-4.x-taxonomy.xml*) defines two fields, *taxonomy* and *taxcatids*. Based on this profile, these fields will be mapped from the corresponding elements (with the same name) within the document processing.The *result-specification categorization* assigns the field *taxonomy* to the built-in categorization support. This includes the *Taxonomy Index* feature and the *Supervised Result Clustering* by category. The `name=` XML element is mandatory but currently not used, as only one categorization field is supported.

The following document related fields are defined as part of this default index profile:

- *taxonomy* – Category ID (text), e.g. “Top/Computers/Internet#Top/Computers/Chat”. The hash character is used as a delimiter between multiple categories assigned with a document. This field will exist for all categorized documents.
- *taxcatids* – Category ID, e.g. “Internet#Chat” where ‘Internet’ and ‘Chat’ is the unique IDs for the categories ‘Top/Computers/Internet’ and ‘Top/Computers/Chat’ respectively. This is used in the same way as ‘taxonomy’, except that the full category representation is replaced with a unique ID. This field may be used when searching within unique categories.

Only the ‘*taxonomy’* field is required in order to perform category search.

A query front-end may implement ‘drill-down into category’ queries by use of these fields. In a similar way as ‘search for similar’ feature it is possible to implement a ‘search within category’ link under each category assigned with a search result. This would then result in a query that consists of the original query with an additional filter term ‘taxcatids:<category ID>. The <category ID> can be found as part of the query result, as long as both fields are defined to be part of the query result set (‘result’ not set to ‘no’ in the index profile for ‘*taxonomy’* and ‘*taxcatids’*. Refer to the *Query Integration Guide* for more details on how to perform category queries.

Taxonomy Toolkit Configuration File

The *workbench.properties* file configures the Taxonomy Toolkit.

Default Configuration File

```
tools=Taxonomy
standalone-mode=true
server-url=
datasource=
look-and-feel=metouia
xml-directory=
clientaccessible-servername=
taxonomy.source-name=FAST
taxonomy.ds-script-location=
taxonomy.ranktuning.absolutekeywords=on
taxonomy.action.feed-to-ds=on
taxonomy.relatedtopics=off
```

Parameters

The following table describes the parameters.

Table 5-5 *workbench.properties* Configuration Parameters

Parameter	Description
tools	For future use, do not change value. *Default: tools=Taxonomy*
standalone-mode	Only standalone mode is supported in FAST InStream. A client-server mode may also be configured, but this requires FAST Professional Services support. *Default: standalone-mode=true*
server-url	Leave empty. Only applicable when using a client-server mode. *Default: server-url=*
datasource	Leave empty, TaXML files are used. It is possible to use a database instead, but this requires FAST Professional Services support. *Default: datasource=*
look-and-feel	This parameter allows you to change the look and feel of the Taxonomy Toolkit editor. Supported look-and-feels: *metouia, kunststoff, system* *Default: look-and-feel=metouia*

Table 5-5 *workbench.properties* Configuration Parameters

Parameter	Description
xml-directory	This parameter identifies the directory (absolute or relative from the current directory) in which to load and save TaXML files. It is recommended that you leave this field empty to use the default *$FASTSEARCH/etc/taxonomy/taxonomies*. NOTE: Use Forward ("/") slashes only for all platforms (including Windows applications); do not use backward ("\") slashes. *Default: xml-directory=*
clientaccessible-server-name	Leave empty, only applicable when using a client-server mode. *Default: clientaccessible-servername=*
taxonomy.source-name	Not used in FAST InStream. The name of the source (origin) of new taxonomy topics and documents. This should be the name of the organization using the tool. *Default: taxonomy.source-name=mysitename*
taxonomy.ds-script-location	Leave empty, not used. *Default: taxonomy.ds-script-location=*
taxonomy.ranktuning. absolutekeywords	Enable (=*on*) this parameter if you want select absolute rank for queries per category. Disable (=*off*) this parameter if you want to limit business manager’s ability to manipulate rank. Use care when enabling this parameter so that you do not give absolute rank position for queries to an entire category. *Default: taxonomy.ranktuning.absolutekeywords=on*
taxonomy.action. feed-to-ds	Enable (=*on*) this parameter if you want to deploy the taxonomy to FAST InStream from the Taxonomy Toolkit menu. Disable (=*off*) this parameter if you are using the Taxonomy Toolkit from a remote server/PC to disable the menu choice. *Default: taxonomy.action.feed-to-ds=on*
taxonomy.relatedtopics	Not used in FAST InStream. *Default: taxonomy.relatedtopics=off*

Taxonomy Deployment Configuration File

The *configuration.taxonomytoolkit.xml* file configures the Taxonomy Deployment into FAST InStream (the *importtaxonomy* utility):

Note! The configuration file *configuration.taxonomytoolkit.xml* is not required for the Taxonomy Toolkit, but only for deployment of the taxonomy files into FAST InStream.

The file has the following XML syntax:

```
<taxonomytoolkit>

<!-- General options -->
<general tscollection = ""
         updatefastservertaxonomy = "yes"
         updatedocumentprocessors = "yes"
         refeedtrainingsetdocuments = "yes"
         batchsize = "20"
         documentstorageservice = "datamanager/cache" />

<!-- Location of the CORBA name service -->
<nameservice hostname="test602.oslo.fast.no" port="24099" />

<!-- File options -->
<files structurefile = "etc/taxonomy/taxonomies/ExampleStructure.xml"
       contentfile = "etc/taxonomy/taxonomies/ExampleContent.xml"
       mappingrulesfile = "etc/taxonomy/taxonomies/ExampleRules.xml"
       ranktuningfile = "etc/taxonomy/taxonomies/ExampleRank.xml"
       trainingsetfile = "etc/taxonomy/taxonomies/ExampleTrainingset.xml"
       examplesfile = "var/taxonomy/examples.xml"
       odpdocumentfolder = "var/taxonomy/odpdocuments"
       categorydocumentfolder = "var/taxonomy/categorydocuments"
       fastservertaxonomyfile = "var/taxonomy/taxonomy.xml"/>

<!-- Logging options -->
<logging loglevel = "0x2f7"/>

</taxonomytoolkit>
```

The following table defines the location and name for the TaXML files; you can edit these files or they will be created using the Taxonomy Toolkit. Refer to *Managing Taxonomy Using TaXML Files* on page 102 for more details. The file path is relative to the FAST InStream installation directory location.

Table 5-6 *configuration.taxonomytoolkit.xml* File Parameters

Parameter	**Description**
structurefile	Taxonomy structure definition file.
contentfile	Describes the taxonomy content.
mappingrulesfile	Describes the mapping-rules for the Rule-based categorization.
ranktuningfile	Describes the categorization related rank tuning parameters.
trainingsetfile	Not used but has to be set. Keep the default value.

Note! It is recommended to use the default directory and file names. If using other file names, you will have to instantiate new document processing stages with corresponding configuration parameters. Refer to the *Index Profile Category Fields* on page 95 subsection for more details.

Starting the FAST Taxonomy Toolkit Application

The FAST Taxonomy Toolkit runs on a UNIX server or a Windows PC. It does not need to be installed on a FAST InStream node.

You need to have Java Virtual Machine (JVM) version 1.4.0 (or higher) installed on the PC/server.

1 Open a Command Line Window.

2 Change directory to the `$FASTSEARCH/bin` (UNIX) or `%FASTSEARCH%\bin` (Windows) folder.

3 Type the applicable command. Note that this is a single command:

(The `-Dfastsearch` parameter tells the Taxonomy Toolkit where FAST InStream is installed.)

UNIX:

```
$FASTSEARCH/bin/workbench.sh
```

UNIX (and enabling Japanese language option):

```
$FASTSEARCH/_jvm/bin/java -Duser.language=ja -Duser.country=ja -
Duser.variant=ja -Dfastsearch="$FASTSEARCH" -cp "$FASTSEARCH/bin/
Workbench.jar:$FASTSEARCH/etc/taxonomy" com.fastsearch.
workbench.Workbench &
```

Windows:

```
%FASTSEARCH%\bin\workbench.cmd
```

Windows (and enabling Japanese language option):

```
%FASTSEARCH%\_jvm\bin\java -Duser.language=ja -Duser.country=ja -
Duser.variant=ja -Dfastsearch="%FASTSEARCH%" -cp
"%FASTSEARCH%\bin\Workbench.jar;%FASTSEARCH\etc\taxonomy"
com.fastsearch.workbench.Workbench
```

4 You may also create a shell script or batch file, and place an icon on your desktop:

```
@ECHO OFF
java -Dfastsearch=%FASTSEARCH% -cp %FASTSEARCH%\bin\Workbench.jar;%
FASTSEARCH%\etc\taxonomy com.fastsearch.workbench.Workbench
```

Note that the last two lines are a single command.

Managing Taxonomy Using TaXML Files

The Taxonomy Toolkit operates on a set of intermediate XML files that defines the taxonomy and category mapping information. A customer application may also create the XML based configuration if that is more appropriate (e.g. within the customer's own category management application). The XML definition is denoted TaXML.

- **ExampleStructure.xml** – Describes the taxonomy structure
  - Based on RDF XML format
  - Example (from Open Directory):

```
<Topic r:id="Top">
    <catid>Top</catid>
    <d:Title>Top</d:Title>
    <narrow r:resource="Top/Business" />
    <narrow r:resource="Top/Sports" />
</Topic>
<Topic r:id="Top/Business">
    <catid>Business</catid>
    <d:Title>Business</d:Title>
      <related r:resource="Top/Science" />
</Topic>
<Topic r:id="Top/Sports">
    <catid>Sports</catid>
    <d:Title>Sports</d:Title>
    <narrow r:resource="Top/Sports/Football" />
</Topic>
<Topic r:id="Top/Sports/Football">
    <catid>Football</catid>
    <tag source="Fast" />
    <d:Title>Football</d:Title>
</Topic>
```

- **ExampleContent.xml** – Describes the taxonomy content including manual mapping.
  - Based on RDF XML format
  - Example:

```
<Topic r:id="Top">
    <catid>Top</catid>
</Topic>
<Topic r:id="Top/Sports">
    <catid>Sports</catid>
</Topic>
<Topic r:id="Top/Sports/Football">
    <catid>Football</catid>
    <link r:resource="http://www.livescore.com/" />
    <link r:resource="http://www.soccernet.com/home.html" />
</Topic>
<ExternalPage about="http://www.homeideas.com/">
```

```
  <tag source="ODP"/>
</ExternalPage>
```

- **ExampleRules.xml -** Describes the mapping-rules for the Rule-based categorization
  - Used by *RuleClassifier* document processor
  - Example:

```
<rules topic="Top/Computers">
     <rule priority="3000">((body.lower().find("internet"))
                                             &gt; -1)</rule>
</rules>
```

- **ExampleRank.xml -** Describes the rank tuning parameters
  - Example:

```
<rank topic="Top/Sports/Football">
    <boost type="relative" value="5000"/>
    <keywords>
        <keyword name="football">
            <boost type="relative" value="100"/>
        </keyword>
    </keywords>
</rank>
```

Note! When creating categorization rules, you must either substitute ‘<’ and ‘>’ by &lt; and &gt; or you can wrap your rule in XML CDATA tag, e.g. <![CDATA[((body.lower().find("A")) > -1)]]>

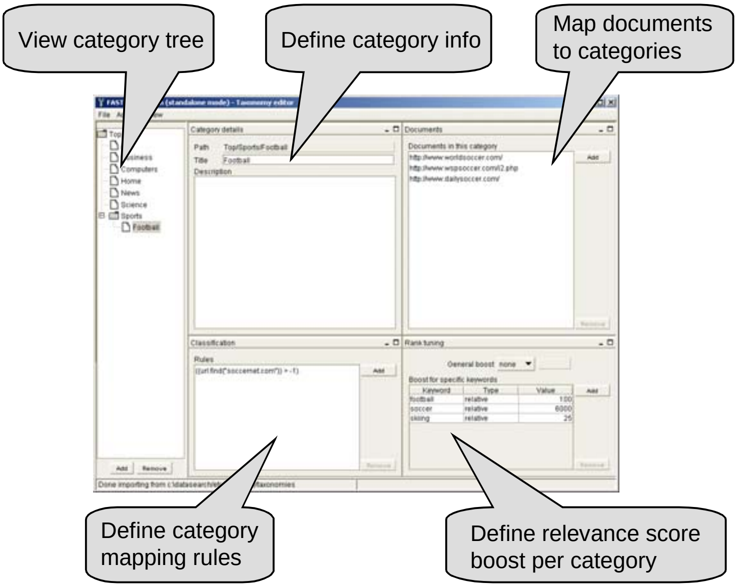
Using the FAST Taxonomy Toolkit

A Taxonomy Toolkit is provided for easy maintenance of the taxonomies and the category mapping. This tool is implemented as a standalone tool that works on XML files that contain the actual configuration. In this way it is possible to update the taxonomy configuration either by using this tool or directly on the XML files. The latter approach enables a customer application to update taxonomy information or mapping.

The Taxonomy Toolkit is delivered with an example taxonomy. This example is populated with documents that can be accessed and edited directly as an XML file or by means of a tree, where categories can be expanded or collapsed. By clicking on a category, a set of operations applicable to that category becomes available. To create your own taxonomy, you can edit this example.

The following lists the topics that can be edited when using the Taxonomy Toolkit:

- Adding / Removing / Editing Categories Using the FAST Taxonomy Toolkit
- Manual Mapping
- Add Categorization Rules
- Taxonomy Rank Boosting



Path Top/Sports/Football
Title Football
Description

Documents in this category
http://www.worldsoccer.com/
http://www.wspsoccer.com/i2.php
http://www.dailysoccer.com/

Rules
((url.find("soccernet.com")) > -1)

General boost none

Boost for specific keywords

Keyword	Type	Value
football	relative	100
soccer	relative	6000
skiing	relative	25

Done importing from c:\datasearch\et... ...\taxonomies

Figure 5-1   FAST InStream Taxonomy Toolkit Editor

Adding / Removing / Editing Categories Using the FAST Taxonomy Toolkit

Adding a Category/Subcategory - To add a new category, click the **Add** button, and a dialog box appears that allows you to type the name of the category to add. You can add multiple categories at a time by separating their names with a semi-colon.

To add a subcategory, highlight the parent category and click the **Add** button.

To deselect a category, press the **Ctrl** key while clicking on the category.

Removing a Category - To remove a category, highlight the category from the tree and click the **Remove** button.

Editing a Category - You can modify the title of a category (topic) and its description by editing the corresponding fields. The description may contain HTML. This information is not utilized inside the FAST InStream Index.

Manual Mapping

URI based categorization is performed by assigning documents (by their URI) to a category. Any number of documents may be assigned to a given category, and a document may be assigned to several categories.

The automatic categorization is performed by the *ManualMapper* document processor.

Note! Multiple duplicate documents with different URI may exist (real duplicates or redirect links to the same document/page). In this case FAST InStream will only index these documents once (as long as they are within the same collection). One of the duplicate URIs is selected as the 'uri' of the document within the Index (and returned in the result). If you define a manual mapping for a duplicate URI (not the URI returned in the 'uri' field), this will still work correctly for the indexed document.

Add Categorization Rules

Rules are used by the *RuleClassifier* processor to enable you to describe criteria for classifying a document.

Rules are used to program almost any criterion and combination of criteria that must be true for a document to be classified into a category. You may add several rules per category.

Rules are specified using the Python programming language (*www.python.org*). This provides a very powerful tool for creating custom categorization rules. The rule definitions are defined per category using the Taxonomy Toolkit editor. The following table provides a set of mapping examples.

Table 5-7 Python Rules Examples

Rule	Coding
Title includes A:	((title.find("A")) > -1)
Body includes A:	((body.find("A")) > -1)
Body includes A or a:	((body.lower().find("a")) > -1)
Body includes A AND B:	((body.find("A")) > -1) and ((body.find("B")) > -1)
Title="Welcome" (in any case) OR body starts with "Dear all":	(title.lower() == "welcome") or (body.find("Dear all")) == 0)
URL includes A or a:	((url.lower().find("a")) > -1)
URL includes www.fast.no AND body includes search engine	((url.lower().find("www.fast.no")) > -1) and ((body.find("search engine")) > -1)

Note! The Taxonomy Toolkit will wrap the rules into XML CDATA tags, so that the ‘>’ and ‘<‘ characters are properly handled within the generated TaXML.

Taxonomy Rank Boosting

The category rank boost feature enables rank boost of all documents within a category against all or specified queries. Use this feature to boost, for example, a certain medical category to queries that are specific to this area of medicine.

The feature enables boost for complete queries (except query filter terms), not individual query terms. It is not possible to boost a given page if the end-user types a query that includes a given term but also may include other terms (e.g. if you boost the query "fast search", the documents will not be boosted if the query is "fast" or "search").

Boosting for specified queries may be useful in association with the query logs provided by FAST InStream. In this way the most frequent queries on the site may be considered for boosting against specific categories.

For example, if you run a food recipes site, you can boost all categories related to food, restaurants, cuisine etc. so that people searching your index see these results first. It is also possible to lower the rank score, if you wish to make documents in a topic appear lower in the result set.

The following rank boost capabilities are provided:

- Relative boost of all documents in a category - for all queries
- Relative boost of all documents in a category - for specified queries
- Absolute boost of all documents in a category - for all queries
- Absolute boost of all documents in a category - for specified queries

The relative boost will increase the probability for the documents in the category to show up early in the result, but the actual effect is dependant on how the dynamic ranking is configured for your system (configured using the index profile).

The absolute boost will assign all documents in a category (that match the query) to a fixed position in the result set.

Caution! Use the absolute boost feature with care. If many documents exist in a category, the absolute boost feature may overrule dynamic ranking and may not provide the desired relevance effect.

Deploying the Taxonomy

A business manager normally operates the Taxonomy Toolkit but does not and should not have direct access to the FAST InStream installation. Instead a FAST InStream system operator deploys the taxonomy using a set of utility programs as defined below.

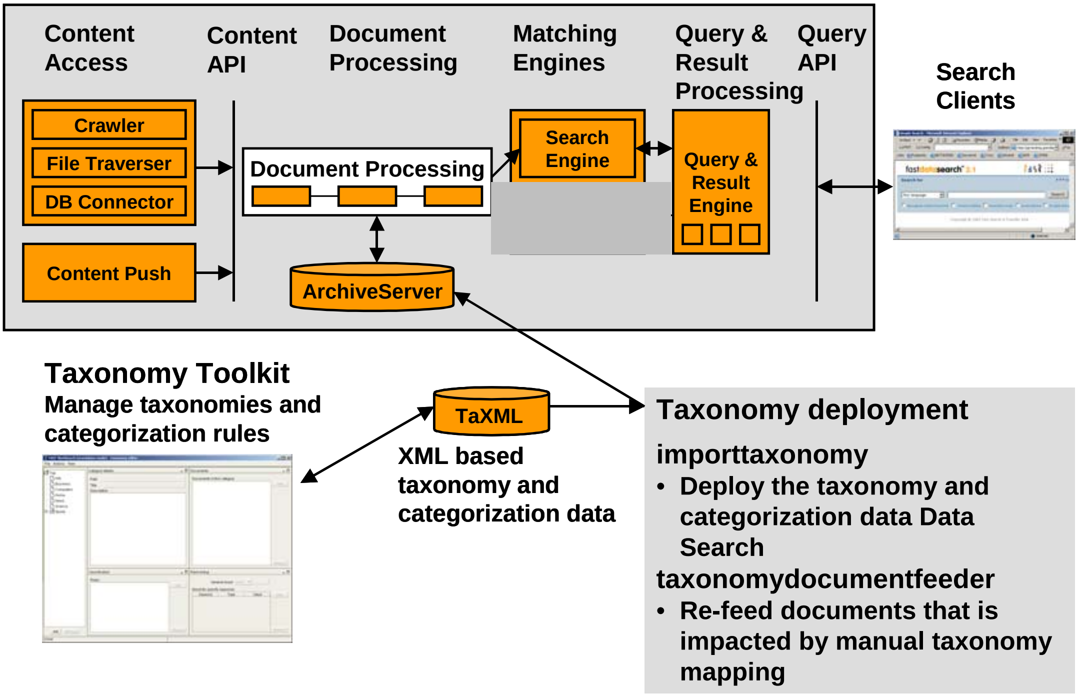


Figure 5-2 Taxonomy Deployment

It may be useful to separate the administration tasks between a Business Manager (managing the business rules) and a FAST InStream System Manager (managing the FAST InStream installation). The Taxonomy Toolkit is very powerful, but changing taxonomy or adding categorization rules may have large implications if applied incorrectly. Hence, a recommended procedure is that a FAST InStream System Manager verifies the new taxonomy configuration data before deploying it into the production environment. This can be done either by inspecting the changes manually, or (more likely) deploying the configuration on a staging server.

Deployment

The taxonomy deployment is performed using the following steps:

1 If running Taxonomy Toolkit on a remote PC/server, you will need to copy the following files to the FAST InStream admin component after editing the taxonomy with the Taxonomy Toolkit:

UNIX: `$FASTSEARCH/etc/taxonomy/taxonomies/*`

Windows: `%FASTSEARCH%\etc\taxonomy\taxonomies\*`

Use the same directory and file names for source and destination.

2 The utility '*importtaxonomy*' is used to populate the categorization data represented as TaXML into the FAST InStream document processing framework. The data is deployed via the ArchiveServer (internal FAST InStream database also used by the Rank Tuning Toolkit).

The '*importtaxonomy*' application performs the following steps:

- Contacts the FAST Configuration Server and submits the TaXML files
- Resets the Document Processing Engines in order to activate the new configuration.

If you run the Taxonomy Toolkit from a component within your FAST InStream installation, the taxonomy deployment can be performed from the Taxonomy Toolkit menu.

If you run the Taxonomy Toolkit on a remote PC/server, run the following command on the Admin component of your FAST InStream installation after copying the XML files as described under step (1):

UNIX:

```
cd $FASTSEARCH
./bin/importtaxonomy
```

Windows:

```
cd %FASTSEARCH%
\bin\importtaxonomy
```

Using a Staging Environment

In most cases it is recommended that you use a test environment (staging server) on a limited part of the content to verify the taxonomy deployment, and to verify that the rules are applied as expected without undesired side-effects. To do this:

1 Install FAST InStream and Taxonomy toolkit on another host.

2 Configure your staging environment exactly like the production environment, but define a test collection with a representative subset of your data.

3 Modify your taxonomy.

4 Make a list of tests to perform to verify each modification.

5 Re-crawl (or re-feed) your data, see above.

6 Check that the results are as expected, that every rule works as intended, that documents are classified into newly added categories etc., according to your test case list from step 4.

7 Repeat steps 3-6 until your configuration is satisfactory.

8 Copy the relevant files over to the production server before setting the configuration live. The files are:

```
$FASTSEARCH/etc/taxonomy/*
```

The etc/taxonomy files are input to the 'importtaxonomy' application that generates the configuration files used by the document processors.

9 Run the ‘importtaxonomy’ to deploy the taxonomy:

UNIX:

```
cd $FASTSEARCH
./bin/importtaxonomy
```

Windows:

```
cd %FASTSEARCH%
\bin\importtaxonomy
```

Verification

With usage of a search front-end that displays categorization of the search results, you will be able to verify that your categorization works as expected. If you use the default (auto-generated) query front-end from the FAST InStream administrator interface, make sure that you select the front-end.

Verification of the categorization:

- Search for a word or set of words you know exists in one of the documents you used in the manual mapping.
- The search result should now include the document (if indexed) as part of your search result list.
- Verify that the displayed categories contain the category you specified for this document or rule.
- You may also use a query like “taxonomy:Top/Computers/Internet” which will retrieve all documents classified into that category or a sub-tree of that category.
- Use a query like "taxcatids:Internet" to retrieve all documents in this specific category.

You can see the results in plain text using the default query front-end.

If the verification of your categorization fails:

- Make sure that the expected document is searchable
- Verify in the log that all your document processors loaded without any problems
- Verify that all the configuration parameters for the document processors match the field names used in your index-profile
- Verify that your pipeline contains the document processor stages required for taxonomy, and that they are configured in the correct order.

Starting the Taxonomy Toolkit on a Remote Server

If you want to run the Taxonomy Toolkit on a remote PC/server (a PC/server outside the associated FAST InStream installation), you must perform the following steps:

1 Create a *datasearch* directory and define the environment variable *$FASTSEARCH* (UNIX) or *%FASTSEARCH%* (Windows) to point to this directory.

2 Copy the following files to the PC/server (you find the files in the same directories in your FAST InStream installation):

UNIX:

```
$FASTSEARCH/bin/Workbench.jar
$FASTSEARCH/etc/taxonomy/workbench.properties
$FASTSEARCH/etc/taxonomy/taxonomies/*
```

Windows:

```
%FASTSEARCH%\bin\Workbench.jar
%FASTSEARCH%\etc\taxonomy\workbench.properties
%FASTSEARCH%\etc\taxonomy\taxonomies\*
```

Chapter 6

Configuring Linguistic Processing

This chapter explains how to customize the dictionaries for linguistic-specific query and content processing within FAST InStream. This chapter includes:

- Master Dictionaries
- Anti-Phrase Dictionary
- Dictionary Management with the dictman Tool
- Synonyms and Spelling Variations
- Master Dictionary Customization Examples
- Configuring Similarity Vector Creation
- Changing Lemmatization Mode

Refer to Chapter 9 *Advanced Linguistic Processing* in the *Product Overview Guide* for supported language information.

Master Dictionaries

Master dictionaries are provided in source format with FAST InStream to support the Spell Checking feature.

These dictionaries are used by the FAST Query and Result Engine and are located in:

UNIX: *$FASTSEARCH/resources/dictionaries/*

Windows: *%FASTSEARCH%\resources\dictionaries\*

in the *phrasing* and *spellcheck* directories. Master dictionaries include support for:

Advanced Spell Checking

- phrase dictionary *(indep_propernames)*
- phrase exception list *(indep_propernames_exceptions)*

Simple Spell Checking

- single word dictionaries for multiple languages (*<language>_spell*)
- single word exception list for multiple languages (*<language>_exceptions)*

These master dictionaries can be modified with the *dictman* tool. The source format (no file extension) is used for editing, while the binary format (includes *.aut* or *.ftaut* file extension) is used for executing Spell Checking.

The following subsections describe these dictionaries in more detail.

Dictionaries that Support Advanced Spell Checking

Phrase Dictionary

FAST InStream supplies a phrase dictionary that contains common phrases such as names of famous persons (for instance "elvis presley"), names of places (for instance "san francisco"), and names of companies (for instance "kraft foods").

The default phrase dictionary is located at:

UNIX:

$FASTSEARCH/resources/dictionaries/spellcheck/indep_propernames_iso8859_1.ftaut

Windows:

%FASTSEARCH%\resources\dictionaries\spellcheck\indep_propernames_iso8859_1.ftaut

You may modify the supplied phrase dictionary by adding or removing terms. Alternatively, you may create separate phrase dictionaries which contain customer specific phrases only. If you choose to create multiple phrase dictionaries, you can enable selecting a specific phrase dictionary to be used for spell checking at query time.

If a query phrase does not exactly match any entry in the selected or default dictionary, but is close to some dictionary entries, the phrase that is considered the closest match is suggested as a replacement to the original query phrase. If a query phrase matches an entry in the dictionary exactly, the phrase is protected by quotes and Simple Spell Checking will not be allowed to change the terms of the phrase. If there are no entries in the dictionary that are close to matching the query phrase, the query phrase remains unchanged and is sent to the FAST Search Engine.

Phrase Exception List

The phrase exception list is a dictionary that contains words that are not to be considered for spell checking. When a query term matches an entry in the exception list, the term will be protected from spell checking changes.

The default phrase exception list is located at:

UNIX:

$FASTSEARCH/resources/dictionaries/spellcheck/indep_propernames_exceptions_iso8859_1.aut

Windows:

%FASTSEARCH%\resources\dictionaries\spellcheck\indep_propernames_exceptions_iso8859_1.aut

Note! The phrase dictionaries are not language specific. However, the default phrase dictionary supplied with FAST InStream are optimized for English.

Dictionaries that Support Simple Spell Checking

Single Word Dictionaries

FAST InStream supplies language specific dictionaries that contain common words for the particular language.

The filenames for the single word dictionaries have the form:

<language>_spell_<encoding>.ftaut

where <*language*> is a 2-letter ISO language code and <*encoding*> is an ISO encoding (Example: *de_spell_iso8859_1.ftaut)*. You may modify the supplied single word dictionaries by adding or removing terms.

The default single word dictionaries are located in the *FASTSEARCH/resources/dictionaries/spellcheck/* folder.

If a query term does not match any entry in the dictionary, but is close to some dictionary entries, the term that is considered the closest match is suggested as a replacement to the original query term. If a query term exactly matches an entry in the dictionary, or there are no entries in the dictionary that are considered close to matching the query term, the query term remains unchanged and is sent to the FAST Search Engine.

Single Word Exception Lists

Single word exception lists are dictionaries that contain words that are not to be considered for spell checking. When a query term matches exactly an entry in the exception list, the term will be protected from Simple Spell Checking. In contrast to the phrase exception list, the single word exception lists are language specific.

The filenames for the exceptions lists have the form:

<language>_exceptions_<encoding>.aut

where <*language*> is a 2-letter ISO language code and <*encoding*> is an ISO encoding (Example: *de_exceptions_iso8859_1.aut*).

The default single word exception lists are also located in the *FASTSEARCH/resources/dictionaries/spellcheck/* folder.

Anti-Phrase Dictionary

The Query Server configuration file enables the anti-phrase dictionary. Refer to Chapter 9 *Advanced Linguistic Processing* in the *Product Overview Guide* for a list of supported dictionaries.

The anti-phrase dictionary is located in:

UNIX: *$FASTSEARCH/resources/dictionaries/phrasing/indep_antiphrasing.aut*

Windows: *%FASTSEARCH%\resources\dictionaries\phrasing\indep_antiphrasing.aut*

The configuration for this is located in:

UNIX: *$FASTSEARCH/etc/config_data/QRServer/<search cluster name>/etc/qrserver/ qtf-config.xml*

Windows: *%FASTSEARCH%\etc\config_data\QRServer\<search cluster name>\etc\qrserver\qtf-config.xml*

and looks like this:

```
<instance name="antiphrase" type="internal" resource="FastQT_Phrase">
   <argument-list>
      <string value="NONPHRASE"/>
   </argument-list>
   <parameter-list>
      <parameter name="autfile" value="../resources/dictionaries/
      phrasing/indep_antiphrases.aut"/>
   </parameter-list>
```

Anti-Phrase Example

The following example describes how to add a term to the master anti-phrase dictionary. The procedure consists of:

- Opening the dictionary with *dictman*
- Adding the new anti-phrase (this example uses the term *whatsup*)
- Closing the dictionary
- Compiling the dictionary (check progress with the *jobstatus* command)
- Deploying the dictionary
- Restarting the FAST Query and Result Engine

Example:

```
bash$ cd $FASTSEARCH
bash$ cd bin
bash$ . setupenv.sh
Environment variables successfully set.

bash$ ./dictman -u <username>-p <password>
logging in on localhost, port 16086 as demo
connected successfully.

command (? for help)> open /phrasing/indep_antiphrases

/phrasing/indep_antiphrases locked
/phrasing/indep_antiphrases opened

command (? for help)> insert /phrasing/indep_antiphrases whatsup
Inserting into /phrasing/indep_antiphrases (type: EntityDictionary)
inserting 1 entry
inserted 1 entry

command (? for help)> close /phrasing/indep_antiphrases
saving dictionary '/phrasing/indep_antiphrases'... saved
/phrasing/indep_antiphrases closed
/phrasing/indep_antiphrases unlocked

command (? for help)> compile /phrasing/indep_antiphrases
started compilation using job number 1

command (? for help)> jobstatus 1
job1.1.1        RUNNING compile /dictionaries/phrasing/indep_antiphrases
job1.1  RUNNING running 1 sub tasks
job1    RUNNING running 1 sub tasks

command (? for help)> jobstatus 1
job1.1.1        OK      compile /dictionaries/phrasing/indep_antiphrases
job1.1  OK      RUNNING -> OK
job1    OK      RUNNING -> OK

command (? for help)> quit
ok

bash$ cp $FASTSEARCH/j2ee/work/compile/dictionaries/phrasing/
indep_antiphrases.aut $FASTSEARCH/resources/dictionaries/phrasing/

Restart the QRServer
```

Dictionary Management with the *dictman* Tool

The *dictman* tool works with the Master Dictionaries that are located in the resource repository; it does not work directly with the *FASTSEARCH/resources/dictionaries/*.

The *dictman* tool replaces the *dictctrl* tool used in previous versions of FAST InStream.

Note! Refer to Appendix E *Dictionary Management Tool Usage* for syntax, usage and options information for the *dictman* tool.

Modifying Dictionaries with the *dictman* Tool

In general, if you want to modify the dictionaries in the resource repository with the *dictman* tool:

- start the *dictman* tool
- edit the dictionaries in the source format
- compile the modified dictionaries into binary form
- copy them to the *FASTSEARCH/resources/dictionaries/* location
- restart the FAST Query and Result Engine to deploy the new modified dictionaries

Refer to the *Synonyms and Spelling Variations* on page 125 for a detailed example of this procedure. Complete the following:

Starting *dictman*

1 Start *dictman* using the following command:

UNIX: `$FASTSEARCH/bin/dictman -u username -p password`
Windows: `%FASTSEARCH%\bin\dictman -u username -p password`

2 Users by default are specified in:

UNIX: `$FASTSEARCH/j2ee/server/vespa/conf/users.properties`
Windows: `%FASTSEARCH%\j2ee\server\vespa\conf\users.properties`

dictman users must have the User role as specified in:

UNIX: `$FASTSEARCH/j2ee/server/vespa/conf/roles.properties`
Windows: `%FASTSEARCH%\j2ee\server\vespa\conf\roles.properties`

Refer to the information in Chapter 9 *Configuring the FAST Business Manager's Control Panel* to make sure your User roles are properly configured.

3. After logging in, the following output appears:

   ```
   $ $FASTSEARCH/bin/dictman -u user -p pass
   logging in on localhost, port 16086 as user
   connected successfully.

   command (? for help)>
   ```

   dictman displays a command line prompt, allowing you to do enter a variety of commands concerning dictionary management.

Editing and Compiling the Dictionaries

This subsection described the procedure for editing and compiling a dictionary. There are examples of this procedure at the end of the subsection.

1. To edit a spellcheck, phrase or anti-phrase dictionary, open a spellcheck, phrase or anti-phrase dictionary. Spellcheck dictionaries are of the *SpellcheckDictionary* type; phrase and anti-phrase dictionaries are of the *EntityDictionary* type.

   Use the `ls` command if you want to list all dictionaries.

   Use the `open` command to open a dictionary.

2. Update or insert terms either by using the `insert` or the `import` command. The `insert` command allows for updating frequencies of already inserted entries, or add new entries if they were not there already. Refer to *Adding Terms via Command Line* on page 128 for additional information about the `insert` command.

3. Close the dictionary using the `close` command to save the dictionary and remove it from memory.

4. Compile the dictionary using the `compile` command.

   Dictionaries are compiled into *.aut* files through the *compile* command. Depending on the dictionary size, compilation may take a couple of minutes and is always executed asynchronously as a job. The job is started by the user and then runs in the background on the *J2EE* server.

   Compilation is always incremental. The compiler job checks if the dictionary has changed since the last compilation, and only recompiles when it is newer than the corresponding *.aut* file. To force a re-compilation, you may need to delete the *.aut* file or use the *clean* command which will delete the whole work directory.

Note! Type '*?*' at the command prompt for help with *compile* command options.

5 The *compile* command returns a job number that can be used to get a summary of the job status. Use the `jobstatus <id>` command to get a summary that contains the job status code and a status message. An `OK` message indicates job completion.

6 Leave *dictman* using the `quit` command.

7 The complete log file is located in:

UNIX: `$FASTSEARCH/j2ee/server/vespa/log/deployment.log`
Windows: `%FASTSEARCH%\j2ee\server\vespa\log\deployment.log`

8 Upon completion, a default folder named *work* is automatically generated. The compiled dictionaries are located in the *work* folder at:

UNIX: `$FASTSEARCH/j2ee/work/compile/dictionaries/`
Windows: `%FASTSEARCH%\j2ee\work\compile\dictionaries\`

Editing and Compiling Dictionaries Example

```
open /spellcheck/en_spell_iso8859_1
lookup /spellcheck/en_spell_iso8859_1 "apple"
apple: 4
insert /spellcheck/en_spell_iso8859_1 "apple"=3
open /spellcheck/en_spell_iso8859_1
lookup /spellcheck/en_spell_iso8859_1 "apple"
apple: 3
import /spellcheck/en_spell_iso8859_1 "c:\new_spellcheck.txt" "\t"
ISO8859_1 [Windows Example]
close /spellcheck/en_spell_iso8859_1
compile /spellcheck/en_spell_iso8859_1
jobstatus 1
quit
```

Editing and Compiling Dictionaries Example using the *with* Command

An alternative way to edit and compile the dictionary is by using the *with* command. Use the *with* command if you plan on making several changes. *with* allows you to specify the dictionary name once and then perform a sequence of operations on it. Using the *with* command, the previous example now becomes:

```
with /spellcheck/en_spell_iso8859_1
open
lookup "apple"
apple: 4
insert "apple"=3
open
lookup "apple"
apple: 3
import c:\new_spellcheck.txt" "\t" ISO8859_1 [Windows Example]
close
compile
```

```
jobstatus 1
endwith
quit
```

Copying the Modified Dictionaries

Compiled dictionaries have to be manually distributed from the work directory to the */resources/dictionaries/* on all processor server nodes.

Use the *copy* command for a single-node installation and copy the generated file to the */resources/dictionaries/* directories on all nodes that have a running FAST Query and Result Engine.

For example:

UNIX: *cp -r $FASTSEARCH/j2ee/work/compile/dictionaries/* $FASTSEARCH/resources/dictionaries*

Windows: *copy /r %FASTSEARCH%\j2ee\work\compile\dictionaries\* %FASTSEARCH%\resources\dictionaries*

Processing the Documents

Make sure that lemmatization is enabled for the related search engine cluster. To enable lemmatization, set the attribute *lemmatize="yes"* for one or more fields in the index profile. This also assumes that your document processing pipelines are configured according to the procedures described in Chapter 2 *Configuring the FAST Document Processing Engine*.

If necessary, re-process any documents that were processed before these configuration changes were made.

Re-Processing the Documents After Configuration Change

Documents processed before configuration changes are made will need to be re-processed if you want these configuration changes (such as lemmatization or synonyms) to apply to them as well.

Documents processed after the configuration changes are made will be processed with the configuration updates.

Note! Re-processing may take an extended period of time depending on your configuration; make sure you allow time for documents to be completely re-processed.

If documents in FAST InStream came from sources other than the crawler, sources such as the File Traverser, database connector or content API, then these documents must be re-inserted.

This procedure applies only to crawled documents. For other pushed content, the documents must be pushed again using the same mechanism as used originally.

To re-process a single collection from the crawler:

1 Stop the *Crawler* from the **System Management** tab in the administrator interface.

2 Enter *$FASTSEARCH/bin* (**UNIX**) or *%FASTSEARCH%\bin* (**Windows**).

3 **For UNIX**: Set up the required environment variables (if not already set in the command line shell):

`source ./setupenv.sh` or `./setupenv.csh`

4 To re-process a specific collection:

```
./postprocess -o -I master -R <collection name>
```

To re-process all collections:

```
./postprocess -o -I master -R "*"
```

5 Wait until the *postprocess* command has terminated.

6 Restart the crawler.

Restarting the Processor Servers

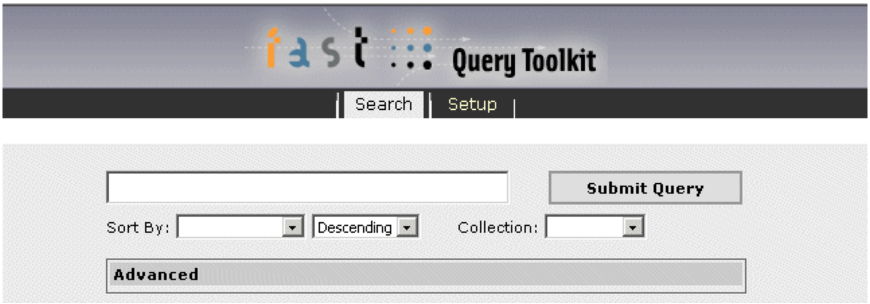
After deploying the dictionaries, restart the document processors. Select **System Management** on the FAST InStream administrator interface navigation bar to restart the module(s) named **Document Processor** by clicking the **Restart** icon.

Note! If there is more than one Processor Server in the system, all Processor Servers must be restarted.

Enabling the New Configuration

To enable the new configuration:

1 Access the *FAST Query Toolkit*.



2 Click **Advanced** and check the box that applies to your modifications such as *Spelling*, *Lemmatization* (for synonyms and lemmatization), or *Proper Name*.

Verifying the New Configuration

To verify the new configuration, search your documents with the synonyms or spelling variations you added and verify that the search result contains documents to match them.

Note that the generated search front-end does not expose all query features available in FAST InStream. You can change the current query setup by modifying parameters in the Query API. Refer to the *Query Integration Guide* for details on how to create a custom search front-end.

Synonyms and Spelling Variations

The synonym and spelling variations process allows different words with the same meaning to be included in your search results. FAST InStream provides two mechanisms for searching for synonyms and spelling variations:

- Indexing-side synonym and spelling variation expansion. This is the process of adding synonyms to the terms in documents when the document is being indexed. Indexing-side synonym and spelling variation expansion is discussed in this section. This feature is performed as part of the lemmatization stage within document processing, and is controlled using the same parameters in the index profile and the query.
- Query-side synonym expansion. This means expanding user queries with spelling variations and the suggestion of synonyms at query time. Query-side synonym expansion requires support from FAST Professional Services.

Synonyms and Spelling Variations Example

This example assumes that:

- *$FASTSEARCH* (**UNIX**) or *%FASTSEARCH%* (**Windows**) is the FAST InStream base installation directory
- You are running a FAST InStream installation
- You have already configured a collection

This example uses English as an example language; adding spelling variations for other languages follows the same procedure.

Problem: How can I make my page *My Favorite Cars* show up in search results when a user searches for the term *automobile*?

Solution: The current search is for the *automobile*, and the page you want to be included in the search results contains the synonym *cars*. Set up the synonym feature so that different words with the same meaning will be included in the search results.

If you want to find a document that contains the word *cars* when you search for *automobile*, then you should add the term *automobile* to the synonym list for *cars* by enabling the spelling variation functionality.

Procedure:

1 Start the *dictman* tool. Refer to the *Starting dictman* on page 119 for proper procedures.

2 Create a *VariationDictionary*. Go to the *Creating a VariationDictionary* subsection and complete the procedure.

3. Edit the dictionary. Go to the *Editing the VariationDictionary* subsection and complete the procedure.
4. Upon completion, return to the *dictman* tool process and complete the procedure described in *Editing and Compiling the Dictionaries* on page 120. For this example, close and compile the dictionary:

   ```
   close /custom/en_synonyms
   compile /custom
   ```

5. Complete the remaining steps starting with *Copying the Modified Dictionaries* on page 122 and finishing with *Verifying the New Configuration* on page 124. By checking the *Lemmatization* box in the *Advanced* section of the *FAST Query Toolkit* screen, the search for *car* should now include search results containing the word *autos* or *automobiles*. Searching for *autos* should include search results for documents containing the word *car* or *cars*. Searching for *automobiles* should include search results for documents containing the word *car*, *cars*, *auto* or *automobiles*.

Creating a *VariationDictionary*

Dictionaries for Indexing-side expansion of synonyms and spelling variations are called *VariationDictionaries*. To add synonyms or spelling variations for a supported language, you must first create a *VariationDictionary.* The *VariationDictionary* for a given language may be edited using the *dictman* tool.

This example assumes that the master dictionaries are managed on the same server as the dictionary service which is contained in the *J2EE* server.

To create a *VariationDictionary*, use the *create* command to specify:

- a *resource handle.* This resembles a file name (including sub directories), but without an extension. It acts as a unique identifier for the dictionary
- the *type*. In this case, the type is a VD or *VariationDictionary*.
- the *language* being used. This must be an ISO-639 compliant language code.

The master dictionaries are held in a resource repository that is managed and held consistent by the *dictman* tool. The repository is located in:

UNIX: *$FASTSEARCH/j2ee/*

Windows: *%FASTSEARCH%\j2ee\*

This directory structure is mapped to compiled dictionaries (*.aut* or *.ftaut* files) when a *compile* command is issued.

Example: In order to create a *VariationDictionary* for English in the */custom* subdirectory of the resource repository, enter the following line in the *dictman* shell:

```
command (? for help)> create /custom/en_synonyms VariationDictionary en
Creating dictionary of type 'vd' in directory '/custom/' with name
'en_synonyms' in language 'en'

command (? for help)> ls /custom
 Listing dictionaries in Directory "/custom" of type "*"
 handle              type      size encoding   language  locked by   open
 ------------------------------------------------------------------------
 /custom/en_synonyms  VD          0 UTF-8      en        user        X

 command (? for help)>
```

In the previous directory listing, the Dictionary *en_synonyms* was created in the */custom* directory. Its contents are saved in UTF-8 encoding (VariationDictionaries are always in UTF-8 encoding). The language was set to *"en"*, and the dictionary is already opened and locked by the current user.

Changes to this dictionary will remain in memory until the dictionary is saved (using the *save* command or implicitly during a close operation).

Note! A different user logging on to *dictman* will see that this dictionary has been created, but will not be able to perform changes unless it is unlocked. Unlocking can be done using the *unlock* or implicitly using the *close* command.

Editing the *VariationDictionary*

VariationDictionaries contain key-value pairs: the key denotes the term to be extended, and the value contains a list of expansion terms.

There are three methods for adding terms to a *VariationDictionary*:

- Adding Terms via Command Line
- Importing a Text File
- Importing an XML File

Terms are always imported into an opened dictionary. In the last example the dictionary was automatically opened after it was created. Use the *open* command to open a dictionary at a later time. The changes will be held in memory until a *save* or *close* operation is performed.

Adding Terms via Command Line

Open a shell and use the *insert* command to add an individual term or a set of terms:

```
insert /custom/myfolder "automobiles" = "[car,cars]"
```

`insert /phrasing/indep_antiphrases "what is"` (in this anti-phrase example, the `=""` can be omitted)

This example shows that:

- each *insert* command gets the dictionary the term is to be inserted into. Use the *with* command if you plan on making several changes. *with* allows you to specify the dictionary name once and then perform a sequence of operations on it
- *insert* gets a list of "key" = "value" pairs
- the format for values in *VariationDictionaries* is a list of terms, put in brackets [], and separated by commas

The following items should be noted:

- key and value have been put in quotes. This is only necessary if they contain spaces, but forgetting them may lead to false keys, which should be avoided
- keys and values have to be in lower case alphanumeric (Unicode) characters and must not contain punctuation or control sequences (other than the brackets and commas as shown)
- a command such as *insert* may be split into multiple lines if the "\" character is put at the end of a line
- if the key already exists in the dictionary, the list of expansion terms is appended to the already existing one. If you want to replace the terms, first execute a *remove <key>* command

Importing a Text File

A list of expansions may be imported using the *import* command:

```
import /custom/myfolder/mysynonyms.txt "\t" UTF-8
```

This command imports the file *mysynonyms.txt* which is in UTF-8 encoding into the */custom/myfolder* dictionary. Keys and values are separated by the Tab character.

The format of keys and values is the same as when inserting terms via command line, but they must not be enclosed in quotes.

The *delimiter* and *encoding* are optional parameters. By default, the delimiter is the Tab character, and the encoding defaults to the one used for the shell. It can be displayed using the *charset* command. See charsets for a list of possible encodings.

Note! The delimiter is a regular expression in a PERL-compatible syntax. It matches the character or sequence of characters between key and value. PERL-regular expressions contain a number of reserved characters that have to be escaped using the "\" (backslash) character. For example, "." would have to be escaped by "\."

Importing an XML File

The Dictionary Management (*dictman)* Tool also supports XML format that was supported by the *dictctrl* tool. Appendix E *Dictionary Management Tool Usage* provides detailed syntax and options information.

To add synonyms to one of the files, for example, the *en_synonyms.xml* file is located in:

UNIX: *$FASTSEARCH/lib/samples/dictionaries/*

Windows: *%FASTSEARCH%\lib\samples\dictionaries\*

```
<?xml version="1.0" encoding ="ISO-8859-1"?>
 <dictionary lang="en">
 <comment>
  Add a change log or something similar.
 </comment>
<entry>
 <input>car</input>
 <input>cars</input>
 <output>automobiles</output>
 <output>autos</output>
</entry>
<entry>
 <input>auto</input>
 <input>autos</input>
 <output>car</output>
 <output>automobiles</output>
</entry>
</dictionary>
```

The previous example identifies the words *automobiles* and *autos* as synonyms for *car* and *cars* (first entry), and *car* and *automobiles* as synonyms for *auto* and *autos*. The *<input>* parameters are the terms as found in the document and the *<output>* parameters are the additional terms that can be used in the query to find the given document.

The previous example results in four *insert* commands to be executed. A text input file would have to look like this:

```
car    [automobiles,autos]
cars   [automobiles,autos]
auto   [car,automobiles]
autos  [car,automobiles]
```

Note! Make sure the "lang" attribute of the "dictionary" tag matches the language of the dictionary.

Since the values are always merged during an *insert* and since duplicates are removed from the merged set of terms, it is possible to import a file twice without any consequences. This means you may add contents of a text or XML file and import it again, and only those terms whose contents are changed will have been affected. However, if terms were deleted, you must delete the dictionary or remove the terms by command line.

Adding Synonyms Example

This example assumes that Fast Bikes sells accessories. One of the more popular items is a “smiley” bike sticker. Fast Bikes would like to update the synonym dictionary to ensure that if the user types in “happy” bike sticker, results are still returned for the “smiley” bike sticker.

To add synonyms to the dictionary:

1 Using the *dictman* tool, create a *VariationDictionary* by typing the following command:

```
create /custom/en_synonyms VariationDictionary en
```

2 To insert a term, enter:

```
insert /custom/en_synonyms “smiley”=“[happy]”
```

3 To lookup the term in the dictionary, enter:

```
lookup /custom/en_synonyms “smiley”
```

4 Close the dictionary:

```
close /custom/en_synonyms
```

5 Compile the dictionary:

```
compile /custom/en_synonyms
```

6 To check the job status, type the following command:

```
jobstatus <#>
```

Note: Running OK (repeated twice) indicates the job has completed.

7 Compiled dictionaries have to be manually distributed from the work directory to the *%FASTSEARCH\resources\dictionaries\util\lemmatizer folder*:

```
copy
c:\datasearch\j2ee\work\compile\dictionaries\custom\en_synonyms\en_syn
onyms.aut c:\datasearch\resources\dictionaries\util\lemmatizer
```

8 In the *util\lemmatizer* folder, replace the original *en_synonyms* dictionary with your new dictionary. Rename the original to *en_synonyms_original*; and rename your new dictionary to *en_synonyms*.

9 Restart the **Document Processor** via the **System Management** screen in the administrator interface.

More Examples Using the dictmal tool for Synonyms

```
command (? for help)> with /custom/en_synonyms
```

using /custom/en_synonyms

Creating:

```
command (? for help)> create vd en
```

Creating dictionary of type 'vd' in directory '/custom/' with name 'en_synonyms' in language 'en'

Error : resource '/dictionaries/custom/en_synonyms' already exists

Opening:

```
command (? for help)> open
```

/custom/en_synonyms locked

/custom/en_synonyms opened

Adding one synonym word:

```
command (? for help)> insert "smiley"="[happy]"
```

Inserting into /custom/en_synonyms (type: VariationDictionary)

inserting 1 entry

inserted 1 entry

Adding two or more synonym words:

```
command (? for help)> insert "cbot"="[cg,ct]"
Inserting into /custom/en_synonyms (type: VariationDictionary)
inserting 1 entry
inserted 1 entry
```

Adding phrase (acronym):

```
command (? for help)> insert "bba"="[british bankers association]"
Inserting into /custom/en_synonyms (type: VariationDictionary)
inserting 1 entry
```

Verifying that changes have been made:

```
command (? for help)> query
querying for '', returning max. 50 entries starting from offset 0
Java Server Pages: [JSP,jsp]
bba: [',british bankers association]
cbot: [chicago board of trade,cb,cg,ct]
smiley: [happy]
returned 4 entries
```

Closing:

```
command (? for help)> close
saving dictionary '/custom/en_synonyms'... saved
/custom/en_synonyms closed
/custom/en_synonyms unlocked
```

Compiling:

```
command (? for help)> compile
started compilation using job number 1
```

Checking status:

```
command (? for help)> jobstatus 1
job1.1.1       OK     compile /dictionaries/custom/en_synonyms
job1.1  OK     RUNNING -> OK
job1    OK     RUNNING -> OK
```

Importing synonyms from a text file:

```
british bankers association    [bba]
cad    [canada]
cadmium [cd]
cbs    [ccs]
ccf    [irgs, cap, caps, floor, floors, flr, collar]
chf    [swiss franc, swissy]
chicago board of trade   [cbot]
chicago mercantile exchange    [cme]
chicago options exchange       [cboe]
```

Note that you can use different delimiters than tab as follows:

```
command (? for help)> insert ?
insert <dictionary> <term[=value] [...]>
   inserts a key-value pair into a dictionary
   The value must fit to the dictionary type:
    Entity:       "string"
    Lemmatization: "[list,of,strings]"
    Variation:    "[list,of,strings]"
    Synonym:      "[[list,of,spell,variants],[list,of,acronyms],[[synonym,set],[...]]])"
    Spellcheck:   weight (integer, logarithmic scale)
if a key already exists, the values will be merged.
command (? for help)>  import /usr/company/test/synonyms.txt "\t"  ISO-8859-1
```

```
importing '/usr/company/test/synonyms.txt' into '/custom/en_synonyms' in ISO-8859-1
encoding

sending 9 entries

added 9 entries (9 total)

finished
```

Getting help and information:

Within the *dictman* tool you can find help about each command via '?'.

For example: `command (? for help)> import ?`

```
import <dictionary> <text file> [delimiter="\t" [encoding=default-
encoding]]

imports the contents of a text file into a dictionary

splits each line into key and value at a delimiter character and performs an insert.

delimiter can be every sequence of characters or '\t' for tab.

encoding is an identifier as returned by 'charsets' signifying the encoding used in the
file. Defaults to the encoding that is returned by the 'charset' command
```

For example: **`command (? for help)> info`**

```
created              Fri Sep 24 14:35:44 EDT 2004

description

dictionaryHandle        /custom/en_synonyms

encoding               UTF-8

generateExpansion       true

generateReduction       true

language               en

lastCompilationFinished  -

lastCompiled            -

lastCompiledBy          -

lastModified           1096063198000

lastModifiedDate        Fri Sep 24 17:59:58 EDT 2004

lastSaved              Fri Sep 24 17:59:58 EDT 2004

locker               demo
```

```
    numEntries        13
    open            true
    type           VariationDictionary
    version
```

Verifying that keys and values have been recognized:

```
command (? for help)> query
querying for ", returning max. 50 entries starting from offset 0
Java Server Pages: [JSP,jsp]
bba: [',british bankers association]
british bankers association: [ , ",bba]
cad: [ , ",canada]
cadmium: [ , ",cd]
cbot: [chicago board of trade,cb,cg,ct]
cbs: [ , ",ccs]
ccf: [ , ",irgs, cap, caps, floor, floors, flr, collar]
chf: [ , ",swiss franc, swissy]
chicago board of trade: [ , "]
chicago mercantile exchange: [ , ",cme]
chicago options exchange: [ , ",cboe]
smiley: [happy]
returned 13 entries
```

To incorporate changes into a custom dictionary:

```
$ cd $FASTSEARCH
$ nctrl stop procserver_1
$ cp j2ee/work/compile/dictionaries/custom/en_synonyms.aut resources/
dictionaries/util/lemmatizer/
$ nctrl start procserver_1
```

Feed documents with the input values and index. Verify the search results by enabling lemmatization

How to import synonyms in XML file with dictman tool

1 Synonym XML File:

```
<?xml version="1.0" encoding ="ISO-8859-1"?>
<dictionary lang="en">
    <comment>
        2002/04/29, name@comp.com: added product list for dept xyz
    </comment>
    <entry>
        <input>JSP</input>
        <input>jsp</input>
        <output>Java Server Page</output>
        <output>Java Server Pages</output>
    </entry>
    <entry>
        <input>Java Server Pages</input>
        <input>Java Server Page</input>
        <input>jsp</input>
        <output>JSP</output>
        <output>jsp</output>
    </entry>
<entry>
        <input>cad</input>
        <output>canada</output>
</entry>
<entry>
        <input>british bankers association</input>
        <output>bba</output>
</entry>
<entry>
        <input>cbot</input>
        <output>chicago board of trade</output>
</entry>
</dictionary>
```

2 **`command (? for help)> with /custom/en_synonyms`**

using /custom1/en_syn

3 **`command (? for help)> varimport ?`**

varimport <dictionary> <variation xml file> [--verify]

imports entries for a dictionary containing spell variations for expansion at indexing-time

the dictionary must be a variation dictionary. Its language must adhere to the language of the file. If the verify option is set, the XML file is verified against its DTD.

4 **`command (? for help)> create vd en`**

Creating dictionary of type 'vd' in directory '/custom1/' with name 'en_synonyms' in language 'en'

5 **`command (? for help)> varimport /usr/company/test/synonyms.xml`**

```
importing entries given in language 'en' from '/usr/company/test/synonyms.xml' into '/custom/en_synonyms'
sending 8 entries
added 8 entries (8 total)
```

6 **`command (? for help)> query`**

```
querying for ", returning max. 50 entries starting from offset 0
british bankers association: [bba]
cad: [canada]
cbot: [chicago board of trade]
java server page: [java server pages,jsp]
java server pages: [java server page,jsp]
jsp: [jsp,java server page,java server pages]
returned 6 entries
```

7 **`command (? for help)> close`**

```
saving dictionary '/custom/en_synonyms'... saved
/custom/en_syn closed
/custom/en_syn unlocked
```

8 **`command (? for help)> compile`**

```
started compilation using job number 1
```

Alternatively you can use:

```
makeaut <inputfile> <outputfile
```

Troubleshooting Guidelines for Synonyms and Spelling Variations

Synonyms do not work:

- Verify that all the synonym configuration steps have been applied. Check that the *FASTEARCH/resources/dictionaries/util/lemmatizer/<lang>_synonyms.aut* file exists. If it is not, recompile using the *dictman* tool.
- Make sure the *Lemmatization* box is checked in the Advanced section in the FAST Query Toolkit.

Synonyms do not work for all documents:

- Re-process the old documents in your system whenever you change the synonym.
- Configuration in order for the new changes to take effect.
- Understand that only documents with the language you have added synonyms for will be affected. Verify that the documents you are missing have the correct language.

Synonyms do not work for all queries:

Synonyms are not subject to lemmatization. For example, if you have added the synonym *automobiles*, then only that term would result in hits and the singular form of the word *automobile* will not. You must add all forms of a synonym to the dictionary.

Master Dictionary Customization Examples

Guidelines for Adding Entries to the Master Dictionaries

The following guidelines apply for adding new information to the dictionaries:

- Add an entry to a phrase dictionary when:
    - the entry to be added is a phrase, for instance "business development toolkit"
    - the entry to be added is a single word, but the end user may tend to write the entry as a phrase. An example is a name like "fileserver", which some end users may write "file server".
    - the entry is a single word proper name, for instance a customer specific and language independent name of a product such as 'whirlpool'.

  Phrases may be added to the default phrase dictionary supplied with FAST InStream. If the default phrase dictionary contains information that is not relevant for the customer specific application, you may instead create a new phrase dictionary containing customer specific terms only and use it instead of the default dictionary supplied with FAST InStream.
- Add an entry to a single word dictionary when the entry to be added is a single word, and at the same time a generic word part of a particular language, for instance "familiarisation" (English) or 'sportfachhandel' (German).
- Add an entry to a phrase or single word exception list when the entry is a word or a phrase that is to be protected from spell checking.

SpellcheckDictionary or *EntityDictionary* Examples

The entries of a *SpellcheckDictionary* or *EntityDictionary* map a string to a positive integer denoting how often the word or phrase given in `<string>` is used. The lower the value is, the more often the word or phrase is used. For example, an entry having a `<frequency>` value of 3 means that it is more common than an entry having a `<frequency>` value of 7.

During spell checking, there may be multiple candidates for replacement of a misspelled term. When several candidates from a dictionary have a similar edit distance to the original term or phrase, then more common words or phrases, listed with a lower `<frequency>` value in the dictionary, are more likely to be chosen.

The following is an example of entries in a *SpellcheckDictionary*:

```
and 2
antidisestablishmentarianism 14
implement 5
zombie 8
```

Phrase dictionaries, exception lists and anti-phrase dictionaries are maintained in dictionaries of the *EntityDictionary* type. The following is an example of entries in an *EntityDictionary* dictionary:

```
astrid lindgren 5
carbon copy 6
elton john 4
new jersey 3
zip code 3
zurich financial services 8
```

Note! The phrase dictionary file may also contain single words.

Compiling and Activating a Phrase Dictionary

If you have created a new phrase dictionary to replace the default phrase dictionary provided with FAST InStream, you will have to modify the *propername.xml* located in:

UNIX: *$FASTSEARCH/etc/config_data/QRServer/<clustername>/etc/qrserver/*

Windows: *%FASTSEARCH%\etc\config_data\QRServer\<clustername>\etc\qrserver\*

The default *propername.xml* looks like this:

```
<?xml version="1.0" ?>
<configuration>
  <matcher type="levenshtein" debug="no">
   <levenshtein automaton = …/resources/dictionaries/spellcheck/
         indep_propernames_iso8859_1.ftaut" phonetics="no" exact="yes"
         match="boundary" start="greedy" prefix="0" tolerance="2">
       <quality type="maximum" penalty="1" threshold="195" />
   </levenshtein>
  </matcher>
</configuration>
```

Note! As indicated, this configuration file also includes a set of configuration options for the spell checking algorithm. It is recommended that you contact FAST Professional Services before modifying these parameters. Modify the filename from *indep_propernames_iso8859_1.ftaut* to the filename of your new phrase dictionary.

It is also possible to use multiple phrase dictionaries. For instance, you can use the FAST default dictionary together with one or more customer specific dictionaries. Contact FAST Professional Services to perform the necessary configuration changes to support multiple phrase dictionaries.

Updating Simple Spell Checking by Language

You may also update the Simple Spell Checking dictionary for a given language.To activate the new dictionary:

1 For all nodes running the FAST Query and Result Engine, copy the binary version of the dictionary to:

UNIX: `$FASTSEARCH/resources/dictionaries/spellcheck/`
Windows: `%FASTSEARCH%\resources\dictionaries\spellcheck\`

2 If the *propername.xml* configuration file has been changed, copy the updated configuration file to all nodes running the FAST Query and Result Engine.

3 Restart all FAST Query and Result Engine components to load the updated dictionary. Additional information on how to use the Spell Checking feature from a custom front end can be found in *Phrase Detection and Spell Checking* in the *Query Integration Guide*.

Changing the Default Language

The query parameter *LANGUAGE* can be used to set the language of the query string in order to apply language specific query processing such as spell check and lemmatization. The default for this parameter is '*en*' (english). Refer to the *Query Integration Guide* for parameter information.

If you want to define another default language for the query string (which applies when the indicated query parameter is not used), this can be done by changing the following configuration file parameter:

Parameter: *default.language*

Configuration file:

UNIX: *$FASTSEARCH/etc/config_data/QRServer/<search cluster name>/etc/qrserver/qtf-config.xml*

Windows: *%FASTSEARCH%\etc\config_data\QRServer\<search cluster\name>\etc\qrserver\qtf-config.xml*

Configuration file location:

The Admin node of your FAST InStream installation.

Example:

<parameter name="default.language" value="en"/>

Valid arguments:

The language code for one of the languages supported by FAST for Advanced Linguistics.

Note! Note: All FAST Query and Result Engine servers in your installation must be restarted in order for the changed configuration to take effect.

Configuring Similarity Vector Creation

Document vectors may be generated for all documents in the processing pipeline, and are controlled from the index profile. Similarity vectors are created based on the elements/ fields that includes the *vectorize* attribute in the index profile field specification.

Detailed functional operation of the vectorization process can be customized via a set of XML configuration files described below.

The configuration files for the vectorization process are located in:

UNIX: *$FASTSEARCH/etc/config_data/DocumentProcessor/linguistics/vectorizer*

Windows: *%FASTSEARCH%\etc\config_data\DocumentProcessor\linguistics\vectorizer*

Stopwords Configuration

This *stopwords* configuration file lists which stopword lists should be used for which languages. The default *configuration.stopwords.xml* configuration file looks like this:

```
<?xml version="1.0"?>
<configuration>
   <stopwords
     language = "en"
     filename = "resources/dictionaries/stopwords/en/vec_stopwords.txt"
     alpha = "none" digit = "any"/>
   <stopwords
     language = "de"
     filename = "resources/dictionaries/stopwords/de/vec_stopwords.txt"
     alpha = "none" digit = "any"/>
   <stopwords
     filename = "resources/dictionaries/stopwords/indep/vec_fallback.txt"
     alpha = "none" digit = "any" length = "4"/>
</configuration>
```

The *resources* directory indicated in the *filename* attribute is relative to the FAST InStream *FASTSEARCH* root directory.

Each language specific stopwords file (indicated in the *filename* attribute in the *stopwords* configuration file) consists of a list of language specific words, one per line. The stopwords list contains common words and phrases for the documents. If no `language` attribute is specified in the file, then the corresponding stopwords list (*fallback.txt* in the file above) will be applied to all documents where the language, automatically detected by FAST InStream, does not fit one of the language specific stopwords lists.

You may add additional stopwords lists for other languages if appropriate, given that you use one of the 54 languages detected by FAST InStream. Languages are identified by ISO codes, and encoding of the files must be UTF-8. It may also be appropriate to tune the stopwords list to your particular application, by adding or removing stopwords from the supplied lists. Refer to Chapter 9 *Advanced Linguistic Processing* in the *Product Overview Guide* for supported language information.

The `language` attribute in the *stopwords* and *boosters* XML files tells the vectorizer which language to use when processing the text. The vectorizer can then use the correct stopwords files and other language-specific configuration settings.

Boosters Configuration

The *boosters* configuration file lists which languages should have simple capitalization-boosting heuristic enabled.

The boosting scheme is based on word capitalization; a word is boosted if it starts with an uppercase letter, unless the word is in the boost exception list (a flat file with one word per line), or is the first word in a sentence.

The default *configuration.boosters.xml* configuration file looks like this:

```
<?xml version="1.0"?>
<configuration>
   <booster language = "en"
   exceptions =
   "resources/dictionaries/stopwords/en/vec_booster_exceptions.txt" />
</configuration>
```

The *resources* directory indicated in the *filename* attribute is relative to the FAST InStream *FASTSEARCH* root directory.

The *boosters* parameter may be defined by language. If no *language* attribute is specified in the file, then the boost will be applied to all documents where the language, automatically detected by FAST InStream does not fit one of the language specific booster entries. The exceptions will be common words that always or frequently start with an uppercase letter, such as abbreviations such as *AM* or month names such as *January.*

FAST InStream provides a booster exception list for English only, so boosting is by default only applied to English documents. You may add additional booster exception lists for other languages if this is one of the 54 languages detected by FAST InStream. Languages are identified by ISO codes, and encoding of the files must be UTF-8. Refer to Chapter 9 *Advanced Linguistic Processing* in the *Product Overview Guide* for supported language information. It may also be appropriate to tune the supplied booster exception list to your particular application.

Note! If no fallback entry exists in the configuration file (a booster entry without language set), boosting will only be applied to the specified languages. It is recommended to perform word capitalization-based boosting with a corresponding exception list.

Re-process the documents after completing changes to the configuration file. Refer to *Re-Processing the Documents After Configuration Change* on page 122 for details.

Changing Lemmatization Mode

FAST InStream includes two modes of lemmatization:

- Lemmatization by Expansion: The words in a document are expanded to all full forms. During indexing, all produced full forms for the document are put into a separate part of the index. No modification to the query is required.
- Lemmatization by Reduction: The words in documents and in the query are reduced to canonical forms (lemmas). Lemmatized query terms are sent to the index of lemmatized words.

Lemmatization by reduction is always the basis for handling lemmatization of Japanese and Korean documents.

For Western languages FAST InStream uses Lemmatization by reduction as the default mode. This section describes how to change the mode of lemmatization for these languages.

Note! It is not possible to change the mode of lemmatization on a per query basis.

Make sure that lemmatization is configured with the same mode on document processing and query processing side. Otherwise queries will not be handled correctly.

Changing the mode of lemmatization requires re-indexing of all documents.

The mode of lemmatization is not defined in the index profile. It has to be configured separately on document processing side and query side.

Note! Lemmatization by reduction and lemmatization by expansion impact memory consumption to varying degrees. For information on how lemmatization impacts memory consumption, refer to Chapter 7 *Benchmarking*, section *Lemmatization Memory Consumption* in the *Deployment Guide*.

Changing Lemmatization Mode for Document Processing

In the following steps the term *<clustername>* denotes the name of the search engine cluster you are using. The default cluster is named *webcluster*.

1 Select **Document Processing** tab in the administrator interface.

2 Select **Advanced Mode**. This enables you to access document processing stages that are normally automatically configured from the index profile.

3 For all pipelines in use within your application (the pipelines you use are listed under the heading **<clustername> Pipelines** in the **Document Processing** Overview page) perform the following steps:

- **a** Select the **Edit Pipeline** icon.
- **b** Note the location of the document processing stage *LemmatizerReduction(<clustername>)* in the pipeline.
- **c** Remove the stage *LemmatizerReduction(<clustername>)*.
- **d** Insert the stage *LemmatizerExpansion(<clustername>)*. Make sure that the new stage is placed in the same location within the pipeline as the removed stage!

4 For the XML (Scope Search) pipeline (pipeline named *'xml (<clustername>)'* listed under the heading **<clustername> Pipelines** in the **Document Processing** Overview page) perform the following additional steps:

- **a** Identify the custom stage *ScopeLemmatizer(<clustername>)* in the **Document Processing** view.
- **b** Select the **Edit Stage** icon for *ScopeLemmatizer(<clustername>)*.
- **c** Change the configfile attribute to *etc/DictionaryConfig_exp.xml*.
- **d** **Submit** change.

Changing Lemmatization Mode for Query Processing

On the query side you may change the lemmatization mode to Lemmatization by expansion by removing the lemmatization stage from your query processing pipelines.

The Query Processing Pipeline configuration file *qtf-config.xml* is located in the following directory on the Administration Node in your FAST Data Search installation (the server where the ConfigServer is running):

UNIX: *$FASTSEARCH/etc/config_data/QRServer/<clustername>/etc/qrserver/*

Windows: *%FASTSEARCH%\etc\config_data\QRServer\<clustername>\etc\qrserver\*

where *<clustername>* indicates the name of the search engine cluster where the configuration applies.

The pipeline configuration is located at the end of the file, and has the following format:

```
<pipeline-list>
  <pipeline name="default">
    <instance-ref name="utf8"/>
    ...
    ...
  </pipeline>

  <pipeline name="scopesearch">
    <instance-ref ... />
  </pipeline>

  ...
  ...
</pipeline-list>
```

To change the lemmatization mode from reduction to expansion, modify the *qtf-config.xml* configuration file as follows:

1 Edit the file and remove all instances named *lemmatizerreduction*. Normally it is most convenient to change these lines in the file to commented XML.

For example, replace

```
<instance-ref name="lemmatizerreduction"/>
```

with

```
<--  <instance-ref name="lemmatizerreduction"/>  -->
```

2 In the same configuration file, add the languages to be lemmatized by expansion to the *"lemmatize"* stage.

For example, for English, German and French, the configuration would look like:

```
<instance name="lemmatize" type="internal"
resource="FastQT_DefaultIndex">
  <parameter-list>
    <parameter name="enable" value="1"/>
    <parameter name="languages" value="en,de,fr,ja,ko,szh,zh-
simplified,tzh,zh-traditional"/>
    <parameter name="reductionlanguages" value="ja,ko,szh,zh-
simplified,tzh,zh-traditional"/>
    <parameter name="newdefaultindex" value="lemcontent"/>
  </parameter-list>
</instance>
```

3 Add the line `<parameter name="queryexpansion" value="true"/>` in the following section of the configuration file:

```
<instance name="packfql" type="internal" resource="FastQT_PackFQL">
  <parameter-list>
    <parameter name="indexcf" value="index.cf"/>
    <parameter name="rootfieldprops" value="etc/qrserver/
parseindex.map"/>
    <parameter name="trueindex" value="alwaystrue"/>
    <parameter name="trueterm" value="true"/>
    <parameter name="queryexpansion" value="true"/>
  </parameter-list>
</instance>
```

4 Restart the FAST Query and Result Engine in order to activate this configuration change.

Chapter 7

Adding Crawler or Document Processor Nodes

This chapter describes how to add a crawler or document processing node to/from a FAST InStream installation. Adding a node involves:

- installing a new instance of FAST InStream to be merged into the existing system configuration
- registering the new server
- configuring the desired node into the system

The original installation can be a single node installation or a multiple node installation. The node in the original installation that is running the administration modules such as the FAST Configuration Server will be referred to in this procedure as the *Administration Node*. The new crawler node is part of a single node installation and will be referred to as the *Crawler Node*.

Adding a Crawler Node

This procedure installs a new instance of FAST InStream to utilize its crawler.

1. If FAST InStream is not already installed on the Administration Node, install it now.
2. Install FAST InStream on the Crawler Node as a single node installation. The basic installation program automatically installs the crawler along with the other FAST InStream modules.
3. Start FAST InStream on the Crawler Node.
4. Verify that the crawler has been successfully installed on the Crawler Node. To do so:
    - **a** Open the administration page on the Crawler Node. The page is located at *http://<crawler_node_hostname>:<base port + 3000>*.
    - **b** Check the **System Management Page** to be sure there is a crawler installed.
5. If the crawler is not installed on the Crawler Node, then install it by going to the Control Panel section on the **System Management Page** and selecting ⊞ **Add Crawler**.
6. Remove the **Document Processor** from the Crawler Node. To do so, load the **Crawler Node System Management Page** and select ⊟ **Remove Doc. Processor**.
7. If there is a crawler running on the Administration node, remove it.
    - **a** Start FAST InStream on the Administration node if it is not running.
    - **b** Load the Administration Node administration page *http://<administration_node_hostname>:<base port + 3000>*.
    - **c** Select the **System Management** page.
    - **d** Select ⊟ **Remove Crawler** on the **System Management** page.
8. Shut down FAST InStream on the Crawler Node.
9. Start up FAST InStream on the Crawler Node.
    - **a** Copy the files *$FASTSEARCH/etc/LoggerConfig.xml*, *$FASTSEARCH/etc/CSLocation.xml* and *$FASTSEARCH/etc/omniorb.cfg* from the administration node to the corresponding location on the crawler node.

- **b** Edit the NodeConf.xml to comment out or remove all processes not needed by the crawler. These processes are:

```
<proc>logserver</proc>
<proc>nameservice</proc>
<proc>contentdistributor</proc>
<proc>configserver</proc>
<proc>cachemanager</proc>
<proc>indexer</proc>
<proc>search-1</proc>
<proc>qrserver</proc>
<proc>statusserver</proc>
<proc>anchorserver</proc>
<proc>storageservice</proc>
<proc>j2ee</proc>
```

10 Verify that the Administration Node detects the Crawler Node. To do so:

- **a** From the Administration Node administration page, select the **System Overview** page.
- **b** Under the **Module List**, verify a NodeControl, DataSource (crawler), FileServer, and PostProcess are listed for your new Crawler Node.

Adding a Document Processor Node

This procedure installs a new instance of FAST InStream to utilize its document processor.

1. If FAST InStream is not already installed on the Administration Node, install it now.
2. Install FAST InStream on the Document Processor Node as a single node installation. The basic installation program automatically installs the Document Processor along with the other FAST InStream modules.
3. Start FAST InStream on the Document Processor Node.
4. Verify that the document processor has been successfully installed on the Document Processor Node. To do so:

   a. Open the administration page on the Document Processor Node. The page is located at *http://<document_processor_node_hostname>:<base port + 3000>*.

   b. Check the **System Management Page** to be sure there is a document processor installed.
5. If a Document Processor is not installed on the Document Processor Node, then install it by going to the Control Panel section on the **System Management Page** and selecting ⊞ **Add Document Processor**. Adding a Document Processor adds the Document Processor configuration to the *NodeState.conf* file.
6. Remove the **Crawler** from the Document Processor Node. To do so, load the **Document Processor Node System Management Page** and select ⊟ **Remove Crawler**.
7. If there is a document processor running on the Administration node, remove it.

   a. Start FAST InStream on the Administration node if it is not running.

   b. Load the Administration Node administration page *http://<administration_node_hostname>:<base port + 3000>*.

   c. Select the **System Management** page.

   d. Select ⊟ **Remove Document Processor** on the **System Management** page.
8. Shut down FAST InStream on the Document Processor Node.
9. Update the FAST InStream installation so that only the Document Processor is started.

   a. Copy the files *$FASTSEARCH/etc/LoggerConfig.xml*, *$FASTSEARCH/etc/CSLocation.xml* and *$FASTSEARCH/etc/omniorb.cfg* from the administration node to the corresponding location on the Document Processor node.

- **b** Edit the *NodeConf.xml* to comment out or remove all processes not needed by the Document Processor. These processes are:

```
<proc>logserver</proc>
<proc>nameservice</proc>
<proc>contentdistributor</proc>
<proc>configserver</proc>
<proc>cachemanager</proc>
<proc>indexer</proc>
<proc>search-1</proc>
<proc>qrserver</proc>
<proc>statusserver</proc>
<proc>anchorserver</proc>
<proc>storageservice</proc>
<proc>j2ee</proc>
```

10 Start up FAST InStream on the Document Processor Node.

11 Verify that the Administration Node detects the Document Processor Node. To do so:

- **a** From the Administration Node administration page, select the **System Overview** page.
- **b** Under the **Module List**, verify a NodeControl and Document Processor are listed for your new Document Processor Node.

Chapter 8

Rank Tuning Bulk Loader

Custom rank tuning of documents using an XML file configuration file is possible through the *Rank Tuning Bulk Loader* (*rtbulktool*) program.

The *Rank Tuning Bulk Loader* can be used to import rank boosting specifications for individual documents into an existing search index. It reads boost records from an XML file and applies the boosts to the specified documents.

The *Rank Tuning Bulk Loader* is a convenient way to boost many queries or documents in one go. You might for example maintain your own list of preferred landing pages in a database or a spreadsheet. The loader allows you to apply this list to your FAST InStream implementation in one operation.

You can also use the *Rank Tuning Bulk Loader* to take a backup of all boost records in FAST InStream.

The *Rank Tuning Bulk Loader* provides the same rank boosting features as the optional FAST Business Manager Control Panel, but is not GUI-based.

Refer to Appendix F *Rank Tuning Bulk Loader Usage* for usage and options descriptions.

This chapter includes:

- Importing Boost Entries into the Rank Tuning Database
- Exporting/ Backing Up Boost Entries from the Rank Tuning Database
- Deleting Entries in the Rank Tuning Database
- rank-tuning-values Attributes

Importing Boost Entries into the Rank Tuning Database

Importing boost entries into the rank tuning database is useful if you already have a collection of preferred landing pages that you want to apply to the FAST InStream index.

To import boost entries into the rank tuning database, enter:

```
rtbulktool -w -f <xmlfile with boost entries to import> -u <user> -p
<password>
```

The XML file format is as follows:

```
The XML file format is as follows:

<?xml version="1.0"?>
<rank-tuning-values>
    <document-boost boost="1000">
        <content-id><![CDATA[http://www.test.com/]]></content-id>
        <collection-id>foo</collection-id>
    </document-boost>
    <query-position>
        <content-id><![CDATA[http://www.tester.org/2]]></content-id>
        <collection-id>foo</collection-id>
        <query position="2"><![CDATA[ranktuning]]></query>
    </query-position>
    <query-position>
        <content-id><![CDATA[http://www.moretest.com/
doit?test=this&foo=bar]]></content-id>
        <collection-id>bar</collection-id>
        <query position="1"><![CDATA[matrix]]></query>
    </query-position>
    <query-boost>
        <content-id><![CDATA[http://www.mytest.com/]]></content-id>
        <collection-id>foo</collection-id>
        <query boost="1000"><![CDATA[vespa]]></query>
    </query-boost>
    <query-boost>
        <content-id><![CDATA[http://www.tested.com/1]]></content-id>
        <collection-id>bar</collection-id>
        <query boost="3000"><![CDATA[bmcp]]></query>
    </query-boost>
</rank-tuning-values>
```

Note! It is possible to import the XML format from FAST InStream 3.x using the -l option.

Exporting/ Backing Up Boost Entries from the Rank Tuning Database

Exporting/backing up boost entries is useful for:

- storing a backup file of the boost entries
- duplicating the rank tuning database in case you use a multi-node setup with more than one document processor node
- obtaining an overview of all entries in the rank tuning database, across all collections.

To export/backup boost entries from the rank tuning database, enter:

```
rtbulktool -r -f <name of file where entries will be stored> -u <user> -p
<password>
```

Deleting Entries in the Rank Tuning Database

This command deletes all boost entries in all collections.

To delete all entries in the rank tuning database, enter:

```
rtbulktool -d -u <user> -p <password>
```

Note that this command deletes all boost entries in the rank tuning database only. It does not remove the documents from the searchable index; you must manually remove documents from the searchable index. Complete the following if you want to manually delete a document from the searchable index.

1. From the **Collection Overview** screen, click the **Edit** button for the collection you want to delete.
2. Click on the **Delete Document** button located in the Control Panel.
3. Enter the URL to the document you want to remove. Click **OK**.

rank-tuning-values Attributes

Table 8-1 <*rank-tuning-values*> Attributes

Element	Attribute	Description
content-id		The id of the document.
collection-id		The id of the collection.
document-boost	boost	Relative document boost that is added to the total rank score. *Valid range = -1000 to 100000*
query-position		Absolute query boost, hard-wiring to a given position, ('result order') in the result set. Hard-wired documents will appear as the top hits for a given query. If two documents are given the same result order ('position') for a query, then the normal rank score will decide the ordering. *Valid range of result orders:* *1-10* *0* =specifying position (excludes the document from the result set for the given query)
query-boost		Relative query boost, added to the total rank score for the given query. *Valid range: Minimum boost = 1 points, maximum boost =61505 points*
query	boost	Refer to query-boost element.
query	position	Refer to query-position element.

Appendix A

Administrator Interface Module and Process Naming Conventions

This appendix lists all modules and processes that are displayed in the FAST InStream Administrator Interface on the **System Overview** and **System Management** tabs (Table A-1).

Table A-1 FAST InStream Module and Process Names

Module or Process Name	Description
Anchor Server AnchorServer Anchor Server Storage anchorserver-storage	This server keeps track of links between documents and associated anchor texts with link target documents. It is used during document processing to provide input for link-based authority ranking.
Cache Manager cachemanager	Persistent storage for document-processing stages. The primary function of this server is to store rank boosts set in the Business Manager's Control Panel, but it can also be used to store other document attributes. This server is used during document processing.
Configuration Server Config Server configserver	The central configuration repository for all FAST InStream modules. It provides all modules, with the exception of the Log Server, with the system information they need.
Content Distributor docdistributor contentdistributor	A submodule of the FAST Document Processing Engine. It is responsible for dispatching incoming documents to the right document processing pipelines.
Document Processor procserver Processor Server ProcessorServer	Process for managing the document-processing pipeline. It processes submitted content according to the specified document-processing pipeline by running it through the relevant document-processing stages.

Table A-1 FAST InStream Module and Process Names

Module or Process Name	Description
Enterprise Crawler DataSource (Enterprise Crawler)	An instantiation of the FAST Enterprise Crawler.
J2EE	This is the framework where the Business Manager's Control Panel runs.
License Manager lmgrd	License server for all components controlled by the licensing scheme.
Log Server Log Server logserver	The FAST Log Server is a module that receives log output from all FAST InStream modules. The Log Server sends the log output to different log files. The log files are parsed and presented on the **Logs** screen of the FAST InStream Administrator Interface each time the screen is accessed.
Name Server Name Service nameservice	Systeminternal service that keeps track of processes and process interfaces.
Node Controller NodeControl nctrl	The Node Controller starts and stops the entire system on the machine it is running on. In addition, it ensures that all modules on its machine are running. There is one Node Controller per machine in the system. All functionalities provided by the Node Controller are only executed on the node where the Node Controller instance is running. If a Node Controller instance is not listed in the drop-down list on the **System Management** screen, this may indicate that either this Node Controller instance is not properly registered with the FAST Configuration Server or that there is no Node Controller instance running on that machine.
Post Process PostProcess postprocess	A submodule of the FAST Enterprise Crawler responsible for post-processing crawled content and feeding it to the document processing pipeline.
QRServer qrserver Search Dispatcher (QRserver) Search Dispatcher	An instantiation of the FAST Query and Result Engine.

Table A-1 FAST InStream Module and Process Names

Module or Process Name	Description
RTS Indexer Search Engine (RTS Indexer) indexer	A submodule of the FAST Search Engine responsible for building the index structures facilitating high-performance search from the input content.
Real-Time Search RTS Search rtsearch Search Engine (RTS Search) search	This is the core search process. It uses the index structures built by the RTS Indexer to provide matches for queries.
RTSDispatch Search Engine (FDispatch) RTS Top Dispatcher Search Dispatcher topfdispatch	The RTS Dispatcher is a sub-module of the FAST Search Engine. It is responsible for merging search results from different Search Engines, as well as potentially load-balance between them.
Status Server StatusService	Monitors the status of a document with respect to document processing and indexing.
Storage Service	This module provides persistent storage for the Business Manager's Control Panel.
Web Server httpd	Internal web server to provide the administration GUI for FAST InStream.

Appendix B

Searchable Characters

This appendix lists the characters that are searchable in FAST InStream.

Note! Non-searchable characters are regarded as word delimiters when matching queries against documents. They are, however, included in alphanumeric sorting and result presentation.

In general, FAST InStream allows you to search for characters with the following Unicode properties:

- Alphabetic
- Diacritic

  Characters that linguistically modify the meaning of another character to which they apply.
- Extender

  Characters whose principal function is to extend the value or shape of a preceding alphabetic character. These are typically length and iteration marks.
- Nd

  decimal numbers

- In addition, FAST InStream is able to index a custom set of characters to enclose CJK letters and months. These are derived from a number of custom character ranges with Bidirectional property (3200-32FF range).

  The specific character ranges are:

  - 3200-3243
  - 3260-327B
  - 327F-32B0
  - 32C0-32CB
  - 32D0-32FE

  For more information, see *http://www.unicode.org/charts/PDF/U3200.pdf*

For details on Unicode properties, see *http://www.unicode.org/Public/UNIDATA/UCD.html#Properties*.

Appendix C

Configuration Files and DTD

This chapter includes the complete files for the following FAST InStream configuration files:

- *default.xml* (the default index profile with lemmatization disabled)
- *index profile.dtd* (the DTD definition for the index profile)
- *FastXML dtd*

The *default* index profile may be found in the directory:

UNIX: *$FASTSEARCH/index-profiles/*
Windows: *%FASTSEARCH%\index-profiles\*

where *FASTSEARCH* indicates the environment variable set to the FAST InStream installation directory.

The *DTD* is also located in this directory and is named *index-profile-x.x.dtd* (where x.x signifies the specific version, for example *index-profile-3.1.dtd)*

Index Profile Configuration File

The following file is the *default.xml* index profile:

```
<?xml version="1.0"?>
<!DOCTYPE index-profile SYSTEM "index-profile-3.2.dtd">
<index-profile name="datasearch">
<field-list>

  <field name="title" fullsort="yes" tokenize="auto">
    <vectorize default="10:0"/>
  </field>
  <field name="body" tokenize="auto" max-result-size="1024"
         fallback-ref="teaser" result="dynamic" index="no">
         <vectorize default="5:5" alternative="{ja,ko,zh,szh,tzh}:5:0"/>
  </field>
  <field name="teaser" index="no"/>
  <field name="headings" tokenize="auto" />
  <field name="description" result="no" />
  <field name="anchortext" result="no" tokenize="auto" />
  <field name="keywords" result="no" />
  <field name="contenttype" element-name="mime" />
  <field name="format" index="no"/>
  <field name="language" />
  <field name="languages" separator=";" />
  <field name="charset" />
  <field name="urls"/>
  <field name="url" index="no"/>
  <field name="domain" element-name="url.domain" result="no" />
  <field name="tld" element-name="url.tld" result="no" />
  <field name="path" element-name="url.path" result="no" />
  <field name="alwaystrue" result="no"/>

<!--  Non-text fields -->
  <field name="crawltime" type="datetime" fullsort="yes" />
  <field name="processingtime" type="datetime" fullsort="yes" />
  <field name="docdatetime" type="datetime" fullsort="yes" />
  <field name="size" type="int32" fullsort="yes"/>
  <field name="generic1" />
  <field name="generic2" />
  <field name="generic3" />
  <field name="generic4" result="no" />
  <field name="igeneric1" type="int32" fullsort="yes" />
  <field name="igeneric2" type="int32" fullsort="yes" />
  <field name="dtgeneric1" type="datetime" fullsort="yes" />
  <field name="dtgeneric2" type="datetime" fullsort="yes" />

<!-- News Entity Extraction Fields -->
  <field name="companies" separator=";" />
  <field name="locations" separator=";" />
  <field name="personnames" separator=";" />
  <field name="topics" separator=";" />
  <field name="emails" separator=";" />
  <field name="taxonomy" />
  <field name="host" separator=";" />

  <!--=========== Fields in the subscopes composite field ===========-->
  <!--== These fields may be used for boosting selected sub-scopes ==-->
  <field name="scopeboosthigh" result="no" tokenize="auto">
    <vectorize default="5:5" alternative="{ja,ko,zh,szh,tzh}:5:0" />
  </field>
  <field name="scopeboostmedium" result="no" tokenize="auto" />
  <field name="scopeboostlow" result="no" tokenize="auto" />

</field-list>

<scope-field-list>

<!-- A Scope Search root scope -->
  <scope-field name="xml" result="dynamic" element-name="data">
    <vectorize default="5:5" alternative="{ja,ko,zh,szh,tzh}:5:0" />
  </scope-field>

</scope-field-list>

<composite-field name="content" rank="yes" default="yes" query-
tokenize="auto">
  <field-ref name="body" level="1"/>
  <field-ref name="headings" level="2"/>
  <field-ref name="path" level="2"/>
  <field-ref name="description" level="2"/>
  <field-ref name="domain" level="3"/>
  <field-ref name="keywords" level="3"/>
  <field-ref name="title" level="4"/>
  <field-ref name="anchortext" type="external" level="5"/>

  <rank-profile name="default" rank-model="default">
     <authority weight="50" field-ref="anchortext" />
     <freshness weight="50" field-ref="docdatetime" auto="yes" />
     <proximity weight="50" />
     <context weight="50">
<field-weight field-ref="body" value="5"/>
<field-weight field-ref="headings" value="20"/>
<field-weight field-ref="path" value="20"/>
<field-weight field-ref="description" value="30"/>
<field-weight field-ref="domain" value="50"/>
<field-weight field-ref="keywords" value="50"/>
<field-weight field-ref="title" value="60"/>
     </context>
  </rank-profile>
  <rank-profile name="news" rank-model="news">
     <authority weight="50" field-ref="anchortext" />
     <freshness weight="200" field-ref="docdatetime" auto="yes" />
     <proximity weight="50" />
     <context weight="50">
<field-weight field-ref="body" value="5"/>
<field-weight field-ref="headings" value="20"/>
<field-weight field-ref="path" value="20"/>
<field-weight field-ref="description" value="30"/>
<field-weight field-ref="domain" value="50"/>
<field-weight field-ref="keywords" value="50"/>
<field-weight field-ref="title" value="60"/>
     </context>
  </rank-profile>

  <rank-profile name="site" rank-model="site">
     <authority weight="70" field-ref="anchortext" />
     <freshness weight="50" field-ref="docdatetime" auto="yes" />
     <proximity weight="60" />
     <context weight="70">
<field-weight field-ref="body" value="5"/>
<field-weight field-ref="headings" value="20"/>
<field-weight field-ref="path" value="20"/>
<field-weight field-ref="description" value="30"/>
<field-weight field-ref="domain" value="50"/>
<field-weight field-ref="keywords" value="50"/>
<field-weight field-ref="title" value="60"/>
     </context>
  </rank-profile>

</composite-field>

<!-- Composite field used for boosting selected sub-scopes -->
<composite-field name="subscopes" rank="yes" default="no"
                 query-tokenize="auto">
  <field-ref name="scopeboostlow" level="1" />
  <field-ref name="scopeboostmedium" level="2" />
  <field-ref name="scopeboosthigh" level="3" />
  <field-ref name="anchortext" type="external" level="4" />

  <!-- Default rank profile for Scope Search -->
  <rank-profile name="scopedefault" rank-model="scopedefault">
     <authority weight="50" field-ref="anchortext" />
     <freshness weight="50" field-ref="docdatetime" auto="yes" />
     <proximity weight="50" />
     <context weight="50">
       <field-weight field-ref="scopeboostlow" value="10" />
       <field-weight field-ref="scopeboostmedium" value="20" />
       <field-weight field-ref="scopeboosthigh" value="35" />
       <scope-weight field-ref="xml" value="20" />
     </context>
  </rank-profile>
</composite-field>

<result-specification>

  <categorization name="default" sort-by="label">
    <field-ref name="taxonomy"/>
  </categorization>

  <clustering name="default" sort-by="none" size="10" threshold="0.30"/>

<!-- Result proximity boosting: Set to "yes" to enable boosting per
default -->
<!--   set to "no" to generate necessary config to allow boosting on a per
-->
<!--   query basis but have boosting off per default -->
  <result-proximity boost="no">
    <field-ref name="body"/>
    <field-ref name="title"/>
    <field-ref name="anchortext"/>
  </result-proximity>

  <numeric-navigator name="sizenavigator"
                     display="Size"
                     unit="kB"
       divisor="1024"
                     intervals="4"
                     resolution="1024">
    <field-ref name="size"/>

    <range-label type="first" format="Less than %.0f" />
    <range-label type="middle" format="Between %.0f and %.0f" />
    <range-label type="last" format="More than %.0f" />

    <ignore-value value="0"/>
  </numeric-navigator>

  <numeric-navigator name="docdatetimenavigator"
                     display="Document Time"
                     unit="Date"
                     intervals="4"
                     resolution="1440">
    <field-ref name="docdatetime"/>

    <range-label type="first" format="Before %.10s" />
    <range-label type="middle" format="Between %.10s and %.10s" />
    <range-label type="last" format="%.10s or after" />
  </numeric-navigator>
  <string-navigator name="contenttypenavigator" display="MIME">
    <field-ref name="contenttype"/>
  </string-navigator>

  <string-navigator name="charsetnavigator" display="Encoding">
    <field-ref name="charset"/>
  </string-navigator>

  <string-navigator name="languagesnavigator"
                    display="Languages">
    <field-ref name="languages"/>
  </string-navigator>

<!-- News Entity Navigators -->
  <string-navigator name="companiesnavigator" display="Companies">
    <field-ref name="companies"/>
  </string-navigator>

  <string-navigator name="locationsnavigator" display="Locations">
    <field-ref name="locations"/>
  </string-navigator>

  <string-navigator name="personnamesnavigator" display="Names">
    <field-ref name="personnames"/>
  </string-navigator>

  <string-navigator name="topicsnavigator" display="Topics">
    <field-ref name="topics"/>
  </string-navigator>

  <string-navigator name="emailsnavigator" display="Emails">
    <field-ref name="emails"/>
  </string-navigator>

  <string-navigator name="hostnavigator" display="Hosts">
    <field-ref name="host"/>
  </string-navigator>

  <result-view name="urls">
   <field-ref name="url"/>
   <field-ref name="urls"/>
  </result-view>

</result-specification>

</index-profile>
```

Index Profile DTD

The following file is the *index profile.dtd*:

```
<!--  (c) 2004 Fast Search & Transfer. All rights reserved.  -->
<!--  Version 3.2 -->
<!--  Please refer to the Data Search documentation -->

<!ELEMENT index-profile  (field-list, scope-field-list?, composite-
field*, result-specification?, xml-filepatch*, txt-filepatch*) >

<!ATTLIST index-profile
          name        CDATA        #REQUIRED
          compress    ( yes | no ) "yes" >

<!ELEMENT tokenize EMPTY>
<!ATTLIST tokenize
          type          ( auto | delimiters )          "delimiters"
          parameter              CDATA                 ""
          output        ( tokens | readings |
                          none )                       "tokens"
          field-ref              CDATA                 "" >

<!ELEMENT vectorize EMPTY>
<!ATTLIST vectorize
          default                CDATA                 "10:0"
          alternative            CDATA
"{ja,zh,szh,tzh,ko}:10:0" >

<!ELEMENT fullsort EMPTY>
<!ATTLIST fullsort
          profile                CDATA                 ""
          length                 CDATA                 "0" >

<!ELEMENT field-list (field+)>
<!ATTLIST field-list
          phrases                (yes | no)            "yes"
          positions              (yes | no)            "yes" >

<!ELEMENT field (tokenize?, vectorize?, fullsort?)>
<!ATTLIST field
          name                   CDATA                 #REQUIRED
          description            CDATA                 #IMPLIED
          element-name           CDATA                 #IMPLIED
          type          ( string | int32 |
                          float | double | datetime )  "string"
          latency         ( normal | low )             "normal"
          lemmatize         ( yes | no )               "no"
          sort              ( yes | no )               "no"
          fullsort          ( yes | no )               "no"
          result        ( static | dynamic | no )      "static"
          fallback-ref           CDATA                 #IMPLIED
          max-index-size         CDATA                 "1024"
          max-result-size        CDATA                 "64"
          index       (yes | no)                       "yes"
          datetime-resolution ( minute | second )      "minute"
          tokenize    (auto | delimiters )             "delimiters"
          vectorize            ( yes | no )            "no"
          substring              CDATA                 "0"
          wildcard         ( prefix | full )           "prefix"
          boundary-match       (yes | no)              "no"
          separator              CDATA                 "" >

<!ELEMENT scope-field-list (scope-field+)>

<!ELEMENT scope-field (query-tokenize?, tokenize?, vectorize?)>
<!ATTLIST scope-field
          name                   CDATA                 #REQUIRED
          description            CDATA                 #IMPLIED
          element-name           CDATA                 #IMPLIED
          type                (text)                   "text"
          lemmas              ( yes | no )             "no"
          phrases             ( yes | no )             "no"
          tokenize          (auto | delimiters )       "delimiters"
          query-tokenize   ( auto | delimiters )       "delimiters"
          vectorize            ( yes | no )            "no"
          wildcard             (yes)                   "yes"
          result        ( static | dynamic | no )      "no"
          datetime-resolution ( minute | second )      "minute"
          fallback-ref           CDATA                 #IMPLIED
          max-result-size        CDATA                 "1024" >

<!ELEMENT field-ref EMPTY>
<!ATTLIST field-ref
          name        CDATA             #REQUIRED
          type       (normal|external)  "normal"
          level       CDATA             "1" >

<!ELEMENT field-ref-group (field-ref+)>
<!ATTLIST field-ref-group
          name        CDATA             #REQUIRED
          type        (append)          "append"
          level       CDATA             "1">

<!ELEMENT query-tokenize EMPTY>
<!ATTLIST query-tokenize
          type                ( auto | delimiters )    "delimiters"
          parameter              CDATA                 "" >
```

```
<!ELEMENT composite-field (query-tokenize?, field-ref*, field-ref-group*,
rank-profile*) >
<!ATTLIST composite-field
          name        CDATA                       #REQUIRED
          type        (text)                      "text"
          default     (yes | no)                  "no"
          wildcard    (yes | no | prefix | full)  "prefix"
          substring   CDATA                       "0"
          rank        (yes | no)                  "no"
          lemmas      (yes | no)                  "no"
          phrases     (yes | no)                  "yes"
          positions   (yes | no)                  "yes"
          query-tokenize   ( auto | delimiters )  "delimiters"
          query-boundary-match  (yes | no)        "no" >

<!ELEMENT rank-profile (authority, freshness, proximity, context) >
<!ATTLIST rank-profile
          name         CDATA             #REQUIRED
          rank-model   CDATA             #REQUIRED >

<!ELEMENT authority EMPTY >
<!ATTLIST authority
          weight        CDATA            #REQUIRED
          field-ref     CDATA            #IMPLIED >

<!ELEMENT freshness EMPTY >
<!ATTLIST freshness
          weight        CDATA            #REQUIRED
          field-ref     CDATA            #REQUIRED
          auto         (yes | no)        "no" >

<!ELEMENT proximity EMPTY >
<!ATTLIST proximity
          weight        CDATA            #REQUIRED >

<!ELEMENT context (field-weight*, scope-weight*)>
<!ATTLIST context
          weight       CDATA             #REQUIRED >

<!ELEMENT field-weight EMPTY >
<!ATTLIST field-weight
          field-ref    CDATA             #REQUIRED
          value        CDATA             #REQUIRED >

<!ELEMENT scope-weight EMPTY >
<!ATTLIST scope-weight
          field-ref    CDATA             #REQUIRED
          value        CDATA             #REQUIRED >
```

```
<!ELEMENT result-specification (result-filter*, categorization?,
clustering?, field-collapsing?, bigram-boost?, result-proximity?,
numeric-navigator*, string-navigator*, result-view*) >

<!ELEMENT clustering EMPTY>
<!ATTLIST clustering
          name            CDATA                           #REQUIRED
          algorithm       CDATA                           "bestfit"
          threshold       CDATA                           "0.30"
          depth           CDATA                           "1"
          size            CDATA                           "5"
          labels          CDATA                           "3"
          join            ( yes | no )                    "yes"
          sort-by  ( none | size | label )         "none" >

<!ELEMENT categorization (field-ref)>
<!ATTLIST categorization
          name            CDATA                           #REQUIRED
          sort-by  ( none | size | label )         "none" >

<!ELEMENT field-collapsing (field-ref)>
<!ATTLIST field-collapsing
          name       CDATA           #REQUIRED >

<!ELEMENT result-proximity (field-ref+)>
<!ATTLIST result-proximity
          boost (yes|no)   "no" >

<!ELEMENT bigram-boost EMPTY>
<!ATTLIST bigram-boost
          maximum-phrases      CDATA                      "20" >

<!ELEMENT result-filter (field-ref)>
<!ATTLIST result-filter
          name            CDATA                           #REQUIRED
          type            ( duplicate )                   #REQUIRED >

<!ELEMENT numeric-navigator ( manual-cut*, field-ref, range-label*,
ignore-value*)>
<!ATTLIST numeric-navigator
          name            CDATA                           #REQUIRED
          display         CDATA                           #IMPLIED
          algorithm       ( equalfrequency |
                            equalwidth |
                            manual |
                            rangedivision )              "equalfrequency"
          unit            CDATA                           #IMPLIED
          resolution      CDATA                           "1"
          intervals       CDATA                           "4"
          divisor         CDATA                           "1"
          deep            CDATA                           "yes"
          default-value   CDATA                           ""
          default-ignore-value CDATA                      "" >

<!ELEMENT manual-cut EMPTY>
<!ATTLIST manual-cut
          value           CDATA                           #REQUIRED >

<!ELEMENT range-label EMPTY>
<!ATTLIST range-label
          format          CDATA                           "auto"
          type            (first|middle|last)             #REQUIRED
          offset          CDATA                           "0" >

<!ELEMENT string-navigator (field-ref, ignore-value*)>
<!ATTLIST string-navigator
          name             CDATA                          #REQUIRED
          display          CDATA                          #IMPLIED
          unit             CDATA                          #IMPLIED
          sort-by          ( auto | name |
                             frequency | number )         "auto"
          sort-order       ( descending | ascending )     "descending"
          filter-buckets   CDATA                          "100"
          filter-frequency CDATA                          "1"
          deep             CDATA                          "yes"
          cutoff-frequency CDATA                          "0"
          cutoff-frequency-exception-minbuckets CDATA     "0"
          cutoff-maxbuckets CDATA                         "1000"
          anchoring        ( auto | none | complete |
                             prefix | suffix )            "auto"
          default-value    CDATA                          ""
          default-ignore-value CDATA                      "" >

<!ELEMENT ignore-value EMPTY>
<!ATTLIST ignore-value
          value CDATA #REQUIRED >

<!ELEMENT result-view (field-ref)+ >
<!ATTLIST result-view
          name        CDATA               #REQUIRED
          description CDATA               #IMPLIED>
```

FastXML DTD

The following file is the *FastXML.dtd*

```
<!-- File level: a sequence of one or more FastXML documents -->
<!ELEMENT documents (document+)>

<!-- Single FastXML document, consists of at least one content element -->
<!ELEMENT document (element+)>
<!ATTLIST document
          id CDATA #REQUIRED>

<!-- each element may have one or more meta-info subelements, and must
     have a value. -->
<!ELEMENT element (meta-info*, value)>

<!-- elements must be named, preferably with an application-specific
     prefix. (e.g, "fast.")
     this is to avoid conflict with a number of predefined element tags,
     some of which we may want to use in the FastXML format directly,
     anyhow. Examples: "getpath", "data", ... -->
<!ATTLIST element
          name CDATA #REQUIRED>

<!-- meta-info may be used to give hints to the FAST DS processing
pipeline.
     Examples: indicate the language of a piece of text, possible encoding
     (e.g, "base64"). -->
<!ELEMENT meta-info EMPTY>

<!-- Required args for meta-info elements are name and value. -->
<!ATTLIST meta-info
          name CDATA #REQUIRED
          value CDATA #REQUIRED>

<!-- "value" is the required sub-element of "element" -->
<!ELEMENT value (#PCDATA)>
```

Appendix D

Index Profile Elements

This appendix describes the structure of the index profile; it identifies and describes the *field*, *scope-field*, *composite-field*, and *result-specification* elements and their respective attributes within the profile.

The default index profile consists of four main XML elements or blocks:

```
<index-profile name="datasearch">
  <field-list>
  <scope-field-list>
  <composite-field>
  <result-specification>
</index-profile>
```

- The `<field-list>` contains a list of all defined fields within the index, including information about mapping from the corresponding document processing elements, indexing and search attributes related to individual fields.
- The `<scope-field-list>` contains a list of all defined scope fields within the index, including information about indexing and search attributes related to individual scope fields.
- One or more `<composite-field>` specifications define combined *field* references with configured relevance score per field. Note that the maximum number of entries in a `composite-field` is eight.
- The `<result-specification>` contains a set of configuration rules for handling of the search result.

Note! When defining the index profile, use only lowercase alphanumeric ASCII characters in the *field* names.

Refer to Appendix C *Configuration Files and DTD* for complete configuration and DTD files. The index profile files are configuration files while the DTD is a specification of how to write a configuration file.

Index Compression

Compression of the index is configured at the top level in the index-profile element. By default, the compression is enabled:

<index-profile name="datasearch" compress="yes">

Compression typically reduces the size of the *FASTSEARCH/data/data_index* catalog by 20% to 50%. It can be disabled:

<index-profile name="datasearch" compress="no">

Field-list Element

This section provides detailed descriptions for the individual attributes associated with the *field-list* element.

This element is used to list all individual fields that are defined within the index profile. The individual fields are included according to the format as described in the *Field Element and Attributes* subsection. The following table lists the attributes associated with the *field-list* element:

Table D-1 *field-list* Attributes

Attribute	Description
phrases	Enables phrase query support in the index for all string-type fields in this *field-list*. Disabling phrase support reduces the size of the search index on disk. *Valid values: yes, no* *Default: phrases="yes”* Note that several features require phrase support to be enabled. This includes Phrase queries, Automatic phrase detection, Substring search, Boundary match and Bigram boost.
positions	Enables proximity support (implicit proximity rank boosting and explicit proximity operators 'NEAR' and 'ONEAR') in the index for all string-type fields in this *field-list*. Disabling proximity support (the positions index) reduces the size of the search index on disk. *Valid values: yes, no* *Default: positions="yes”*

Field Element and Attributes

This subsection provides detailed descriptions for the attributes associated with the *field* element.

A field defines an attribute of a document, such as a *title* or *body.* A *field* is searchable by default, and is also part of the default *result-view.*

The *field* element may also contain *tokenize, vectorize*, and/or *full-sort* sub-elements. These sub-elements and their attributes are described in *Field Sub-Elements and Attributes* on page 186.

The following table lists the attributes associated with the *field* element.

Table D-2 *field* Attributes

Attribute	Description
name	**Required**: Name of the field. *Valid values:* Lowercase alphanumeric characters (for example*, title, body, teaser, headings, description, anchortext, keywords, format, contenttype, language, charset, url, domain, tld, path)* Reserved names not to be used: *collection, docvector, docid, morehits, doctype, anchor*, hwboost, links* Avoid using names with one of the prefixes: *elem**, *vec**, *lem**, *res**, *nco**, or *read**. If these prefixes are used, ensure that the suffix part of the name does not appear as a unique field name. For example, do not define two fields as *lemmings* and *mings*. *Default:* empty
description	Optional description of field. *Valid values:* Any text string *Default:* empty

Table D-2 *field* Attributes

Attribute	**Description**
type	Type of field for sorting. *Valid types are:* *string* - a free-text string field *int32* - a 32 bit signed integer *float* - decimal number (represented by 32 bits) *double* -decimal number with extended range (represented by 32 bits) *datetime* - date and time (ISO-8601 format) Refer to the *Content Integration Guide*, chapter *The FAST Data Search Document Model* for details on supported formats on the content access side. Refer to the *Query Language and Query Parameters Guide* for details on supported formats within the FAST InStream query languages. *Examples:* *<field name="modified" type="datetime" />* *<field name="size" type="int32" />* *<field name="income" type="double" />* *Default: type=“string”*
lemmatize	Enables lemmatization and synonym queries for this field. This configuration is applied to the corresponding document processing pipeline. *Valid values: no, yes* *Example: <field name=“body” lemmatize=“yes”/>* *Default: lemmatize="no”* Note that lemmatization requires that language dependent tokenization is also specified for the field. To do this, specify *tokenize="auto"* for the field. Also, lemmatization is only applied to composite fields. To apply lemmatization, you need to (1) apply *lemmas="yes"* to the *composite-field* (refer to *Composite-field Attributes* on page 199), and (2) apply *lemmatize="yes"* to the applicable fields within the *composite-field*.
sort	Specifies if a field should be configured for numeric, single-level sort only. Note that this attribute enables the sorting form which was available in earlier versions of FAST InStream; sorting on the first four characters of words. If *yes*, then the query parameters *sortby=field* and *sortdirection={ascending|descending}* can be used. *Valid values: no, yes* *Default: sort="no"*

Table D-2 *field* Attributes

Attribute	**Description**
fullsort	Specifies if a field should be configured for full-string and multi-level sorting. This sorting configuration is more flexible than *sort* and is normally recommended to use unless performance is crucial. *Valid values:* *fullsort="no"* *fullsort="yes” (full-string)* *Example: <field name="address” fullsort="yes”>* *Default: fullsort="no”* By default, full-text sorting is based on plain ASCII sorting of the full string. An alternative representation is available, defining *fullsort* as a sub-element to the field element. Table D-5 provides format details.
result	Specifies how the field is to be returned in the search results. *Valid values:* *dynamic* - specifies that a dynamic teaser should be generated for the field and returned as part of the result. Refer to *Dynamic Teasers* on page 65 for more information. *no* - this field is not returned as part of a search result *Default: result="static"*
fallback-ref	For fields with *result=dynamic* you can specify a fallback field to use if the dynamic teaser generation on this field yields an empty result. This will also affect duplicate removal-type *result-filter*s. *Valid values:* field name (for example, *teaser*) *Default:* empty
max-index-size	Maximum number of kilobytes indexed for this field. The measured value equals the data as stored prior to indexing and is slightly larger than the effective number of kilobytes that is searchable for the field. *Valid values:* Positive integer 0 through 2 M *Default: max-index-size="1024"* (KB)

Table D-2 *field* Attributes

Attribute	Description
max-result-size	Maximum number of kilobytes that are used for results. For a field having *result=dynamic* this size refers to the size of the source data used for the dynamic teaser. *Valid values:* Positive integer 0 through 2097151 (0 through 2 GB) Note that FAST InStream utilizes Zlib compression of large result fields in order to save disk space. This compression is only enabled for fields with *max-result-size* > 64 (KB), e.g. larger than the default value. If you know that most documents have content of size 10-60 KB for a given field, and disk space reduction is desired, then you might consider setting attribute *max-result-size="65"* for that field, so that result-text compression is enabled on the field. *Default: max-result-size="64"* (KB)
index	Determines if field will be part of the index and searchable. *Valid values*: *yes* - the field will be a part of the index and is searchable. A field that is not indexed can still be included in *composite-field* or just returned in the result. *no* - used if you want the field included in the result but you do not want to search for it. This value is used for fields already included in *composite-fields*. This content is not indexed as a separate field, but part of the *composite-field* where different field components may have a different rank contribution to the search result. *Default: index="yes"*
datetime-resolution	This specifies whether the *datetime* type fields uses a resolution in seconds or minutes within the search index. This impacts the granularity and the earliest date to be represented for datetime type fields. This attribute does not impact the *datetime* type format. *Valid values:* *second* - resolution in seconds since epoch 01/01/1970; earlier dates will be set to this date. *minute* - resolution in minutes since 01/01/1900; the seconds component in field and query are truncated. *Default: datetime-resolution="minute"*

Table D-2 *field* Attributes

Attribute	Description
tokenize	Enables language dependent tokenization (removing/normalizing punctuation, capital letters, word characters) at the field specific level. *Valid values:* *delimiters* - basic, western-style tokenization. The tokenization is performed after the document process pipeline, prior to indexing. This tokenization may apply for all pipelines. *auto* - automatic tokenization, handling both western and CJK tokenization. The tokenization is performed in the document processing pipeline, prior to lemmatization. This tokenization may be configured for all standard document processing pipelines except the *Express* pipeline. *Example: <field name="body" tokenize="auto">* *Default: tokenize="delimiters"* An alternative representation is available, defining *tokenize* as a sub-element to the field specification. The alternative representation is mainly applicable for CJK installation. Table D-3 provides format details. Tokenization on query side can also be specified and is configured in the *composite-field*, *query-tokenize* attribute. This applies to CJK (Asian language) applications. Tokenization on the query side will be the same as for the field when searching the index belonging to the field. For example, if field *title* exists, and the user searches using Japanese input *title:MyJapaneseText*, the term *MyJapaneseText* will be tokenized as Japanese. You must specify it for the *composite-field* again because fields with different tokenization settings can be included in the *composite-field*, and the user must specify how the *composite-field* queries should be tokenized.
vectorize	Enables creation of similarity vectors for the indicated field. Similarity vectors are used for the Unsupervised Clustering and Find Similar features. *Valid values: yes, no* *Example: <field name="body" vectorize="yes">* *Default: vectorize="no"* An alternative representation is available, defining *vectorize* as a sub-element to the field specification. The alternative representation can be used in order to enable non-standard configuration of the vectorization document processing. Table D-4 provides format details. Refer to *Configuring Similarity Vector Creation* on page 143 for details on advanced configuration of the similarity vector document processing.

Table D-2 *field* Attributes

Attribute	**Description**
substring	If specified, the *field* will be configured to support *substring* queries. Refer to *Substring Query Support* on page 67 for configuration information. If a field that has *substring* support is included in a *composite-field*, the text will not be directly substring searchable when searching the *composite-field*. Substring support must be specified for the *composite-field* for that to happen. If the *composite-field* has substring support, it does not matter whether or not the origin field has substring support. *Valid values:* positive integer *Default: substring="0"*
boundary-match	Determines if searches within a field can be anchored to the beginning and/or the end of the field. *Valid values: yes, no* *Example: <field name="title" boundary-match="yes"/>* *Default: boundary-match="no"*
separator	If more than one value is present in the source element for this string field (multi-value field), then the individual values should be separated by this character. This is used by the navigators for dynamic drill-down and for boundary match queries on multi-value string fields. Setting the *separator* attribute implicitly sets *boundary-match="yes"* even if *boundary-match* has already been set in the index profile for this field. *Example: #* ('#' is used when assigning category info, as this is the defined way to separate categories in the *'taxonomy'* field) *Default: " "*
element-name	Optional name of document element for this field. Only used if the element name is different from the field name. *Valid values:* Any document element name that is used in the source content or by optional custom document processing. *Default:* If not specified, the element is assumed to have the same name as the indicated field.

Table D-2 *field* Attributes

Attribute	Description
wildcard	Supports wildcard queries for this field. This attribute is only applicable for fields of type *string*. *Valid values:* *prefix* - supports *prefix* wildcard search only *(term*)*. This is a simpler algorithm with slightly better performance than *full* *full* - supports full wildcard search on entire word, using '*' and '?' where '*?*' means any one character and '*' means zero or more random characters (for example, **e?onic** should find all occurrences of Veronica) A '*' by itself is not allowed (searching for any word). *Default: wildcard="prefix"*

Field Sub-Elements and Attributes

A *field* element may include the following optional sub-elements (in the given order):

- *tokenize*
- *vectorize*
- *fullsort*

These sub-elements are described in the following subsections.

tokenize Sub-element

Table D-3 describes attributes for the *tokenize* sub-element. This sub-element can be used within a `<field>` specification to configure advanced tokenization parameters for CJK (Asian languages) installations.

Table D-3 *tokenize* Sub-element Attributes

Attribute	Description
type	Tokenization type. *Valid values: auto, delimiters* This parameter is included for future use. It must always be set to *"auto"* in order to use the advanced options below.
parameter	Defines if optional token decomposition is to be used for Japanese and Korean. *Valid values:* “ ” - empty, no decomposition *decompose* - decompose Japanese and Korean tokens *Example:* *<field name=“body”>* *<tokenize type=“auto” parameter=“decompose” output=“tokens”/>* *</field>* *Default: “ ”*
output	Determines if normal tokens or Japanese style readings forms are produced. *Valid values:* *tokens* - normal, tokenized output *readings* - output CJK-style readings forms *Example:* *<field name=“bodyreadings”>* *<tokenize type="auto" output=“readings” field-ref=“body”/>* *</field>* *Default: "tokens”*
field-ref	Optional specification of original field if Japanese readings forms are generated in a separate field (when using *'output="readings”'*) *Example:* Japanese readings for a field are put in a separate field using a *field-reference* to indicate the original field. Refer to the *Example* in the *output* attribute description.

vectorize Sub-element

Table D-4 describes attributes for the *vectorize* sub-element. This sub-element can be used within a `<field>` specification to configure similarity vector boost parameters.

You may create similarity vectors for multiple fields. The Unsupervised Clustering and Find Similar features will then be based on the vectors created for all these fields.

Table D-4 *vectorize* Sub-element Attributes

Attribute	Description
default	Defines the default boost configuration. *Format: weight:boost*, where *weight* = an integer that specifies the relative importance of the field (default weight for words within this sub-element). *boost* = an integer that defines the additional weight added to words that satisfy certain boosting criteria. The default boosting criterion is based on word capitalization considerations. *Default: 10:0*
alternative	Specification of *alternative* vectorization option for asian (CJK) languages. Use this attribute instead of *default* for the languages that require CJK tokenization. This attribute will only be effective when *tokenize=“auto”* is also specified for the field. *Format: language-list:weight:boost*, where *language-list* = a set of ISO language codes separated with ',', enclosed in '*{ }*' *Example: "{ja,ko}:10:0"* *Default: "{ja,zh,szh,tzh,ko}:10:0"*

You can use this feature to define different boost weight (using the *default* or *alternative* attribute) for the different fields, based on the relative importance of the fields within your content. The default index profile defines *10:0* for *title* and *5:5* for *body*. This implies that *title* words are twice as important as *body* words, unless the *body* words satisfy the boost criteria, in which case they are equally important. For example:

```
<field name=“body”>
   <vectorize default=“10:0” alternative=“{ja,zh,szh,tzh,ko}:10:0”/>
</field>
```

Refer to *Configuring Similarity Vector Creation* on page 143 for details on advanced configuration of the similarity vector feature including stopwords and boosting criteria.

fullsort Sub-element

Table D-5 describes attributes for the *fullsort* sub-element. This sub-element can be used within a `<field>` specification to indicate a custom sort profile for full text sorting. It enables definition of language dependant sorting mechanisms including correct multi-byte character sorting for languages that does not use ASCII-based sorting.

Table D-5   *fullsort* Sub-element Attributes

Attribute	Description
profile	Sort profile name. The profile name identifies a sub-directory with the desired alpha sorting profile. The sub-directory must be located in the *$FASTSEARCH/etc/alphasort* (UNIX) or *%FASTSEARCH%\etc\alphasort* (Windows) directory within your FAST InStream installation, and must contain a master configuration file *AlphaSortMasterFile.xml*. *Valid values:* A valid directory name without any path component. *Example: profile="myprofile"* indicates that the alpha sort profile located in *$FASTSEARCH/etc/alphasort/myprofile/* is used. *Default: If not specified, plain ASCII sorting is applied.* Note: The default alphanumeric sorting is based on ASCII sort order. A case-insensitive sort profile is also available. This is enabled using the following sort specification: *<field name="title">* *<fullsort profile="generic"/>* *</field>*
length	Integer value indicating the maximum number of bytes to be used for text sorting. This parameter can be used for performance optimization in case of large fields. *Valid values:* All positive values between 0 and 2097150 (2GB minus 1). A value "0" indicates no limitation. *Example:* *<field name="address" ...>* *<fullsort profile="norwegian" length="100">* *</field>* This example defines that the alpha sort profile located in *$FASTSEARCH/etc/alphasort/norwegian/* is used; maximum of *100* characters is basis for the sorting. Refer to *Text Sorting* on page 76 for details on how to create an alpha sort profile. *Default: length= “0” (entire field taken into consideration)*

<field> Example

The following example shows a snippet of a *field* definition in the index profile:

```
<field-list>
  <field name="title" sort="yes" fullsort="yes" tokenize="auto">
  </field>
  <field name="body" tokenize="auto" max-result-size="1024"
         fallback-ref="teaser" result="dynamic" index="no">
         <vectorize default="5:5" alternative="{ja,ko,zh,szh,tzh}:5:0"/>
  </field>
  <field name="teaser" index="no"/>
</field-list>
```

This example defines the fields *title*, *body*, *teaser* in the index.

The *title* field includes support for *full result sorting* and automatic *tokenization*.

The *body* field is configured for (*dynamic*) teaser generation, as defined in the subsection *Field Element and Attributes on page 180*. Maximum size of the *result* field is set to *1024* kilobytes, which limits the size of the result field on disk, and implies that the dynamic teaser generation is based on the first megabyte of text from the document only.

Scope-field-list Element

This section provides detailed descriptions for the individual attributes associated with the *scope-field-list* element.

This element is used to list all scope fields that are defined within the index profile. The scope fields are included according to the format as described in the *Scope-field Attributes* subsection.

The *scope-field* element may also contain *query-tokenize, tokenize,* and/or *vectorize* sub-elements.These sub-elements and their attributes are described in *Scope-field Sub-Elements and Attributes* on page 197.

Scope-field Attributes

The *scope-field* element consists of attributes listed in the following table.

Table D-6 *scope-field* Attributes

Attribute	Description
name	**Required**: Name of the *scope-field*. *Valid values:* Lowercase alphanumeric characters Reserved names not to be used: *collection, docvector, docid, morehits, doctype, anchor*, hwboost, links* Avoid using names with one of the prefixes: *elem*, vec*, lem*, res*, nco*, or read**. If these prefixes are used, ensure that the suffix part of the name does not appear as a unique field name. For example, do not define two fields as *lemmings* and *mings*. *Default:* empty
description	Optional description of *scope-field*. *Valid values:* Any text string *Default:* empty

Table D-6 *scope-field* Attributes

Attribute	Description
element-name	Optional name of document element used for this *scope-field*. *Valid values:* A document element name that exists within the document processing pipeline. If no value specified, the element name is the same as the field name. **Note**: For the default scope search pipeline the element name should be set to '*data'*, which is the element normally used when submitting content via the File Traverser or Content API. The document processing pipeline stage *Scopifier* will be automatically configured based on this index profile setting. Refer to section *Scopifier(<clustername>* on page 48. It is possible to include an optional xpath based selection criteria in the attribute. In this case you use the advanced format as described below: *<element>[<xpath>]* The resulting scope field will only contain the content of the source document *element* that match the XML sub-tree indicated by the xpath expression in *<xpath>*. *<xpath>* can match several nodes, they will all be inserted as sub-scopes of the scope field (rootscope). **Note**: The square brackets are part of the attribute syntax and not the xpath expression. The *xpath* specification must not contain spaces. *Example*: element-name=”data[//title]” Only the XML that match the xpath expression (//title) will be mapped to the scope field. *Default:* empty (element name equals scope field name)
type	This parameter is included for future use. *Default: type="text"*

Table D-6   *scope-field* Attributes

Attribute	Description
lemmas	Enables lemmatization and synonym queries for this *scope-field*. When this feature is used, each text scope is expanded with morphological variations or spelling variations of the word based on dictionary lookup during document processing. *Valid values: yes, no* *Example: <scope-field name="xml" lemmas="yes"* *Default: lemmas="no"* Note that lemmatization requires that language dependent tokenization is also specified for the *scope-field*. The standard way to do this is to specify *tokenize="auto"* for the *scope-field*.
tokenize	Enables language dependent tokenization (removing/normalizing punctuation, capital letters, word characters) at the *scope-field* specific level. *Valid values:* *delimiters* - basic, western-style tokenization. The tokenization is performed after the document process pipeline, prior to indexing. This tokenization may apply for all pipelines. *auto* - automatic tokenization, handling both western and CJK tokenization. The tokenization is performed in the document processing pipeline, prior to lemmatization. This tokenization may be configured for all standard document processing pipelines except the *Express* pipeline. *Example: <scope-field name="content" tokenize="auto">* *Default: tokenize="delimiters"* An alternative representation is available, defining *tokenize* as a sub-element to the *scope-field* specification. The alternative representation is mainly applicable for CJK (Asian languages) installations. This alternative format is equal to what is used for the *field* element.

Table D-6 *scope-field* Attributes

Attribute	Description
query-tokenize	Specifies how a *scope-field* should be configured for query-side tokenization. This is mainly relevant for Asian (CJK) queries. This is applied on *scope-field* level to queries that have the indicated *scope-field* as scope of query. *Valid values:* *delimiters* - basic, western-style tokenization. *auto* - automatic tokenization, handling both western and CJK tokenization. *Example:* *Default: query-tokenize="delimiters"* You can define a query-tokenize sub-element for additional configuration to specify that token decomposition should not be used. Token decomposition is by default *on* when saying *query-tokenize="delimiters"* for this attribute. This alternative format is equal to what is used for the *composite-field* element. Refer to the *query-tokenize Sub-element* on page 202 for more information.
vectorize	Enables creation of similarity vectors for the indicated *scope-field*. Similarity vectors are used for the Unsupervised Clustering and Find Similar features. *Valid values: yes, no* *Example: <scope-field name="content" vectorize="yes">* *Default: vectorize="no"* An alternative representation is available, defining *vectorize* as a sub-element to the *scope-field* specification. The alternative representation can be used in order to enable non-standard configuration of the vectorization document processing. Note that the similarity vector tags the entire document with the extracted vectors; it does not append to individual scopes.
wildcard	Enable wildcard query support for this *scope* field. This attribute is for future use only, it is always enabled. *Valid values: yes* *Default: wildcard="yes"*

Table D-6 *scope-field* Attributes

Attribute	**Description**
result	Defines how the query result is presented for this *scope-field*. *Valid values:* *no* - Content from this scope field will not appear in the query result. *dynamic* - A dynamic document summary (teaser) is returned for this scope field in the query results. For *result=”dynamic”* you may also apply a fallback field using the *fallback-ref* attribute. *Example:* *Default: result=”no”*
fallback-ref	For fields with *result=dynamic* attribute, you can indicate a string-type non-scope field as the fallback. If it is not possible to create a dynamic document summary (*teaser*), then the content of this field is returned instead. *Valid values:* Field name *Example: teaser* *Default:* empty
datetime-resolution	This specifies whether the *scope-field* of type *datetime* uses a resolution in seconds or minutes within the search index. This impacts the granularity and the earliest date to be represented for *datetime* type scopes. This attribute does not impact the datetime type format used in queries and content. You can still indicate seconds in the query term, but the seconds component will be discarded in case of minutes resolution. *Valid values:* *second* - resolution in seconds since epoch 01/01/1970; earlier dates will be set to this date. *minute* - resolution in minutes since 01/01/1900; the seconds component in *datetime* scopes and query expressions are truncated. *Example: <scope-field name="xml" datetime-resolution="second"/>* *Default: datetime-resolution="minute"*

Table D-6 *scope-field* Attributes

Attribute	Description
max-result-size	Maximum number of kilobytes that are used for results. For a field having *result=dynamic* this size refers to the size of the source data used for the dynamic teaser. *Valid values: Positive integer 0 through 2097151 (0 through 2 GB)* Note that FAST InStream utilizes Zlib compression of large result fields in order to save disk space. This compression is only enabled for fields with *max-result-size* > 64 (KB), e.g. larger than the default value. If you know that most documents have content of size 10-60 KB for a given field, and disk space reduction is desired, then you might consider setting the attribute *max-result-size*="65" for that field, so that result-text compression is enabled on the field. *Default: max-result-size="64" (KB)*

Scope-field Sub-Elements and Attributes

A *scope-field* element may include the following optional sub-elements (in the given order):

- *query-tokenize*
- *tokenize*
- *vectorize*

These sub-elements are described in the following subsections.

query-tokenize Sub-element

This sub-element can be used to specify how a *scope-field* should be configured for query-side tokenization. This is mainly relevant for Asian (CJK) queries. This is applied on *scope-field* level to queries that have the indicated *scope-field* as scope of query.

tokenize Sub-element

This sub-element can be used to configure advanced tokenization parameters for CJK (Asian languages) installations.

vectorize Sub-element

This sub-element can be used within a *scope-field* element to configure similarity vector boost parameters.

Note! Rank profiles may be used in association with a *scope-field*, but require definition of a *composite-field* for this purpose. Refer to section *Relevance Tuning for Scope Fields* on page 81 for more details.

<scope-field> Example

The following example shows a snippet of a *scope-field-list* definition in the index profile:

```
<scope-field-list>
  <scope-field name="xml" result="dynamic">
    <vectorize default="5:5" alternative="{ja,ko,zh,szh,tzh}:5:0" />
    </scope-field>
</scope-field-list>
```

This example defines a *scope-field-list* consisting of one scope-field named *xml*. The *scope-field* is configured for (dynamic) teaser generation, as defined in the *Field Element and Attributes on page 180.* Alternative vectorization for a set of languages is also defined.

Composite-field Element

This section provides detailed descriptions for the individual attributes associated with the *composite-field* element.

Fields may be grouped into *composite-fields*. Grouping is performed for two main reasons:

- A set of fields (such as *header*, *body, title*) may be referenced with one unique name in queries.
- Each of the fields within the *composite-field* may be assigned to context relevance (ranking) weight. For example, a match in the *title* part of the document is more important than the *body.*
- *Composite-field* may be used to facilitate dynamic ranking of queries against scope fields. Refer to *Relevance Tuning for Scope Fields* on page 81 for details.

A *composite-field* definition consists of a number of field references. These references can either be of *field-ref* type or *field-ref-group* type. A *field-ref-group* is simply a set of *field-ref* entries that are addressed as a logical unit.

Note! The maximum number of *field-ref* and *field-ref-group* entries in a *composite-field* is eight.

For example, you could create the *composite-field content*, which references the two individual fields *title* and *description*. Then a query for the term *news* against these two fields can be expressed as *query=content:news*. A *composite-field* may also contain *rank-profile* entries that define how dynamic ranking should be performed.

The *composite-field* element may also contain *query-tokenize*, *field-ref*, *field-ref-group*, and/or *rank-profile* sub-elements.These sub-elements and their attributes are described in *Composite-field Sub-elements and Attributes* on page 202.

Composite-field Attributes

The *composite-field* element consists of attributes listed in the following table.

Table D-7 *composite-field* Attributes

Attribute	Description
name	**Required**. Name of the *composite-field*. This name is used in the same way as *field* for query purposes. *Valid values:* Lowercase alphanumeric characters
type	This parameter is included for future use. *Default: type="text"*
default	If set to *yes*, this is the default search target for queries that do not explicitly specify *field* or *composite-field.* *Valid values: yes, no* *Default: default="no"*
wildcard	Supports wildcard queries for this *composite-field.* *Valid values:* *yes* - equal to *prefix*, included for backwards compatibility *no* - wildcard queries not supported *prefix* - supports *prefix* wildcard search only *(term*)*. This is a simpler algorithm with slightly better performance than *full* *full* - supports full wildcard search on entire word, using '*' and '?' where *'?'* means any one character and '*' means zero or more random characters (for example, **e?onic** should find all occurrences of Veronica) A '*' by itself is not allowed (searching for a single letter) since it will generate a too great number of searches. *Default: wildcard="prefix"*
substring	If specified, the *composite-field* will be configured to support *substring* queries. Refer to *Substring Query Support* on page 67 for configuration information. *Valid values: 0* through *31, 33* through *63* *Default: substring="0"*
rank	If set to yes, dynamic ranking is used for this *composite-field.* Dynamic rank implies that rank components are computed during matching related to the level of match between document and query. This must be set to *yes* when including a *rank-profile* in the *composite-field.* *Valid values: yes, no* *Default: rank="no"*

Table D-7 *composite-field* Attributes

Attribute	Description
lemmas	Enables lemmatization and synonym queries when using this *composite-field*. Lemmatization only applies for the content of the fields that has been defined for lemmatization using the *lemmatize* attribute in the field specification. If you want to use lemmatization on a given set of fields you must enable this feature in the *composite-field*, and the query must be performed against this *composite-field*. However, by having this configurable both on *field* level and *composite-field* level, you can create a *composite-field* that does not enable lemmatization, even if the *composite-field* includes individual fields enabled for lemmatization. *Valid values: yes, no* *Default: lemmas=“no”*
phrases	Enables phrase query support in the index for this *composite-field*. Disabling phrase support reduces the size of the search index on disk. *Valid values: yes, no* *Default: phrases="yes”* Note that several features require phrase support to be enabled. This includes Phrase queries, Automatic phrase detection, Substring search, Boundary-match and Bigram-boost.
positions	Enables proximity support (implicit proximity rank boosting and explicit proximity operators 'NEAR' and 'ONEAR') in the index for this *composite-field*. Disabling proximity support (the positions index) reduces the size of the search index on disk. *Valid values: yes, no* *Default: positions="yes”* Note: This only applies to index-side proximity support. Result proximity may still be used.
query-tokenize	Specifies if a field should be configured for query-side tokenization. This is mainly relevant for Asian (CJK) queries. This is applied on *composite-field* level; applied to queries that have the indicated *composite-field* as scope of query. *Valid values:* *delimiters* - basic, western-style tokenization. *auto* - automatic tokenization, handling both western and CJK tokenization. *Default: query-tokenize="delimiters”* You can define a query-tokenize sub-element for additional configuration to specify that token decomposition should not be used. Token decomposition is by default *on* when saying *query-tokenize="delimiters"* for this attribute.

Table D-7   *composite-field* Attributes

Attribute	Description
query-boundary-match	Enables boundary match queries for this *composite-field*. This means the ability to perform field exact match and match with start/end of field when the scope of a query is a *composite-field,* not an individual field. Note: Boundary match must also be enabled for the individual fields (*boundary-match="yes"* in the field specification) Refer to the *Query Integration Guide* for details on how to use this feature in queries. *Valid values: yes, no* *Default: query-boundary-match="no”*

Composite-field Sub-elements and Attributes

A *composite-field* element may include the following sub-elements (in the given order):

- *query-tokenize* (optional)
- *field-ref* (one or more)
- *field-ref-group* (zero or more)
- *rank-profile* (zero or more)

These sub-elements are described in the following subsections.

query-tokenize Sub-element

You can specify query-side tokenization. This is mainly relevant for Asian (CJK) queries. This is applied on the *composite-field* level and applied to queries that have the indicated *composite-field* as scope of query. This is configured using the *query-tokenize* sub-element in the *composite-field* element as described in this subsection.

Table D-8 *query-tokenize* Sub-element Attributes

Attribute	Description
type	Type of tokenization. *Valid values:* *auto* - automatic tokenization, handling both western and CJK tokenization. This attribute must be set to *auto* in order to use the *decompose* option. *delimiters* - basic, western-style tokenization, no decomposition. *Default: type="delimiters"*
parameter	Use this attribute if optional token decomposition is to be used for Japanese and Korean. *Valid values:* "" - empty, no decomposition *decompose* - decompose Japanese and Korean tokens *Example:* *<composite-field name="content">* *<query-tokenize type="auto" parameter="decompose">* *</composite-field >* *Default: parameter=""*

field-ref Sub-element

A *field-ref* is a reference to a defined field, to be used in other index profile elements. A maximum of eight field references can be included in the *composite-field* element.

Table D-9 *field-ref* Sub-element Attributes

Attribute	Description
name	Name of the field. *Valid values:* Lowercase alphanumeric characters (for example, *body, headings, path, description, domain, keywords, title, anchortext)* *Default:* empty
type	Type of field. Ranking may be improved by distinguishing between *normal* content (part of the visible document) and *external* metadata. Some ranking rules, such as detecting where the first occurrence of a term is within in a field, does not apply to metadata. This attribute only applies within the *composite-field field-ref* definition (not for *field-ref-group* and *rank-profile*). *Valid values: normal, external* *Default: type="normal"*
level	Level specifies the importance of the field for this field reference. Refer to the *<composite-field> Example* on page 210 and *Field Importance Level* on page 66 for details. This attribute only applies within the *composite-field field-ref* definition (not for *field-ref-group* and *rank-profile*). *Valid values:* Positive integer 1 through 6 *Default: level="1"*

field-ref-group Sub-element

Defining an index with a large number of individual fields (200 or more) will have a large impact on search performance. One way to reduce the number of fields is to define a *field-ref-group*.

A *field-ref-group* is an optional group of fields that can be addressed as a single field within a *composite-field*. This feature may be used to link a number of smaller individual text fields into a *composite-field*. The *composite-field* has a limitation of handling a maximum of eight individual fields; the *field-ref-group* provides a way to include additional text fields within one *composite-field* beyond the eight entries limit. This is especially relevant for database content containing a large number of fields per record.

All fields specified *field-ref* are appended in sequence to construct the *field-ref-group*.

Note that it is not possible to apply different dynamic rank rules to individual fields in a *field-ref-group*.

Table D-10 *field-ref-group* Sub-element Attributes

Attribute	Description
name	Name of the *field-ref-group*. *Valid values:* Lowercase alphanumeric characters *Default:* empty
type	Type of *field-ref-group.* For future compatibility. *Default: append*
level	The *drill-down level.* Refer to *Field Importance Level* on page 66 for details. *Valid values:* Positive integer 1 through 6 *Default: “1”*

rank-profile Sub-element

This sub-element can be used to tune relevance within a *composite-field*. Refer to Chapter 3 *Index Profile Features Management*, section *Relevance Tuning Using Rank-Profile* for additional relevance tuning information.

A *composite-field* may include one or more *rank-profile* definitions. A *rank-profile* may be selected per query using the *SORT-BY* query parameter.

A *rank-profile* sub-element must include a *rank-model* attribute as well as the following mandatory sub-elements (in the given order):

- *authority*
- *freshness*
- *proximity*
- *context*

Refer to the *<composite-field> Example* on page 210 and for an example of *rank-profile* format.

Table D-11 *rank-profile* Attributes (*composite-field*)

Attribute	Description
name	**Required**. Name of this *rank-profile* to be used in queries (in the SORT-BY query parameter). *Valid values:* Lowercase alphanumeric characters
rank-model	The rank-model is included as an attribute to the *rank-profile* element, and defines additional rank parameters than the ones available through the authority, freshness, proximity and context sub-elements. *Valid values:* *default*: generic model which will suit most environments *site*: optimized for Site Search applications *news*: optimized for indexing fresh information such as news articles Contact FAST Professional Services if you need to customize your own rank models.

Table D-12 *rank-profile* Sub-element Attributes (*composite-field*)

Sub-element	Attribute	Description
authority	weight	Relative weight of the authority rank component, defined as a percentage value. Refer to *Configuring Authority Rank Boost* on page 209 for more details.
	field-ref	This attribute indicates a field that should be basis for *Partial Authority Boost*. If this attribute is not set, only *Complete Authority Boost* is supported within this *rank-profile*. Refer to *Configuring Authority Rank Boost* on page 209 for more details.
freshness	weight	Relative weight of the freshness rank component, defined as a percentage value. *Valid values: 0 through 100* *Examples: weight=0* indicates no freshness boost; *weight=100* indicates maximum freshness boost. *Default:*
	field-ref	Name of the field that is used as the date/time base. When using one of the standard document processing pipelines, use the value: *field-ref="docdatetime"*

Table D-12 *rank-profile* Sub-element Attributes (*composite-field*)

Sub-element	Attribute	Description
	auto	Determines how the source for the age (date/time) of the document is selected. *Valid values:* *auto=”no”* : The age of the document is based on the *datetime* value of the element indicated by the *field-ref* attribute. If this element is empty, a *default age value* is assigned to the document. *auto=”yes”* : The age of the document is based on the *datetime* value of the element indicated by the *field-ref* attribute. If this element is empty, the following elements will be used as alternative source for the date/time (age) reference (in priority order): *dtgeneric1* : (optional generic date/time element) *dtgeneric2* : (optional generic date/time element) *crawltime* : Date as detected by the Enterprise Crawler (if used as content source for the collection) If none of the indicated elements contain any date/time value, the freshness rank score of these documents is set to a *default age value* that corresponds to a document age between 23 and 45 days. Refer to the *Content Integration Guide* for details on how to manage datetime elements from the content source.
proximity	weight	Relative weight of the proximity rank component, defined as a percentage value. *Valid values: 0 through 100* *Examples: weight=0* indicates no proximity boost; *weight=100* indicates maximum proximity boost. *Default:*
context	weight	Relative weight of the context rank component, defined as a percentage value. The context sub-element should also include a number of *field-weight* and/or *scope-weight* sub-elements defining the relative weight of individual fields.

Table D-12 *rank-profile* Sub-element Attributes (*composite-field*)

Sub-element	Attribute	Description
field-weight	field-ref	Name of field (non-scope field). For each field to weight, ensure that there is a *field-weight* and/or *scope-weight* attribute with the appropriate field reference and weight value. For the *field-weight* sub-element the *field-ref* attribute must refer to one of the *field-ref* sub-elements of the composite field.
	value	Relative weight of match within this scope field compared to the other fields (scope fields or text fields) of the context sub-element. The value is given as a percentage value. The weight percentages may require additional tuning for best results, based on testing of perceived relevance for the queries within your application.
scope-weight	field-ref	Name of scope field. For each field to weight, ensure that there is a *field-weight* and/or *scope-weight* attribute with the appropriate field reference and weight value. For the *scope-weight* sub-element, the *field-ref* attribute must refer to one of the *scope-field* elements defined in the index profile. Note that this is different from the *field-weight* sub-element where scope fields do not require separate *field-ref* sub-elements in the composite field.
	value	Relative weight of match within this scope field compared to the other fields (scope fields or text fields) of the context sub-element. The value is given as a percentage value. The weight percentages may require additional tuning for best results, based on testing of perceived relevance for the queries within your application.

Configuring Authority Rank Boost

Query match in anchor texts of links to this page/document is used to compute an authority rank component. The authority rank is depending on the level of match with anchor texts and the number of matching anchor texts that points to this document/page.

An anchor text is the textual label associated with a HTML hyperlink. In the HTML:

```
<a href=“http://foo.bar.com”>This is an anchor text</a>
```

the anchor text is “This is an anchor text”. Anchor texts may also be associated with image links (<ALT> tags).

Anchor text extraction are supported in the document processing pipeline templates *SiteSearch* and *NewsSearch*.

FAST InStream defines two concepts for anchor text rank boost:

- Complete Authority Boost. There is an exact match between the query and the anchor text. This will give an additional rank boost. Complete Authority Boost on anchor texts are enabled when the <*authority*> sub-element is defined with a boost weight (independent of the <*field-ref*> setting).
- Partial Authority Boost. One or more of the query terms match one or more terms in anchor texts (full match is not required). Set *field-ref=”anchortext”* to enable *Partial Authority Boost* for the anchor texts extracted from web documents

If your application has other sources of authority information (e.g. citation info) that is more applicable than web style anchor text, you can set the field-ref attribute to this element instead. This will enable Partial Authority Boost for this element.

If this attribute is not set, only Complete Authority Boost is supported within this *rank-profile*.

Note! It is not possible to configure the field that is used for Complete Authority Boost. It will always be ‘anchortext’ which is the built-in field carrying the web page anchor text information.

It is also possible to enable static rank boosting based on link cardinality (similar as use on popular web search engines, but this depends on the structure of your site. Within a site the internal link structure may not necessarily be used for this purpose, as linking is used for many other purposes than authority links. Contact FAST Professional Services if you want to utilize this functionality.

Updating and Verifying Composite-field Relevance Tuning

Follow this procedure if you want to update relevance tuning using *field-weight* static boosting):

Note! This procedure should preferably be performed on a test/staging server first.

1 Determine a set of queries where you want improved relevance, and log the result sequence. It is important that the test queries include words that occur only in the *title* field for some documents and only in *body* field for other documents. This setup will allow you to measure whether or not the relevance tuning changes you create will have the expected effect.

2 Update the index profile. Refer to *Updating the Index Profile* on page 82 for details.

3 Make sure that the relevant documents are re-indexed by manually feeding these documents from the FAST InStream administrator interface.

4 Apply the same set of queries and verify that the result rank change is as expected.

5 Re-feed/re-index all the documents in the search engine.

<composite-field> Example

The following example shows a snippet of a *composite-field* definition (including *rank-profile*) in the index profile:

```
<composite-field name="content" rank="yes" default="yes" query-
tokenize="auto">
   <field-ref name="body" level="1"/>
   <field-ref name="headings" level="2"/>
   <field-ref name="path" level="2"/>
   <field-ref name="description" level="2"/>
   <field-ref name="domain" level="3"/>
   <field-ref name="keywords" level="4"/>
   <field-ref name="title" level="5"/>
   <field-ref name="anchortext" type="external" level="6"/>
   <rank-profile name="default" rank-model="default">
      <authority weight="50" field-ref="anchortext" />
      <freshness weight="50" field-ref="docdatetime" auto="yes" />
      <proximity weight="50" />
      <context weight="50">
        <field-weight field-ref="body" value="5" />
        <field-weight field-ref="title" value="60" />
      </context>
   </rank-profile>
</composite-field>
```

This example defines the *field-ref* sub-element in the *content composite-field*.

The *content composite-field* includes support for *rank*, *default*, and *query-tokenize*. Since *rank* is set to *yes*, dynamic ranking is used for this *composite-field*. Note that this must be set to *yes* when including a *rank-profile* in the *composite-field*. The *default* attribute is set to *yes*, making this the default search target for queries that do not explicitly specify *field* or *composite-field.* The *query-tokenize* attribute is set to *auto*, allowing for automatic tokenization.

The *level* attribute in the *field-ref* sub-element determines the *field importance level* for the named field. The *composite-field* includes the *field-ref* fields *body, headings, path, description, domain, keywords, title* and *anchortext*. The *body* field is placed on *level 1, headings, path* and *description* are placed on *level 2, domain* is *level 3, keywords* is *level 4, title* is *level 5 and keyword* is at *level 6.* Initially the Search Engine locates all documents that match any of the fields in the *composite-field*. If the total number of hits exceeds a fixed limit, the Search Engine will continue looking for hits within all fields with *level >1*, looking for matches in *headings, path, description*, *domain, keywords, title,* and *anchortext* only. If the total number of hits still exceeds the limit, consecutive attempts are performed by increasing the minimum value for *level*. After narrowing the scope using *level*, the matching entries are ranked by static/dynamic rank. Refer to the *Field Importance Level* on page 66 for more information.

Result-specification Element

This section provides detailed descriptions for the individual attributes associated with the *result-specification* element.

The index profile *result-specification* block contains indexing and index field related configuration that applies to search result processing.

A *result-specification* element includes the following sub-elements (in the given order):

- *result-filter* (zero or more, currently max 2 elements)
- *categorization* (optional)
- *clustering* (optional)
- *field-collapsing* (optional)
- *bigram-boost* (optional)
- *result-proximity* (optional)
- *numeric-navigator* (zero or more)
- *string-navigator* (zero or more)
- *result-view* (zero or more)

These sub-elements are described in following subsections.

Result-filter Element

A *result-filter* element is a filter applied to the documents before any results are returned for the query. One type of *result-filter* is available, a result-side duplicate removal filter.

Refer to *Dynamic Duplicate Removal* on page 69 for a detailed description of the result-side duplicate removal filter.

The following table lists the *result-filter* attributes.

Table D-13 *result-filter* Attributes

Element	Attribute	Description
result-filter	namer	Name of the filter. *Valid values:* Lowercase alphanumeric characters (for example, *name="duplicateurlremover"*; this filter removes duplicates for results with identical urls)
	type	Type of filter. *Valid values: duplicate* *Default: type="duplicate"*

Field Reference Sub-Element

The *field-ref* sub-element is present in all the *result-specification* elements. The following table lists the *field-ref* attributes.

Table D-14 *field-ref* Attributes (*result-specification*)

Sub-Element	Attribute	Description
field-ref	name	Reference to the field that the filtering will be based on. *Valid values: field-ref* or *field-ref-group* (for example, name="url")
	type	Ignored in the *result-specification*.
	level	Ignored in the *result-specification*.

Categorization Element

Refer to *Categorization* on page 70 for a detailed description of the categorization feature.

The following table lists the *categorization* attributes.

Table D-15 *categorization* Attributes

Element	Attribute	Description
categorization	name	Name of the categorization specification. *Valid values:* Lowercase alphanumeric characters *Example: name="sitecollapse"*
	sort-by	Specifies if and how nodes in the tree should be sorted. *Valid values:* *none*: specifies that the tree nodes should not be explicitly sorted. This does not mean that they are returned in random order, however, since the algorithms that build the tree add nodes in an order determined by the ranks/positions of the documents in the node. *size*: sorts the nodes according to their size (how many documents they cover). *label*: sorts the nodes lexicographically according to their labels/names.
field-ref	name type level	Refer to Table D-14 for descriptions.

Clustering Element

Refer to *Result Clustering* on page 71 for a detailed description of the clustering feature.

The following table lists the *clustering* attributes.

Table D-16   *clustering* Attributes

Element	Attribute	Description
clustering	name	Name of the clustering specification. *Valid values:* Lowercase alphanumeric characters *Example: name="default"*
	algorithm	Algorithm used for Unsupervised Clustering. *Valid values:* *bestfit* (assigns a document to the cluster with the best fit, if any) *quickfit* (assigns a document to the first cluster that has a 'good enough' fit, if any.) *Default: algorithm="bestfit"*
	threshold	Specifies the minimum degree of similarity that two documents must have for them to be considered to be members of the same cluster. *Valid values:* between *0* and *1* *Default: threshold="0.30"*
	depth	Maximum depth of cluster tree. *Example1: cars/make/volvo is of depth 3. Will stop parsing the category field after reaching this depth. The default value 1 implies that recursion is not applied to the clustered results.* *Example2: Assume that we have two categories sport/tennis and sport/tennis/wimbledon. These topics are considered equivalent if depth equals 2.* *Default: depth="1"*
	size	Minimum member count of a cluster node before recursive clustering is applied. Recursive clustering will build up a cluster tree. Using the default *depth="1"*, recursive clustering will not apply. *Default: size="5"*

Table D-16 *clustering* Attributes

Element	Attribute	Description
	labels	Number of atomic labels to use to describe each cluster node. An atomic label is the string found in a cluster member's document vector. *Default: labels="3"*
	join	If specified, atomic labels are analyzed to see if compound labels can be computed. *Valid values: yes, no* *Example:* Assume that two atomic labels are 'world news' and 'news report'. The joining algorithm would join these two bigrams into the trigram 'world news report'. Joining works by considering if the first/last words of two atomic labels overlap. A whole chain of overlapping atomic labels can be joined. Circular chains are broken arbitrarily. Some cases may exist where the joining heuristic can generate nonsensical labels, ('manchester united nations'). In practice, however, such situations are very rare. Joining of atomic labels usually only makes sense if the strings in the document vector are computed using a bigram type vectorizer. This is the default case in FAST InStream. *Default: join="yes"*
	sort-by	Specifies if and how nodes in the tree should be sorted. *Valid values:* *none*: specifies that the tree nodes should not be explicitly sorted. This does not mean that they are returned in random order, however, since the algorithms that build the tree add nodes in an order determined by the ranks/positions of the documents in the node. *size*: sorts the nodes according to their size (how many documents they cover). *label*: sorts the nodes lexicographically according to their labels/names. *Default: sort-by="none"*

Field-Collapsing Element

Refer to *Field-Collapsing* on page 75 for feature information.

The following table lists the *field-collapsing* attributes.

Table D-17 *field-collapsing* Attributes

Element	Attribute	Description
field-collapsing	name	Name of the field-collapsing specification. *Valid values:* Lowercase alphanumeric characters *Example: name="fieldcollapse"*
field-ref	name type level	Refer to Table D-14 for descriptions.

Bigram-Boost Element

The *bigram-boost result-specification* enables the *bigram-boost* proximity ranking feature.

Table D-18 *bigram-boost* Attributes

Attribute	Description
maximum-phrases	Maximum number of phrases to boost from a query. *Example:* To enable *bigram-boost* proximity for queries where only the first 20 phrases occurring in a query are affected (such a high value is relevant for applications that create queries automatically, as it is unlikely that a user specifies more than 20 phrases in a query), *<bigram-boost maximum-phrases="20"/>* *Default: 20*

Result-proximity Element

The following table lists the *result-proximity* attributes.

Table D-19 *result-proximity* Attributes

Element	Attribute	Description
result-proximity	boost	Turns on/off default setting for result-side proximity-boost. If set to *yes*, this indicates that result-side proximity-boost is enabled by default for all queries. If set to *no*, this indicates that *result-proximity-boost* is not enabled by default, but can be enabled on a per query basis. *Valid values: no, yes* *Default: boost="no"*
field-ref	name type level	Refer to Table D-14 for descriptions. Note that the *field* must be included in the result (*result="yes"*).

Navigator (numeric- and string-) Element

The index profile defines a *navigator* element to manage the dynamic drill-down feature. *Navigators* returned for a specific query will be configured to provide navigation and drill-down capabilities based on the full result set. Refer to *Dynamic Drill-Down (Navigators)* on page 73 for feature information.

There are two types of navigators:

- A *numeric-navigator* operates on number-type fields. The *navigator* provides drilling using value-ranges for the actual field. A *numeric-navigator* element includes the following sub-elements (in the given order):
    - *manual-cut* (zero or more)
    - *field-ref* (mandatory)
    - *range-label* (zero or more)
    - *ignore-value* (zero or more)
- A *string-navigator* operates on literal field string values. The *navigator* provides drilling using value-ranges for the actual field. A *string-navigator* element includes the following sub-elements (in the given order):
    - *field-ref* (mandatory)
    - *ignore-value* (zero or more)

It is possible to create more than one navigator per field. These sub-elements are described in the following subsections.

Table D-20 lists the attributes for both *numeric*- and *string*- type *navigator* attributes.

Table D-20 *navigator* Attributes (*numeric*- and *string*-)

Attribute	Description
name	Name of the navigator. *Valid values:* Lowercase alphanumeric characters *Examples: name="sizenavigator", name="contenttypenavigator", name="charset-navigator", name="languagenavigator"*
display	Display text returned with each field result (header text). *Valid values:* Any text string *Examples: display="Document Size", display="MIME Type", display="Character set", display="Language"*

Table D-20   *navigator* Attributes (*numeric-* and *string-*)

Attribute	Description
unit	The unit of a navigator if needed. The *unit* attribute may apply both for numeric-navigators (*second*) and string-navigators (*color*). *Valid values:* Any text string *Example: unit="kB"*
deep	Determines the type of dynamic drill-down algorithm used. *Valid values:* *yes* - this indicates the use of *index-side* (deep*)* navigators. In this case the statistical analysis and creation of dynamic buckets are performed within the core search engine by means of an optimized vector indexing algorithm. This also enables dynamic drill-down on the entire result set in an optimal way. *no* - this indicates using *result-side* navigators. In this case the statistical analysis and creation of dynamic buckets are performed as part of the query result processing. This is equal to the dynamic drill-down mechanism supported in older versions of FAST InStream. This option is mainly included for backwards compatibility, but may be considered in special cases where index disk usage is critical and number of queries/second are limited. *n* - this is equal to *yes,* but where *n* indicates the maximum number of results that are analyzed. This can be used to optimize performance in large indices, so that query overhead is limited for very large result sets (e.g. from very general queries). This implies that the actual bucket sizes will not be accurate for such large result set queries, but may still represent a fairly correct relative distribution of buckets. *Default: deep="yes"*
default-value	Defines a given value (numeric or string) that is used internally to represent an empty value for the field (no value set for a field from content source or processing). This attribute is optional, but is recommended for performance optimization when using *deep* navigators. The attribute has no effect if the *separator* attribute is set for the field. This also means that any document with the indicated default value in this field also will be treated as *no value present* and will not appear within any of the result buckets. *Default:* none
default-ignore-value	Defines a given value (numeric or string) that is used internally to represent all values defined in *ignore-value* elements. This attribute is optional, but is recommended for performance optimization when using *deep* navigators. The attribute has no effect if the *separator* attribute is set for the field. *Default:* none

Table D-21 lists the attributes for *numeric-navigators*.

Table D-21 *numeric-navigator* Attributes

Attribute	Description
algorithm	The algorithm used for discretization. *Valid values:* *equalfrequency* - specifies that the width (value-range) of different buckets may have different widths. The widths are calculated such that approximately the same number of observations fall into each bucket. *equalwidth* - specifies that the width (value range) of each bucket is equal. The width is static and not computed dynamically. This might produce very unevenly populated buckets, or empty buckets. *manual* - specifies that the width (value range) of each bucket is static and supplied by the user. The value ranges are indicated by the sub-element *manual-cut*. *rangedivision* - specifies that the width (value range) of each bucket is considered to be equal.The width is calculated by dividing the range of observed values into a given number of intervals. This might produce very unevenly populated buckets, or empty buckets. *Default: algorithm="equalfrequency"*
resolution	Resolution of the returned labels. *Example:* To align to every 100 boundary, use *resolution=“100”* *Default: resolution="1"*
intervals	Maximum number of buckets generated. *Default: intervals="4"*
divisor	The divisor used to scale down values before display. *Example:* If the actual values are in bytes and the desired unit is kilobytes, then *divisor="1024"*. *Default: divisor="1"*

Table D-22 lists the attributes for *string-navigators*

Table D-22 *string-navigator* Attributes

Attribute	**Description**
sort-by	Sorting algorithm for string navigators. *Valid values:* *frequency* - orders by occurrence within the buckets *name* - orders by label name *number* - treats the strings as numeric and use numeric sorting. This may be useful when you need descrete values, for example when performing numeric sorting of processor speed. *auto* - combination of *frequency* and *number*. Numeric sorting is used if the field content is numbers. Otherwise frequency sorting is used. *Default: sort-by="auto”*
sort-order	Sort direction for string navigators. *Valid values: descending, ascending* *Default: sort-order="descending”*
filter-buckets	Indicates the maximum number of returned buckets. After sorting the histogram's buckets, use this attribute to cut off any trailing buckets. *Example:* If *filter-buckets=“10”*, then only the first10 buckets are returned, according to the indicated sorting *sort-by* algorithm. *Default: filter-buckets=“100”*
filter-frequency	Limits the number of returned buckets. After bucket filtering, use this attribute to remove buckets that have lower frequency counts. Example: *filter-frequency=“n”* indicates that only the buckets with *members >= n* are returned. If both *filter-buckets* and *filter-frequency* are indicated, filtering based on *filter-frequency* is performed first. *Default: filter-frequency=“1” (no filtering)*
cutoff-frequency	If the number of occurrences of a navigator value in a result set for a search partition is less than or equal to the cutoff-frequency value, then the navigator value is not returned for that partition. For any value *N* other than the default, only navigator values that occur more than *N* times in a search partition will be returned. *Default: cutoff-frequency="0”* (cutoff-frequency is disabled)

Table D-22   *string-navigator* Attributes

Attribute	**Description**
cutoff-frequency-exception	If the number of unique navigator values in a result set for a search partition is less than this value, then no frequency cutoff is done, and all navigator values are returned from that search partition. If cutoff-frequency is used, then this attribute can be used to specify a minimum number of unique navigator values that will be returned regardless of number of occurrences. *Default: cutoff-frequency-exception="0"* (cutoff-frequency is done regardless of the number of unique navigator values)
cutoff-max-buckets	This attribute specifies a limit for the number of unique values (buckets) that will be returned for a navigator within a search partition. Within each partition the best values (with highest frequency) are returned. This is the preferred way to enhance search performance when string navigators with large number of buckets are returned, because it enforces an upper limit on the network traffic between the search nodes. *Default: cutoff-maxbuckets="1000"* (the value *"0"* implies no cut-off)

Table D-22 *string-navigator* Attributes

Attribute	Description
anchoring	Matching mode for string navigator modifiers. This defines how a drill-down query relates to the actual content of the referenced field and the completeness criteria for match. If the referenced field is defined as a multi-value string field, the criteria as defined below applies for individual strings within the field. *Valid values*: *none* - the modifiers will not be anchored. This means that the drill-down query will match documents containing the modifier terms, but the matching field/string may contain additional terms before or after the terms. *prefix* - the modifiers will be anchored to the beginning of the field. This means that the matching field/string must start with the modifier terms. *suffix* - the modifiers will be anchored to the end of the field. This means that the matching field/string must end with the modifier terms. *complete* - the modifiers will be anchored to both the beginning and the end of the field. This means a complete match between modifier and field/string of matching document. *auto* - same as *complete* if *boundary-match="yes"* is given for the referenced field, otherwise it is the same as *none*. *Default value: anchoring="auto"* ***Note!*** For the values *prefix, suffix*, and *complete* to work as intended, it is necessary to specify *boundary-match="yes"* in the referenced field.

Table D-23 lists the *navigator* sub-elements.

Table D-23   *navigator* Sub-elements

Element	Description
field-ref	Reference to the source *field* to be used by the navigator. Note that a *string navigator* may also refer to a *numeric field*. In this case the navigator uses the literal string value. Only one source field (*field-ref*) may be specified per navigator element. *Valid values: field-ref*
ignore-value	Configuration elements that specify values that should be ignored for drill-down consideration by the navigator. Using ignore values for *floating point* and *datetime* fields requires that you know the source format of the value string in the document prior to indexing (as represented from content source or after any custom document processing); the string representation of the *ignore-value* is compared, not the actual numeric value. Common values should be added as *ignore-value*s. *Valid values:* for numeric-navigator - numeric value according to the actual representation from the content source. for string-navigators - alphanumeric string *Default: value="0"*
manual-cut	A manual cut value that applies when *algorithm="manual"* for a numeric navigator. Multiple manual-cut elements can be specified, where each element represents one cutoff-value.
range-label	This element is used to format the bucket (also called modifier) labels. Only valid for numeric-navigators. You typically indicate three range-label elements, for *first, last, middle* respectively. Refer to Table D-24 for attribute descriptions.

Table D-24 lists the *range-label* attributes.

Table D-24 *range-label* Attributes

Element	**Description**
format	A formatting text string that defines the text and the formatting of the value range in the labels. *Valid values*: *auto* - Create English-language range labels automatically. The range label definitions are dependant of the actual field type and provides labels that should suit most 'normal' cases. *<text string with formatting codes>* - This enables you to specify how the label string is formatted. The text string should contain one or two ‘C’ style ‘sprintf’ format codes (%g) that is replaced with the actual value. One value is required for *type="first"* and *type="last"*; two values are normally required for *type="middle"*. *Examples for <text strings with formatting codes>*: *"Less than %g sec"* (to be used with *type="first”*) *"%g sec - %g sec"* (to be used with *type=“middle”)* *"More than %g sec"* (to be used with *type=“last”)* ***Note!*** If *range-label* is not specified for the navigator, all labels will be created using the *format="auto"* setting. *Default value: format="auto"*
type	Label type. *Valid values:* *first* is the label for the first bucket (modifier). *last* is the label for the last bucket (modifier) *middle* is the label for any bucket (modifier) in between.
offset	An offset value that is added to the label value. May be used to display ‘less than’ value. *Valid values: integer* *Examples: offset="1"* *Default: offset="0"*

Result-view Element

The default result view contains all fields that have been specified with the *result* attribute not equal to *no*.

Alternative result views can be defined using the *result-view result-specification*. A result view may contain fields and *composite-fields*.

The alternative result view can be specified in queries using the `RESULT_VIEW` query parameter.

The following table lists the *result-view* attributes.

Table D-25 *result-view* Attributes

Element	Attribute	Description
result-view	name	Name of the *result-view*. This is the name that is specified as the *result-view* parameter at query time. *Valid values:* Lowercase alphanumeric characters Reserved names not to be used: *search*, *content*, *servedcontent*, *proximityboost*, *everything*
	description	Optional description of the *result-view.*
field-ref	name type level	Refer to Table D-14 for descriptions.

Appendix E

Dictionary Management Tool Usage

The Dictionary Management (*dictman)* Tool is a command-line based tool that allows you to update, extend, and maintain your dictionaries.

The Dictionary Management Tool can either run interactively or as a batch processor.

For procedures on how to use the Dictionary Management Tool, refer to Chapter 6 *Configuring Linguistic Processing*.

Note! Using the Dictionary Management Tool requires you to login with a username and password. These are the same as with the Business Manager's Control Panel (BMCP). For details on how to set up a username and password, refer to Chapter 9 *Configuring the FAST Business Manager's Control Panel*.

Basic Syntax

```
dictman -s <server address> -u <username> -p <password> [-P port] [-f
filename] [-d] [-e "commands"]
```

If a filename is sent as a parameter, *dictman* will execute the commands in this file and exit. Responses will be printed to `stdout`.

If no filename is given, *dictman* will run in interactive mode. That is, a prompt will come up.

The port is the base port as issued to the installer.

Options

Table E-1 lists the Dictionary Management Tool options:

Table E-1 Dictionary Management Tool Options

Option	Description
`q [--keepopen]`	Leaves *dictman*. Saves, closes and unlocks dictionaries edited by the current user unless the `[--keepopen]` flag is set. Aliases: quit, exit
`??`	Displays the help page.
`dir [directory=/ [type=*]]`	Lists all dictionaries in a given directory. The type can be: ▪ `sd`: synonym dictionaries ▪ `sp`: spell checking dictionaries ▪ `vd`: spell variation dictionary ***Note!*** The Dictionary Management Tool also supports other FAST InStream dictionary types such as for lemmatization and simple spell check. Contact FAST Professional Services if you need to modify these dictionaries.
`create <dictionary> <type> <language>`	Creates a dictionary of type `<type>` in language `<language>`. This option is useful for verification purposes. e.g. during a thesaurus import The type can be: ▪ `sd`: synonym dictionaries ▪ `sp`: spell checking dictionaries ▪ `vd`: spell variation dictionary ***Note!*** The Dictionary Management Tool also supports other FAST InStream dictionary types such as for lemmatization and simple spell check. Contact FAST Professional Services if you need to modify these dictionaries. `language` must be a ISO-639-compliant 2-letter language code.

Table E-1 Dictionary Management Tool Options

Option	Description
`open <dictionary>`	Opens the specified dictionary for querying or editing and locks it if necessary.
`lock <dictionary>`	Locks the specified dictionary. Dictionaries must be locked to prevent concurrent accesses. Only one user can lock a dictionary at a time
`close <dictionary> [--keeplock]`	Saves and closes the specified dictionary after editing and unlocks the dictionary unless the `--keeplock` flag is set.
`unlock <dictionary>`	Unlocks a dictionary and saves and closes it, if it is open
`deletedictionary <dictionary>`	Purges the specified dictionary from the server. ***Note!*** This action cannot be undone! This command is not subject to the abbreviated form using 'with'. The dictionary needs to be locked to be deleted.
`info <dictionary>`	Prints meta data about the specified dictionary, such as a description, or the time the dictionary was compiled last.
`query <dictionary> <prefix> [count=50 [offset=0]]`	Performs a query against the specified dictionary for keys with a given prefix. Returns as many hits as specified by `count`, starting from the offset specified in `offset`.
`lookup <dictionary> <key>`	Looks up the specified key in the specified dictionary.

Table E-1 Dictionary Management Tool Options

Option	Description
`insert <dictionary> <term[=value] [...]>`	Inserts a key-value pair into the specified dictionary. The format of the value must comply to the format required by the specific dictionary type: ▪ variation dictionaries: The value must be a list of strings, separated by a comma and included in square brackets (`[aaa,bbb,ccc]`) ▪ synonym dictionaries: The value must be a list of spelling variants, separated by comma and included in square brackets plus a list of acronyms, separated by comma and included in square brackets, plus pairs of synonyms, separated by a comma and included in square brackets (`[[list,of,spell,variants],[list,of,acronyms],[[synonym,set],[...]]`) ▪ spellcheck dictionaries: The value must be an integer or logarithmic scale weight ***Note!*** If a key already exists, the values will be merged
`remove <dictionary> <term [...]>`	Removes key-value pairs from the dictionary.
`import <dictionary> <text file>`	Imports the contents of the specified text file into the specified dictionary by splitting each line into key and value at a tab character and performs an insert into the dictionary.
`thesimport <dictionary> <thesaurus xml file> [--verify]`	Imports a thesaurus into a synonym dictionary, used for query-time synonym search. The xml file must be in FAST thesaurus format. The dictionary must be a synonym dictionary. Only those terms will be imported whose language match the dictionary's language. If the `verify` flag is set, the XML file is verified against its DTD.
`varimport <dictionary> <variation xml file> [--verify]`	Imports entries for the specified dictionary containing spell variations for expansion at indexing-time. The dictionary must be a variation dictionary. Its language must adhere to the language of the file. If the `verify` flag is set, the XML file is verified against its DTD.

Table E-1 Dictionary Management Tool Options

Option	Description
`setproperty <dictionary> <property> <value>`	Sets a meta data property of the specified dictionary. Possible properties are: ▪ `encoding`: encoding used inside the dictionary. Except for spell checking, which only accepts ISO-8859-1 to -5, most dictionaries will use UTF8. ▪ `description`: allows for annotating a dictionary ▪ `language`: sets the dictionary's language ***Note!*** Other properties may be read-only and can be displayed using the ?? command.
`save <dictionary>`	Saves the specified dictionary, which needs to be open, without closing it.
`download <path | dictionary> <target file name.dar>`	Downloads a dictionary archive (DAR) containing one or more dictionaries. ***Note!*** The `target file name` needs to specify an absolute path name.
`upload <archive file> [path=/]`	Uploads a dictionary archive and extracts the dictionaries to the specified path.
`compile <path>`	Starts a compile job for compiling dictionaries inside a given path. Compiled dictionaries will be put into a working directory on the server and can be downloaded later using the `downloadcompiled` command. ***Note!*** Compilation is only done on closed and unlocked dictionaries and only on those that have changed since the last compilation. The compilation process may take a couple of minutes. To see its progress, use the `jobstatus` command.
`jobstatus <job number>`	Displays the status of a compile job. The returned list shows a hierarchy of the running jobs. `OK` denotes a finished job

Table E-1 Dictionary Management Tool Options

Option	Description
`waitforjob [job number]`	Waits for the job to be completed. If no job number is given, the Dictionary Management Tolls waits for the last job that was started
`downloadcompiled <target file name>`	Downloads a zip file containing the contents of the compile directory containing compiled versions of the dictionaries created using the `compile` command.
`clean`	Clean the working directory, allowing for a clean rebuild.
`with <dictionary>`	Use the specified dictionary for all subsequent commands until the `endwith` command is called.
`endwith`	Ends a block that performs commands on a given dictionary.
`charsets`	Displays possible encodings (charsets). Set the current charset with the `charset` command.
`charset [charset]`	Displays the current charset and sets the charset for the standard input and for the text import.

Appendix F

Rank Tuning Bulk Loader Usage

Basic Syntax

```
.\rtbulktool [options]
```

Options

Table F-1 lists the options available for use with the *Rank Tuning Bulk Loader* program:

Table F-1 rtbulktool Options

Option	Description
`-d`	This makes the loader delete values from tables given by `-t`.
`-b <bulksize>`	This specifies the maximum number of operations in a bulk commit. *Default: 500*
`-f <file>`	This specifies the path and file name of the XML file that is to be used for input and output.
`-h`	This displays the rtbulktool Help page.
`-i`	This makes the loader deploy the boosts or positions. *Default: off*
`-l`	This makes the loader use the FAST InStream 3.x XML format. *Default: off*
`-p <password>`	This provides the user password necessary to do rank tuning.

Table F-1 rtbulktool Options

Option	Description
`-r`	This makes the loader read values from the boost or position table given in `-t` and write them to the file given in `-f`.
`-t <table>`	This specifies the table to be used: 1 = boost, 2 = position, 3 = both *Default: 3*
`-u <user>`	This provides the user name necessary to do rank tuning.
`-w`	This makes the loader read values from the file given in `-f` and write them to the boost or position table given in `-t`.

Index