FAST INSTREAM

version 4.1

PRODUCT OVERVIEW GUIDE

Document Number: INS1002, Document Revision: A, March 23, 2005

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 and explains the basic concepts, features, and modules of FAST InStream. In addition, it contains a glossary of terms that can be used across all documents in the documentation set for FAST InStream.

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

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

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 to understand the data flow and basic working in FAST InStream.

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

The FAST InStream Documentation Set

About this Chapter

This chapter lists the components of the FAST InStream documentation set and explains how and when to use them.

The chapter includes:

- The Standard FAST InStream Guides
- Additional FAST InStream Components and Documentation
- Using the FAST InStream Guides

The Standard FAST InStream Guides

The FAST InStream documentation set consists of both reference and task-oriented documentation. It includes the following guides:

The FAST InStream Product Overview Guide

The *FAST InStream Product Overview Guide* explains the basic concepts of FAST InStream and describes its features and functionalities. It contains a glossary of terms that can be used across all documents in the set. It serves as an introduction to FAST InStream.

It is referred to as the *Product Overview Guide*.

The FAST InStream Installation Guide

The *FAST InStream Installation Guide* describes the procedures you need to perform to install FAST InStream.

It is referred to as the *Installation Guide*.

The FAST InStream Configuration Guide

The *FAST InStream Configuration Guide* describes the basic procedures for creating a collection, configuring document processing, managing index profiles, configuring advanced linguistics, and other configuration information. The appendices in the guide provide reference information about configuration options and logging. In addition, it contains the DTDs used in FAST InStream.

It is referred to as the *Configuration Guide*.

The FAST InStream Operations Guide

The *FAST InStream Operations Guide* describes operational processes such as starting and stopping the system, monitoring processes, and back-up procedures. In addition, it gives you troubleshooting guidelines.

It is referred to as the *Operations Guide*.

The FAST InStream Deployment Guide

The *FAST InStream Deployment Guide* lists deployment considerations such as system dimensioning, fault-tolerance, component optimization, or benchmarking.

It is referred to as the *Deployment Guide*.

The FAST InStream BMCP User Guide

The *FAST InStream BMCP User Guide* describes the Business Manager's Control Panel (BMCP) which provides Business Manager tools for tuning the ranking of documents within a FAST InStream installation, for monitoring the end-user's query behavior, and for managing user accounts to the BMCP.

It is referred to as the *BMCP User Guide*.

The FAST InStream Crawler Guide

The *FAST InStream Crawler Guide* describes the FAST Crawler and how it is configured.

It is referred to as the *Crawler Guide*.

The FAST InStream File Traverser Guide

The *FAST InStream File Traverser Guide* describes the FAST File Traverser and how it is configured.

Note! The FAST InStream documentation set covers both standard and optional FAST InStream features. Optional features are enabled with individual license keys. If you have not purchased these optional features, these will not be enabled in your installation of FAST InStream.

Additional FAST InStream Components and Documentation

FAST also provides the following additional components with separate documentation sets, listed below:

FAST InStream Software Development Kit (SDK)

The SDK contains additional integration tools for query, content and document processing integration, and search front-end development. It includes the following guides:

- The FAST InStream Content Integration Guide

  The *FAST InStream Content Integration Guide* describes the available programming interfaces along with the required steps needed to integrate FAST InStream with your content sources.

  It is referred to as the *Content Integration Guide*.

- The FAST InStream Query Integration Guide

The *FAST InStream Query Integration Guide* describes the available programming interfaces along with the required steps needed to set up a customized search interface towards your FAST InStream implementation.

It is referred to as the *Query Integration Guide*.

- The FAST InStream Query Language and Parameters Guide

  The *FAST InStream Query Language and Parameters Guide* describes the basics of the FAST Query Language and the FAST InStream query parameters.

  It is referred to as the *Query Language and Parameters Guide*.

FAST InStream Content Connectors

FAST provides an extensive set of Content Connectors for integration with databases and other content repositories.

FAST InStream Security Module (FDSSM)

The Security Module provides application level security when integrating with security environments such as Active Directory.

FAST Real-Time Filter

The Real-Time Filter provides real-time filtering of streams of documents against alert queries.

For purchase information on these additional components, contact FAST Support.

Using the FAST InStream Guides

Use the guides of the FAST InStream documentation set in the following sequence:

- Read the *Product Overview Guide* to get acquainted with the basic concepts, terms, and features of FAST InStream.
- Read the *Deployment Guide* to learn about sizing considerations.
- Read the *Installation Guide* to determine your system requirements and install FAST InStream on your specific platform.
- In the *Configuration Guide*, read those chapters you need to perform a certain configuration task. Refer to the appendices for parameter, options and usage information.
- Use the *Content Integration Guide* to learn about how to integrate FAST InStream with your content application using the provided APIs.
- Use the *Query Language and Parameters Guide* to learn about the FAST Query Language and query parameters.
- Use the *Query Integration Guide* to learn about how to create a custom search front-end towards your FAST InStream implementation.
- In the *Operations Guide*, read those chapters you need for specific monitoring and troubleshooting tasks.
- Refer to the *BMCP User Guide*, the *Crawler Guide*, or the *File Traverser Guide* if you need information on either of these specific modules.
- Refer to any additional documentation provided with the optional components listed in section *Additional FAST InStream Components and Documentation* on page 3.

Chapter 2

FAST InStream at a Glance

About this Chapter

This chapter gives you a high-level overview about the FAST InStream product, its system architecture, features, and modules.

This chapter includes:

- Introduction
- Feature Overview
- System Overview
- Module Overview

Introduction

FAST InStream is an integrated software application that provides searching and filtering services. It is a distributed system that enables information retrieval in any type of information, combining real-time searching, advanced linguistics, and a variety of content access options into a modular, scalable product suite.

FAST InStream:

- retrieves or accepts content from web sites, file servers, application specific content systems, and direct import via API
- transforms all content into an internal document representation
- analyzes and processes these documents to allow for enhanced relevancy
- indexes the documents and makes them searchable
- processes search queries against these documents
- applies algorithms or business rule-based ranking to the results
- and presents the results along with navigation options.

The following sections and chapters explain these processes in more detail.

Feature Overview

FAST InStream provides the following features:

Multiple Content Access Support

FAST InStream can extract content from Web servers, file servers, databases, and other specific applications. In addition, FAST InStream allows you to push content from your own content sources through an API.

Support for Multiple Document Formats

FAST InStream accepts a wide range of different document formats, such as Microsoft Office or PDF. For details, refer to Appendix A *Supported Document Formats*.

Content Freshness

FAST InStream is capable of making new content searchable and filterable in near real-time.

Enhanced Relevancy through Advanced Linguistics

FAST InStream can process both submitted content, and queries with respect to advanced linguistics like spell checking, lemmatization, or proper name recognition to enhance the relevancy of the results returned to the user.

Chapter 10 *Advanced Linguistic Processing* gives you a detailed description of the advanced linguistics features provided in FAST InStream.

Advanced Content Categorization and Result Presentation

FAST InStream provides advanced result presentation through features like classification, dynamic drill-down, or advanced sorting.

Chapter 8 *Processing Queries and Results* and Chapter 11 *Categorizing Content and Search Results* give you detailed descriptions of the content categorization and result presentation features.

Enhanced Usability

FAST InStream integrates with the Business Manager's Control Panel. This is a GUI-based tool that allows the Business Manager to monitor your end users' query behavior and to tune the ranking of individual documents accordingly.

Performance

FAST InStream is infinitely linearly scalable in three dimensions: volume of data, number of users, and freshness of data.

For details about scaling, refer to Chapter 6 *Making Documents Searchable* and to the *Deployment Guide*.

Multiple Platform Support

FAST InStream 4.1 runs on a variety of operating systems including the UNIX-based systems of Red Hat Linux, SuSe Linux, Sun Solaris, IBM AIX, HP-UX, and HP-Ithanium in addition to running on Microsoft Windows. For specific version levels on the supported platforms, refer to Chapter 1 *Installation Requirements and Concepts* in the *Installation Guide*.

System Overview

System Architecture

Figure 2-1 gives an architectural overview of FAST InStream:

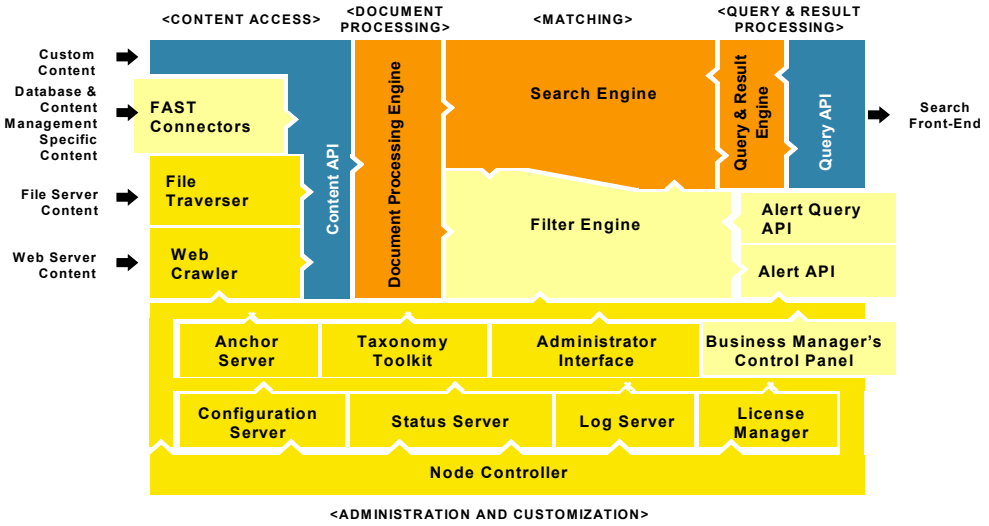

Figure 2-1 : FAST InStream System Architecture

Note! For definitions of terms used in FAST InStream documentation, refer to the *Glossary*.

Data Flow Overview

The data flow through the FAST InStream system consists of the following basic steps:

Submitting Content

Content is submitted using either:

- one of the predefined data source modules that are included with FAST InStream: the FAST Crawler, or the FAST File Traverser
- one of the FAST Connectors, available as separate software packages
- the FAST Content API to push content directly to FAST InStream.

For detailed concepts about submitting content refer to Chapter 4 *Integrating FAST InStream with your Content and Query Infrastructure*.

Analyzing and Processing Documents

Once a content entity has been submitted to the FAST InStream system, it is converted to a document that complies with the FAST InStream internal document format. Each document goes through a set of document processing steps performed by the FAST Document Processing Engine.

The purpose of document processing is to extract additional information, such as the language the content is written in, and to add additional information to the document to improve the search relevancy, such as information about the anchor text a web document might have, or information on which predefined category it belongs to.

For detailed concepts about document processing refer to Chapter 5 *Processing Documents*.

Processed documents are passed on to the FAST Search Engine.

Matching Documents and Search Queries

As new documents arrive at the FAST Search Engine, the Search Engine generates search indices from them.

The end-user or external query application submits search queries through a search front-end or directly to the FAST Query API. The Query API in turn sends the query to the FAST Query and Result Engine, which pre-processes the search queries to improve the relevancy of the results returned upon the query. Examples for such pre-processing are spell-checking or proper name recognition (see Chapter 10 *Advanced Linguistic Processing*).

After having been pre-processed, the search queries are sent to the FAST Search Engine which matches them against its indices and returns a list of resulting documents along with result set navigation options. The FAST Query and Result Engine can then perform post-processing on the result list, such as category result grouping, sorting, or adding navigators for dynamic drill-down.

Finally, the result list is returned through the FAST Query API back to the search front-end or the external query application.

For detailed concepts about matching documents and search queries refer to Chapter 6 *Making Documents Searchable* and Chapter 8 *Processing Queries and Results*.

Managing and Tuning

The FAST Administrator Interface allows you to easily manage and monitor your FAST InStream implementation. It displays status messages from a range of administrative modules such as the FAST Log Server, or the FAST Configuration Server.

For details about system administration, refer to Chapter 13 *Operation and System Administration*.

Furthermore, the Taxonomy Toolkit allows you to manage the categories along which you want the search results to be grouped.

For details about taxonomy management, refer to Chapter 11 *Categorizing Content and Search Results*.

In addition, the Business Manager's Control Panel, which can be purchased separately, allows Business Managers to view the end-user's query behavior and to tune the ranking of individual documents accordingly.

Module Overview

The FAST InStream system consists of different types of modules that can be grouped along with their roles within the data flow in FAST InStream.

Table 2-1 lists and groups these modules.

Table 2-1 FAST InStream Modules

	Module	Description
Data Sources	FAST Crawler	Locates and retrieves files on Web Servers.
	FAST File Traverser	Traverses and retrieves files from directories on file servers.
Document Processing	FAST Document Processing Engine	Performs document processing tasks for format conversion and document relevancy, such as language detection, Asian language tokenization, and lemmatization.
	FAST Taxonomy Toolkit	Enables creation and editing of taxonomy structures and allows you to map documents to categories.
Matching and Result Processing	FAST Search Engine	Performs the indexing and searching tasks within FAST InStream. It indexes new documents coming from the FAST Document Processing Engine, matches them against search queries submitted by the Query and Result Engine, and returns a list of resulting documents and result set navigation options to the Query and Result Engine.
	FAST Query and Result Engine	Processes search queries and search results to enable relevancy-focused searching and result presentation. It provides linguistic query processing features like proper name recognition or spell checking, and result processing features like query highlighting.

Table 2-1 FAST InStream Modules

	Module	Description
APIs	FAST Content API	Allows the standard data source modules of FAST InStream, as well as custom applications, to push content to the FAST Content Distributor. The API is available in Java, C++, and .NET.
	FAST Query API	Allows external search front-end systems to submit their queries and receive result sets in return. The API is available in Java, C++, HTTP GET, and .NET.
Administrative Modules	FAST InStream Administrator Interface	Provides a browser-based graphical user-interface that allows the system administrator to monitor and configure FAST InStream.
	License Manager	Controls all licenses within FAST InStream.
Relevancy Tuning Modules	Rank Tuning Bulk Loader	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.

Table 2-1 FAST InStream Modules

	Module	Description
Optional Modules	FAST Business Manager's Control Panel	Allows system administrators and business users to monitor the end-users' query behavior and to fine-tune the ranking of individual documents based on the monitoring results.
	FAST Connectors	Various FAST Connectors allow you to submit content from databases such as DB2, Oracle, or SQL Server, and other specific applications.
	FAST SDK	Provides additional integration tools for query, content, and document processing integration and search front-end development.
	FAST Real-Time Filter	Provides real-time filtering of streams of documents against alert queries.
	FAST InStream Security Module	Provides application level security when integrating FAST InStream with security environments such as Active Directory

Further Reading

The entire FAST InStream documentation set provides more specific information on the features and modules listed hear. For details on which guide to use for individual subject matters, refer to Chapter 1 *The FAST InStream Documentation Set*.

Chapter 3

Basic Concepts

About this Chapter

This chapter introduces you to the basic concepts of FAST InStream.

It includes:

- Content
- The Content Flow in FAST InStream
- Collections
- Documents and Document Elements
- Index Fields

Content

Data that has not yet been submitted to the FAST InStream system is called *content*. Searchable content entities are called *documents* (see section *Documents and Document Elements* on page 20).

Examples of content are MS Word files, HTML-pages, or database entries.

The Content Flow in FAST InStream

Content submitted to and flowing through the FAST InStream system undergoes different steps of normalization, document processing, and indexing before it is available for searching.

Figure 3-1 gives you an overview of this flow.

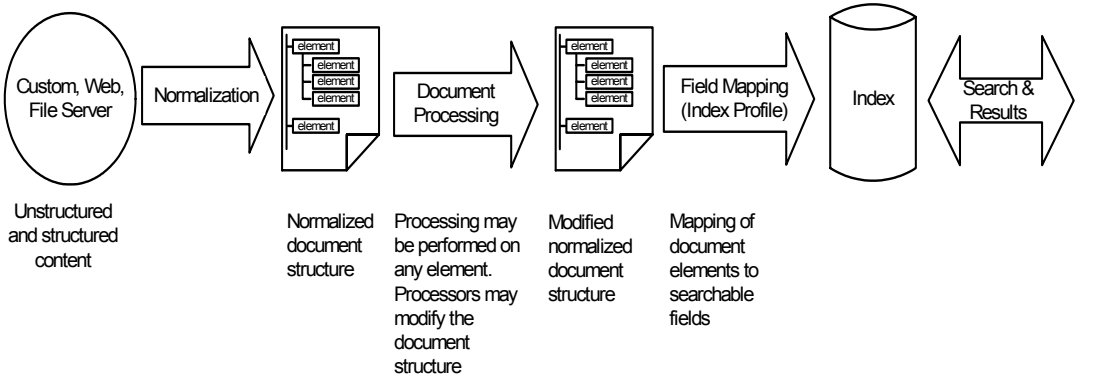

Figure 3-1 The Content Flow within FAST InStream

The following subsections explain the concepts related to the content flow within a FAST InStream installation and describe how content is processed to be searchable.

Collections

Content that is to be retrieved, processed and made searchable, is grouped into so called *collections*.

Collections allow you to treat different groups of content differently, specifying for each collection the way in which its documents are to be processed (see Chapter 5 *Processing Documents*) and indexed (see Chapter 6 *Making Documents Searchable*).

Grouping of content into collections is typically based on criteria like

- origin of the content, such as a product database vs. Web content,
- text document formats, such as Web vs. MS Word files,
- content ownership, such as intranet versus extranet content,
- special processing rules, such as meta data handling,
- authorization level, such as content that is to be made available only for a defined user group.

Grouping content enables end-users or external query applications to narrow down the scope of a search to specific types of documents.

The collection concept does not imply any physical partitioning of the index. FAST InStream may effectively support very large numbers of collections with minor performance impacts.

A collection is set up by defining the content source, for example a set of Web domains, and the document processing rules (see Chapter 5 *Processing Documents*) to be applied. For procedural details on how to do this, see Chapter 1 *Basic Setup*, section *Creating a Basic Web Collection* in the *Configuration Guide*.

Documents and Document Elements

When content is submitted to the FAST InStream system, it is converted into *documents*.

Generally, there is a one-to-one relation between a content entity and a document. What is defined as content entity depends on the way your content is structured.

A document represents the content entity as a set of data elements. These elements contain information extracted from the original content entity, such as the information contained in the *title* or *body* section of an HTML-page.

This document representation is used for the processing performed during document processing (see Chapter 5 *Processing Documents*) prior to indexing.

In addition to the information included in the original document, information improving search and filter relevancy is added to the document as it flows through the system.

Examples of elements are:

- title text
- author
- body text
- an ID that uniquely identifies the document
- the language the document is written in

The conversion preserves the structure of the documents, as well as meta data if embedded in the documents.

By default, text elements are assumed to be encoded according to UTF-8 (Unicode).

The document concept is independent of the type of data being added to the system. For example, if the content source is a database table, each row of information from a table or view may become a document. For both search and filtering, each document is treated as one searchable item and is listed as such in the result list.

Each document has a document identifier that is unique across the entire set of documents handled by the FAST InStream system.

Note that this is not necessarily a URL. It may be a constructed URI representing for example the exact location of a record in a database. There are no restrictions to the format of the URI. However, for crawled content, it makes sense to use the URL of the crawled document. For content pushed to the system from a custom application using the Content API, the client pushing the document into FAST InStream needs to supply the URI. In this

case, the URI may, for example, be the key for storing and loading documents in external storage.

Index Fields

Prior to indexing a document, the FAST Search Engine maps the document's elements to so called *fields*. Fields are defined document elements that are to be searchable.

Defining fields allows the end-user or external query application to specify searches that cover only individual parts of a document such as the *title* or *body* part.

You define fields by creating and specifying an Index Profile (see Chapter 6 *Making Documents Searchable*). FAST InStream supports text, signed and unsigned integer, float, double, and datetime fields. Text fields may contain words or numbers, and queries can be specified for single words, phrases, or a combination of these. Integer, float, and double fields contain numerical values that can be matched against a query by using numerical comparisons such as *less than*, *greater than*, and *equal to*.

Multiple fields may be grouped in so called *composite fields*, allowing a query to be executed on several fields at the same time.

Scope Fields are a special field type that supports dynamic indexing and searching in hierarchical content, such as XML. Refer to Chapter 9 *Scope Search and Dynamic XML Indexing* for details.

For details on the Index Profile structure, refer to the *Configuration Guide*

Further Reading

The following parts of the FAST InStream documentation provide related information:

- Chapter 4 *Integrating FAST InStream with your Content and Query Infrastructure*
- Chapter 5 *Processing Documents*
- Chapter 6 *Making Documents Searchable*
- Chapter 5 *Index Profile Management* and related appendices in the *Configuration Guide*

Chapter 4

Integrating FAST InStream with your Content and Query Infrastructure

About this Chapter

This chapter introduces you to the basics of integrating FAST InStream into your content and query infrastructure.

It includes:

- Integrating FAST InStream on the Content Side
- Integrating FAST InStream on the Query Side

Integrating FAST InStream on the Content Side

Overview

The FAST InStream system accepts content submitted using either one of its standard data source modules or pushed from a customer application through the Content API.
Table 4-1 lists the content access options with relation to the type of content you want to submit:

Table 4-1 Content Access Options

Type of Content to be Submitted	Data Source Module to Use	Type of Module
Content stored on Internet, Intranet or Extranet Web servers	FAST Crawler	standard FAST InStream data source
Content stored on file servers, including XML data exported from databases	FAST File Traverser	standard FAST InStream data source
Other content	Pushing content through the FAST Content API	customer application using the FAST Content API
Content stored in databases, or specific applications	FAST Connectors	optional data source module

The content push approach implies that a custom application or third-party messaging middleware sends data directly to FAST InStream through the Content API.

Crawling

Intranet, Extranet and Internet content from Web servers can easily be submitted using the FAST Crawler. It scans specified web sites by following links for appropriate content and extracts the relevant information.

The FAST Crawler

- allows crawling based on an unlimited number of start URLs.
- scales in a cost-efficient manner with total content size, number of documents, and number of different sites being crawled.
- allows you to specify subcollections within collections with separate request rate and refresh. This enables you to crawl individual subdomains of sites differently.

- enables incremental crawling: The FAST Crawler can be configured to focus on retrieving new content only, or detecting modified or deleted items in previously retrieved content.
- allows you to specify the types of files to be crawled by adding the MIME type through the FAST InStream Administrator Interface, telling the FAST Crawler to recognize and bring back the desired file types only.
- detects whether content on a Web server has been deleted. When a document once detected has not been seen for a given period of time, the FAST Crawler regards it as deleted. This document is then deleted from the collection(s) it belongs to.
- enables specific crawling parameters per collection such as crawling frequency, excluded documents paths and domains.
- retrieves both static and dynamically generated web content.
- allows you to manually activate crawling of specific URIs, sites, or collections.

Internal Process and Data Flow

The FAST Crawler starts crawling from a single start URL, or list of start URLs, and follows every link from this set according to the configuration of the collection it is set to crawl for.

The crawling process consists of two steps – content retrieval and post processing. Figure 4-1 displays this two-step process.

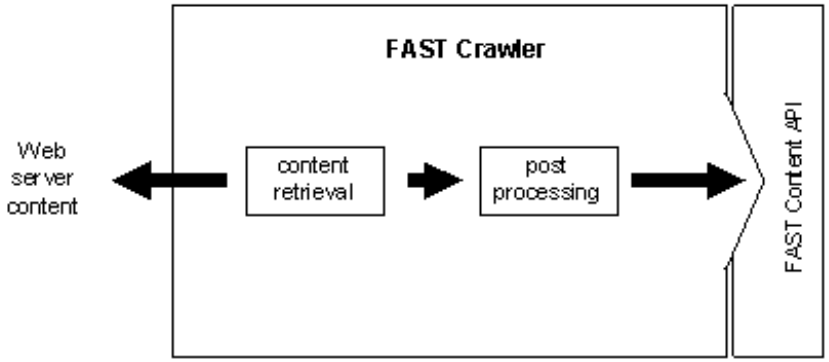

Figure 4-1 FAST Crawler Architecture

During content retrieval, Web content is retrieved. During post processing, the retrieved content is analyzed to determine new or modified content and the parts of the content on the crawled Web server that have been deleted. In addition, during this step, the FAST Crawler detects duplicates within a collection.

Interfacing

The FAST Crawler interfaces directly with the Content API to submit the content.

Note! To retrieve content from locations other than Web servers, you can use either the FAST File Traverser for regular file servers, which is included in your FAST InStream distribution. Or you can use one of the FAST Connectors, allowing you to retrieve content from specific applications like Microsoft Exchange or Documentum (see section *Using a FAST Connector* on page 29). For purchase information, contact FAST Support.

For details on the features of the FAST Crawler and how it is configured, refer to the *Crawler Guide.*

File Traversing

Files on file servers can easily be retrieved using the FAST File Traverser. It scans specified file directories on file servers, retrieves content of various formats, and submits it to a collection in your FAST InStream installation. The File Traverser:

- works on any reachable file server.
- allows you to locate individual types of files by specifying individual file extensions like *html*, *htm*, *pdf*, *doc,* etc.
- sends the located files to the Content API in batches. The size of the batches is configurable by two parameters – total file size and number of files.
- allows you to locate files incrementally by reporting only those files that have changed since the last run (mods_only mode). Typically, file servers contain a lot of static content in the sense that there are many documents that do not change frequently. If the File Traverser is run in mods_only mode, it will not submit content that has already been submitted to the specified collection and that has not changed since then. This saves your FAST InStream installation from processing documents that it has processed before, and helps to increase system performance while ensuring index freshness.
- allows you to determine the files that have been deleted between two runs of the File Traverser (dels_only mode) and to delete them from the collection(s) they belongs to.

- can be run to report without actually performing any operations (report mode). This allows you to verify your File Traverser configuration.
- traverses and submits any XML files, including FastXML.
- can run independently from FAST InStream on a separate node.

Internal Process and Data Flow

The File Traverser is a command line tool. It works on any reachable file server, recursively locating any files associated with the top directory specified in the command line. It processes files that match some specified file extensions like *.html*, *.htm*, *.pdf* or *.doc*. Furthermore, you can configure the File Traverser to map file names to URLs based on a given URI prefix.

Interfacing

The File Traverser interfaces directly with the Content API to submit content.

Monitoring and Logging

The File Traverser logs to the FAST Log Server. You can monitor its log messages in the FAST InStream Administrator Interface.

In addition, the File Traverser logs output to the shell it is started in.

Note! To retrieve content from locations other than file servers, you can use either the FAST Crawler for Web servers, which is included in your FAST InStream distribution. Or you can use one of the FAST Connectors, allowing you to retrieve content from applications like Microsoft Exchange or Documentum (see section *Using a FAST Connector* on page 29). Please contact FAST Support for purchase information.

For details about the File Traverser features, refer to the *File Traverser Guide*.

Optional Data Source Modules

In addition to the FAST Crawler and the FAST File Traverser, the optional FAST Connectors provide support for extracting and submitting content from databases such as DB2 and individual content management systems.

For purchase information contact FAST Support.

Pushing Content to the FAST Content API

Overview

If the content you want to submit is not retrievable from a Web server, a file server, or one of the applications covered by the optional FAST Connectors, you may use the Content API directly to push your content to FAST InStream. The FAST Content API:

- allows submission of content and attached meta data.
- packages the raw data and submits it to the Document Processing Engine.
- allows for passing the content entity as such or passing a URL pointing to the content.
- allows the standard data sources and the custom application to add, remove, and update content within the FAST Search Engines.
- is supplied as a Java J2SE compliant API implementation as well as a C++ implementation.

The FAST Content API allows the standard data sources of FAST InStream as well as custom applications to push content to the FAST Document Processing Engine. This implies improved freshness, as content may be submitted when published, and allows integration with applications not supported by the standard FAST InStream data source modules or one of the FAST Connectors.

You can use the Content API to submit all types of content formats compliant with FAST InStream. For details about content formats accepted by FAST InStream, refer to the section *Allowed Content Formats* on page 28.

When content is pushed to FAST InStream through the Content API, the structure of the retrieved content is preserved and mapped to FastXML. XML content that already is coded according to the FastXML structure is processed and mapped directly into the index. Other XML dialects are converted during document processing (see Chapter 5 *Processing Documents*) using a built-in XSLT transformation stage.

The Content API uses Corba as the underlying transport mechanism between the API client and FAST InStream.

For details about the FAST Content API refer to the *Content Integration Guide*.

Allowed Content Formats

FAST InStream allows you to submit content in different formats:

- in one of the multiple document formats the FAST Document Processing Engine is able to handle. For details, refer to Appendix A *Supported Document Formats*.

- directly in a format complying to the FastXML DTD
- in any XML dialect using an XML transformation stage. The XML transformation is based on standard XSL style sheets.

Using a FAST Connector

In addition to crawling internet sites and traversing file servers, FAST InStream allows you to submit content from other specific applications using the respective FAST Connector. FAST Connectors are optional modules. For purchase details, contact FAST Support.

Integrating FAST InStream on the Query Side

FAST InStream provides the following application programming interfaces (APIs) for creating search interfaces and to integrate FAST InStream on the query side.

- the FAST Query API, available in Java, C++, and HTTP
- the FAST Software Development Kit (FAST SDK), available in Java.

The FAST Query API handles the search query and result traffic between the search frontend and the FAST Query and Result Engine. The Query API:

- takes search queries sent by the end-user and passes them to the Query and Result Engine.
- takes results coming from the Query and Result Engine and distributes them in the desired format for presentation to the end-user or external application.
- is supplied as a Java J2SE compliant API implementation as well as a C++ and a HTTP implementation.

For detailed deployment information, refer to the *Query Integration Guide* and the *Query Language and Parameters Guide*.

The Query API uses Corba as the underlying transport mechanism between the API client and FAST InStream

For details about how to use the APIs, refer to the *Query Integration Guide*.

Further Reading

The following parts of the FAST InStream documentation provide related information:

- the *Content Integration Guide*
- the *Query Integration Guide*
- the *Crawler Guide*
- the *File Traverser Guide*
- the *Query Language and Parameters Guide*

Note! You may also refer to the FAST SDK documentation.

Chapter 5

Processing Documents

About this Chapter

This chapter introduces you to the basic concepts of processing documents.

The chapter includes:

- Overview
- Document Processing Stages and Document Processing Pipelines
- Applying Custom Document Processing
- Entity Extraction

Overview

After content has been retrieved, submitted via the FAST Content API, and converted to documents, these documents are processed within the FAST Document Processing Engine for format conversion and relevancy enhancement.

As explained in section *Documents and Document Elements* on page 20, a document consists of a set of named elements, which contain values such as text strings or integers. Within the Document Processing Engine, these element values are read, analyzed and modified when required. New values can be added to empty elements.

How document processing is performed, is defined per collection.

Document Processing Stages and Document Processing Pipelines

The Document Processing Engine:

- provides linguistic processing of documents through customizable document processing pipelines consisting of multiple document processing stages, instances of document processors.
- allows customers to modify document processing pipelines.
- allows customers to write specific document processors with a minimum of constraints and plug them into arbitrary points in any pipeline, thus extending and customizing the functionality of the existing Document Processing Engine.
- provides support for entity extraction.

Internal Process and Data Flow

The Document Processing Engine consists of multiple document processing pipelines. Any incoming document is sent through a specified document processing pipeline.

Document processing pipelines consist of multiple document processing stages. These document processing stages read element values of the document to be processed, compute analyses on them, and modify or add elements to the document.

A document processing stage performs a particular document processing task and can modify, remove, or add information to a document. It takes one or more document elements (see *Documents and Document Elements* on page 20) as input and outputs new or modified elements that may be processed further on.

With each document processing stage focusing on one particular area of document processing, document processing stages can be reused in a multitude of settings.

When you configure one of the data sources provided with FAST InStream, you specify the collection(s) to which the data source submits documents. Then you assign the collection to a unique document processing pipeline that defines how the collection's documents are processed prior to indexing.

Document processing pipelines are configurable through the FAST InStream Administrator Interface. For details about configuring document processing pipelines, refer to Chapter 3 *Configuring the FAST Document Processing Engine* in the *Configuration Guide*.

You can define new document processing pipelines from the interface, as well as specify the document processing stages to be involved and the sequence of execution within each pipeline.

A typical document processing pipeline for web-retrieved information may consist of the following stages:

- format detection to detect the MIME type of the document and determine if a format conversion is required
- format conversion to convert the document's format from one of a whole range of external formats to the internal FAST InStream document structure
- HTML parsing to extract structure from HTML documents such as title or body
- language and encoding detection to enable language dependent processing and narrowing the scope of a search
- unification of character encoding to UTF-8 Unicode representation
- tokenization
- special tokenization for Asian languages
- extraction of document summary
- lemmatization

Deployment Perspective

The Document Processing Engine also includes a Content Distributor, that is responsible for dispatching incoming documents to the right document processing pipelines by controlling so called *processor servers*. The Content Distributor sends the current document to the processor server along with a pipeline request, and the processor server executes the stages in the requested pipeline on the document.

It is possible to increase the capacity of the Content Distributor by deploying hierarchical content distribution based on top-level and second-level Content Distributors. For details about hierarchical content distribution, refer to the *Configuration Guide*.

Interfacing

The Document Processing Engine interfaces with data sources or the Content API for input and with the Search Engine for output.

Monitoring and Logging

The Document Processing Engine sends its log messages to the Log Server.

The Document Processing Engine can be monitored through the FAST InStream Administrator Interface.

Supported Document Formats

The FAST Document Processing Engine supports a large variety of document formats. For details, refer to Appendix A *Supported Document Formats*.

Applying Custom Document Processing

If you want to apply custom document processing to a set of documents without using and customizing one of the document processors provided with FAST InStream, you can do so by using the *ExternalDataFilterTimeout* document processor as an interface from and to which you can output and input documents.

Also, it is possible to develop custom document processing stages using the FAST SDK. Refer to the FAST SDK documentation for details.

Entity Extraction

Document processing also includes entity extraction.

This feature supports extraction of the following entities:

- geographic names
- personal names
- company names

Entity extraction is per default part of the NewsSearch document processing pipeline, but can be freely used in custom document processing pipelines for example, extracting the above mentioned entities to be used as attributes in dynamic drill down for the analyzed documents.

Entity extraction for other entities are available via two approaches:

- by expanding the framework for the feature above, with other entity lists for custom concepts
- via a regular expression document processor which supports entity extraction based on regular expressions. The default configuration of this document processor supports extraction of E-mail addresses and US locations. Additional regular expressions can be defined for example, extracting product names or customer specific information.

For support on extending the entity extraction feature, contact FAST Professional Services.

Further Reading

The following parts of the FAST InStream documentation provide related information:

- Chapter 11 *Categorizing Content and Search Results*
- Chapter 2 *Configuring the FAST Document Processing Engine* in the *Configuration Guide*
- Chapter 6 *Configuring the Taxonomy Toolkit* in the *Configuration Guide*

Chapter 6

Making Documents Searchable

About this Chapter

This chapter introduces you to the basic concepts of indexing documents to make them searchable.

The chapter includes:

- Indexing Documents: the FAST Search Engine
- Defining how Documents are Searchable: Index Profiles
- Including Meta Data
- Excluding Parts of Documents from Being Indexed
- Executing Search Queries and Returning Results

Indexing Documents: the FAST Search Engine

Overview

The FAST Search Engine receives processed documents from the Document Processing Engine and makes them available for searching. The Search Engine consists of three sub-modules:

- the RTS Indexer: It indexes all documents arriving from the Document Processing Engine and stores the index.
- the RTS Searcher: It runs queries submitted by the end-user or external query application against the index stored by the RTS Indexer.
- the RTS Dispatcher: It merges search results from different RTS Searcher nodes.

On the input side, the Search Engine interacts directly with the document processing pipelines. On the output side, the Search Engine interfaces with the Query and Result Engine.

Both RTS Indexer and RTS Searcher may be instantiated on one or more machines. They may be spread across columns and rows to balance load and network traffic. For details about how to arrange RTS Indexer and RTS Searcher instances refer to the *Deployment Guide*.

The Search Engine provides:

- freshness ranking
- prefix search
- substring search
- wildcard search
- integer search, including range queries and decimal support
- string support
- exact phrase support
- full-string, multilevel sorting
- proximity
- fault-tolerance
- support for indexing and search on a single node or separating indexing and search to their own node. In the latter approach, FAST InStream automatically installs and configures remote search nodes.

Search Engine Clusters

Search Engine instances are grouped into *search engine clusters*. A search engine cluster is a group of Search Engine instances that share the same index configuration, provided by an index profile (see section *Defining how Documents are Searchable: Index Profiles* on page 40).

A search engine cluster has a number of collections - logical groups of content (see Chapter 3 *Basic Concepts*, section *Collections*) - assigned to it. One collection resides inside one search engine cluster, but may be spread across multiple search columns. Since all Search Engine instances in one search engine cluster share the same index profile, all collections assigned to this search engine cluster are indexed in the same way.

There is a one-to-one relationship between an index profile and a search engine cluster: Each search engine cluster in your system needs one index profile. That means, if you want all content fed to your FAST InStream system to be handled according to one index profile, only one search engine cluster is required. If the content fed to your FAST InStream system consists of different types of content, where each content type requires a separate index profile, several search engine clusters are needed.

Each cluster needs its own instance of the FAST Query and Result Engine.

For details on how to deploy the FAST Search Engine refer to the *Deployment Guide*.

Defining how Documents are Searchable: Index Profiles

Overview

In the process of creating the search index, FAST InStream uses a so called *index profile*. An index profile is an XML-based configuration file. It defines the way documents are searchable. It specifies search properties like

- which document elements are to become searchable fields
- which document elements are to become fields that are returned as part of a result
- how to calculate values that are used for sorting and ranking (see Chapter 7 *Concepts of Relevancy*, section *Sorting Results*).

The purpose of an index profile is, to some extent, similar to the process of defining a database.

Each document arriving at the FAST Search Engine is parsed and indexed based on the document's elements. These elements are mapped to the fields given in the index profile. Once the document resides in the index, you can search directly on these fields.

Since documents are grouped in collections, any collection needs to refer to an index profile.

You can set up and use several index profiles to address different types of content, for example Web pages and product database entries, grouped in collections.

FAST InStream contains the following index profile choices upon installation:

- *datasearch-4.0-default.xml*: the default index profile with lemmatization disabled and a default scope field
- *datasearch-4.0-lemmatization.xml*: as the default index profile, but with lemmatization enabled

When you install FAST InStream, you can choose between these types of index profiles or load a custom index profile.

All default index profile files are located in *$FASTSEARCH/index-profiles/* (UNIX) or *%FASTSEARCH%\index-profiles* (Windows) with $FASTSEARCH and %FASTSEARCH% being environment variables set to the directory where FAST InStream is installed.

The Relationship between Document Processing, the Index Profile, and Search Engine Clusters

The index profile concept is closely tied into the concepts of document processing and search engine clusters. Figure 6-1 shows this relationship.

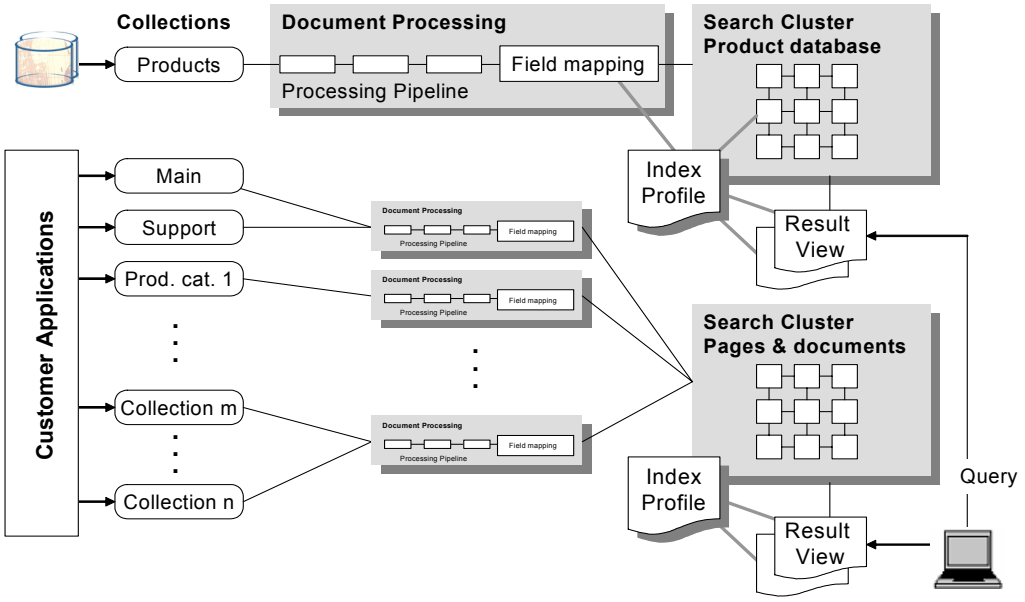

Figure 6-1 The Relationship between Document Processing, Indexing and Search Engine Clusters

During document processing, each document is represented by a set of elements, that can be further processed and later mapped to searchable fields via the index profile. Both elements and fields represent content parts and attributes related to the document, for example, body, title, heading, URI, author, category, etc.

The index profile defines the layout of the searchable index, and specifies how fields are to be treated by query and result processing. Each search engine cluster has an associated index profile.

The Index Profile also includes one or more result views. A result view defines alternative ways for a query front-end to view the index with relation to queries.

The Index Profile Structure

Fields

The basic entity of an index profile is a *field* with its attributes. A field is searchable by default, and is also the basic entity in the result presentation. Typical field attributes are `name`, specifying the field's name, `type`, specifying the type of content the field holds, or `index`, specifying whether the field should be searchable.

Composite Fields

Composite fields allow you to group individual source fields into larger groups of fields by referencing the source fields through field-ref tags or field-ref-group tags. This enables addressing groups of fields together. You can use this feature to apply a common rank score to a group of fields or to make them searchable together.

Table 6-1 describes the available fields defined by the standard Index Profile in FAST InStream. It provides an overview of the default index configuration, indicating the correspondence between the elements of a typical document, the index profile fields that define the parts of the document that are to be searchable, and the fields that are displayed in the result list along with the type of data their values may be in. The differences between the default index profiles listed above is indicated in the table.

Table 6-1 Field definition and mapping in the default Index Profiles

Field or Element	Index	Result	Type	Description and Attributes
anchortext	Y	N	string	Anchor text. Anchor text means navigation text associated with a link to this page/document from other pages. This may be a textual link or alternative text under an image link. This element is updated by the web/document pipelines and should not be changed. Used for the 'authority' based dynamic ranking.
body	N	*	string	The body of a document or web page. * Result: The result field is automatically replaced with the dynamic teaser if this feature is activated in the Index Profile. If there is no dynamic teaser, the result falls back to the static teaser ('teaser' field)
charset	Y	Y	string	Character set of the original (external) document. For example, `iso-8859-1`

Table 6-1 Field definition and mapping in the default Index Profiles

Field or Element	Index	Result	Type	Description and Attributes
companies	Y	Y	string	For entity extraction of company names. The NewsSearch pipeline is configured to extract global company names.

content (composite field)	Y	N	string	The 'content' composite field consists of the individual fields:
				`anchortext`
				`body`
				`description`
				`domain`
				`headings`
				`keywords`
				`path`
				`title`

contenttype *	Y	Y	string	MIME-type of the original (external) document. For example, `text/html` * Element name: `mime`
crawltime	Y	Y	datetime	Timestamp from crawler.
description *	Y	N	string	Reserved for future use.
docvector *	N	Y	string	The document similarity vector reference. * This is a built-in field, not configurable in the Index Profile
docdatetime	Y	Y	datetime	Default datetime field to be used for freshness boosting.
domain *	N	Y	string	URL domain, for example, *www.yourdomain.com.* * Element name: `url.domain`
dtgeneric1 dtgeneric2	Y	Y	datetime	General purpose datetime fields
emails	Y	Y	string	For entity extraction of e-mail addresses provided by the NewsSearch pipeline.
format	N	Y	string	Symbolic representation of the document format (for example, 'Internet HTML')
generic1 generic2 generic3	Y	Y	string	General purpose text fields.
generic4	Y	N	string	General purpose text field.
headings	Y	Y	string	This contains the text of the HTML heading elements (`<h1>`, `<h2>`, etc.).

Table 6-1 Field definition and mapping in the default Index Profiles

Field or Element	Index	Result	Type	Description and Attributes
igeneric1 igeneric2	Y	Y	int32	General purpose integer fields.
keywords	Y	N	int32	Keywords from HTML.
language	Y	Y	string *	Language as detected during processing of HTML documents and documents converted using the document converter. * Type: string (ISO-639)
languages	Y	Y	string *	All languages detected during document processing. * Type: string (ISO-639)
locations	Y	Y	string	For entity extraction of location data. The NewsSearch pipeline is configured to extract major geographical entities.
meta.collection *	Y	*	string	The collection the document is member of. * This is a built-in field, not configurable in the Index Profile * Result field name: `collection`
morehits *	N	Y	int32	When you define an Index Profile with the Field Collapse feature enabled, this result field will indicate if more hits exists for the given value of this field. 0: No more hits available 1: More hits available This is typically used to control whether a 'more hits' link shall be presented below a result. * This is a built-in field, not configurable in the Index Profile
path *	Y	N	string	The path component of the URL, that is starting with the first '/' following the domain. * Element name: `url.path`
personnames	Y	Y	string	For entity extraction of company names. The NewsSearch pipeline is configured to extract global person names.
processingtime	Y	Y	datetime	Timestamp from document processing.
size	Y	Y	int32	The size of the original document ('data' element - bytes).
taxonomy	Y	Y	string	Default taxonomy field

Table 6-1 Field definition and mapping in the default Index Profiles

Field or Element	Index	Result	Type	Description and Attributes
teaser	N	Y	string	Static Teaser (document summary). This is the static teaser generated during document processing (query independent). Used as Fallback for 'body' field in case dynamic teaser is not possible to provide.
title	Y	Y	string	Title field available for searching. This attribute is normally retrieved through HTML parsing or from 'properties' field in documents. You can set the title to another available metadata field instead if more appropriate.
tld *	Y	N	string	Top level domain for the document * Element name: `url.tld`
topics	Y	Y	string	For entity extraction of e-mail addresses and US locations provided by the NewsSearch pipeline.
url	N	Y	string	The URL of the original document. Note: The 'url' field is not searchable (use 'urls')
urls	Y	Y	string	A list of all URLs that points to the document (including duplicates and HTML redirects)
xml	Y	*	scope field	Scope field configured for indexing dynamic XML content. Result = dynamic

Features Enabled by the Index Profile

The following features are enabled by being specified in the Index Profile:

- ranking
- sorting
- tokenization
- lemmatization
- teaser generation
- drill down

For procedural details about how to configure an index profile, refer to the *Configuration Guide*.

Including Meta Data

To include meta data about your content in the indexing process, FAST InStream offers you the following alternatives:

- You may push meta data information along with the content using the Content API. The meta data information is treated as any other element of the content entity and transformed into a document element after the content has been submitted. You then need to design the index profile accordingly to catch any document elements you might want to be included in searching or result presentation, regardless of whether they originate from content meta data or not.
- If your content consists of HTML files, the built-in HTML parser extracts all HTML `<meta>` tags as document elements whose names are prefixed with `meta_`. For example, with the HTML fragment `<meta name="DC.Identifier" content="http://www.fastsearch.com">`, the internal document representation of this HTML file includes an attribute called `meta_DC.Identifier`, that has the value `http://www.fastsearch.com`.

> ***Note!*** The FastHTMLParser is available from the **Advanced mode** Document Processing screen of the FAST Administrator Interface.

- For other content that complies to one of the formats the FAST Document Processing Engine is able to handle (see Chapter 5 *Processing Documents*), the Document Processing Engine is able to detect meta data. For content in MS Word format, for example, it extracts meta data using the MS Word Properties field.

Excluding Parts of Documents from Being Indexed

FAST InStream is able to identify custom tags in HTML documents that indicate sections of the document that shall not be indexed. This can be used in order to avoid indexing of auto-generated content that is similar for multiple pages such as common menu structures.

For details on how to configure this refer to the *Configuration Guide*.

Executing Search Queries and Returning Results

The FAST Search Engine receives queries from the FAST Query and Result Engine, which may have pre-processed the query, for instance to perform spell checking.

When a query is received, the FAST Search Engine matches it against the search index to identify a list of documents that match the query. This list of documents is ordered according to parameters supplied with the query. The ordering is either based on sorting or on ranking (see Chapter 7 *Concepts of Relevancy*, section *Sorting Results*). The FAST Search Engine returns a set of fields from the most relevant documents – based on ranking or sorting – from this ordered list to the FAST Query and Result Engine. The number of documents to return is specified as part of the query. Which fields to return for each document is defined in the Index Profile. Finally, the FAST Query and Result Engine may perform post-processing on the result such as query highlighting (see Chapter 8 *Processing Queries and Results*, section *Query Result Highlighting through Teasers*) before it is returned to the end-user.

Refresh Rate

One of the key strengths of FAST InStream is its refresh rate. The refresh rate is based on a so called freshness chain that contains of a number of indices, where incoming documents reside for a certain amount of time before they are moved to the main index.

The Search Engine can be tuned in various ways for freshness by either:

- increasing the number of indices used in the freshness chain. Using more indices in the freshness indexing chain supports higher search load without hurting freshness.
- the period of time a document resides in each index in the freshness chain.

For details about fine-tuning refresh rates, contact FAST Professional Services at FAST Support.

Fault-Tolerance

The Search Engine and its submodules can be instantiated in multiple numbers to allow for fault-tolerance.

Fault-tolerance is specified during installation. For details refer to the *Deployment Guide* and the *Installation Guide*.

Scaling

The Search Engine can be set up to allow for load-balancing based on the following criteria:

- quantity and types of information stored within the system
- input data rate
- network characteristics
- query rates that the system is set up to provide

For details about how to set up FAST InStream to meet the needs of your settings, refer to the *Deployment Guide*.

Query Highlighting in Dynamic Teasers

Query Highlighting in Dynamic Teasers extracts a neighborhood of the document centered on representative occurrences of the query terms. The relevancy is improved by increasing perceived precision.

The document body is stripped for markup during document processing and stored. A maximum of 64 KB is stored. For each document on the result page, this document extract is retrieved and text segments are generated that include the best matches of the query in that document.

The query used for teaser generation is subject to the same query transformations as those applied to the original search query. This means that if the original search query is subject to proper name recognition, also the query used for teaser generation will undergo proper name recognition. Highlighting is done on individual terms of the query. In particular, phrases are broken down into individual terms, but the preference to proximal terms will maintain the phrasing in the generated teaser.

Further Reading

The following parts of the FAST InStream documentation provide related information:

- Chapter 5 *Index Profile Management* and related appendices in the *Configuration Guide*
- the *Installation Guide*

Chapter 7

Concepts of Relevancy

About this Chapter

This chapter introduces you to the basic concepts of relevancy features and tuning.

The chapter includes:

- The Main Components of Search Relevancy
- Ranking Results
- Sorting Results
- Controlling the Ranking and Sorting of Query Results
- Exact Matching
- Duplicate Removal

The Main Components of Search Relevancy

Search relevancy means the Search Engine's ability to provide relevant results to the end-user. FAST InStream supports search relevancy through the following key steps:

- *Data mining*

  The document processing framework provides support for extensive data mining to perform real-time content relevancy refinement. This includes embedded relevancy tools and integration points for 3rd party modules. For details on document processing, refer to Chapter 5 *Processing Documents*.

- *Linguistic Processing*

  Multiple linguistic processing features provide a number of approximate matching techniques to improve query recall. This includes automatic spell check, matching with inflectional variations of terms (lemmatization), thesaurus (synonym) matching and natural language support (anti-phrasing). The advanced linguistics features are described in further detail in Chapter 10 *Advanced Linguistic Processing*.

- *Sorting*

  Sorting results based on individual document elements allows for highly relevant result presentation. For details on sorting refer to section *Sorting Results* on page 62 in this chapter.

- *Rank Value Calculations*

  The calculation of a rank value based on the FAST InStream ranking model provides a multi-faceted measurement of the quality of the match between the query and a candidate result document. This rank value consists of query dependent and query independent parameters as further described in section *Ranking Results* on page 53.

- *Query Context Analysis*

  Query context analysis refers to the ability to present the information from the query results in context of the query. FAST InStream supports dynamic document summaries that display the segments of the matching document that provide the most relevant match with the query. This feature is further described in section *Query Result Highlighting through Teasers* on page 78 in this chapter.

- *Data Driven Navigation*

  Data driven navigation provides drill-down into the query result or related areas. Drill-down queries may be based on document similarities, category, entity, or terminology information extracted from the documents, parametric drill-down into multiple dimensions of the query result (dynamic drill-down) and drill-down into content domains (for example, all documents from a given site). This feature is further

described in Chapter 11 *Categorizing Content and Search Results*, section *Dynamic Drill-Down*.

- Duplicate Removal

The following sections provide more information on sorting, ranking, and duplicate removal.

Ranking Results

The term *ranking* is used when a set of results is ordered based on calculating a ranking value for each of the resulting documents from a set of relevancy parameters. A high ranking value implies that the document with this ranking value has a high relevancy with respect to the search query. Thus, the resulting documents are ordered descendingly according to their ranking value, listing the most relevant documents first.

The FAST InStream Ranking Concept

FAST InStream ranking is based on a multi-faceted measurement of the quality of the match between the query and a candidate result document.

The relevancy of a document with respect to a query is represented by a ranking value.

In the index profile, you specify one or more *rank profile*s. A rank profile specifies the relative weight of each rank component for a given query. This enables individual relevance tuning of different query applications using a FAST InStream installation.

The FAST ranking model is based on the individual tuning of the ranking parameters of freshness, authority, quality, proximity and context.

The following sections describe these parameters.

Freshness

Freshness denotes the age of a document compared to the point in time when the query is issued.

Refer to section *Freshness Rank Boosting* on page 55.

Authority

Authority denotes the importance of a document as determined by the number and kind of links from other documents to the document in question. To determine a document's authority, FAST InStream detects links from other documents and uses the anchor texts associated with these links to compute an authority rank component.

Refer to section *Link Cardinality and Anchor Text Analysis* on page 56.

Quality

Quality denotes the assigned importance of a document. Since quality metrics are assigned to individual documents or groups of documents directly, the quality of a document is query independent.

FAST InStream provides a set of business manager tools that allow you to assign quality metrics to individual documents or groups of documents.

Assigning quality metrics is further described in section *Relevancy Modifications Based on Business Rules* on page 56.

It is also possible to apply quality metrics through metadata when submitting content via the Content API. This is further described in the *Content Integration Guide*.

The recommended value range for query independent rank values is between 0 and 1000. Values higher than 1000 will typically override dynamic rank completely.

Proximity and Context

Proximity and context measurements determine how well the content of a document matches the query. This is based on the following aspects of a query match:

- the number of query terms matching (for an OR type query) a document within the result set
- query term weighting

  Different relevancy weights may be applied to different terms in a query.
- proximity

  When a query contains multiple terms that are not detected as known phrases, the ranking process takes the relative position of the terms and determines the most relevant results based on the proximity the matching terms in the document have to each other.

  Proximity denotes the distance between, and location of, query terms in the documents. Refer to section *Proximity Ranking and Matching* on page 59 for further details.

- Frequency of query terms occurring in a matching document, compared to the global frequency of the terms in the index.

  More occurrences in the matching document imply a higher ranking value. However, if the term has high frequency over the total index, this will reduce the ranking value.

- Composite Field Relevance Tuning.

  Different document fields, for example title, body, description, price, or type, may be assigned different relevance weight. This allows you to specify for example that a match in the title field of a document contribute more to the document's ranking value than a match in the body field of a document. Refer to section *Controlling the Ranking and Sorting of Query Results* on page 64 for further details.

The *proximity* and *context* parameters of the rank profile control these statistics metric, except for the query term weighting which is selected at query time.

Note! Proximity and context metrics do not apply to numeric queries (integer, floating point or datetime fields).

Freshness Rank Boosting

The freshness rank boosting feature controls to what extent relative age of the documents impacts the rank (relevance score). If enabled, newer documents will appear higher up in the result set.

The date of a document is set when processing a document. The date source may be the content source itself (for example, an application submitting documents via the API), date information from file servers or web servers, or the time of processing within FAST InStream.

The Crawler, File Traverser and Content Connectors will automatically set the time stamp for the document when submitting to FAST InStream. The *Content Integration Guide* describes how you can apply a custom time/date source for documents submitted via the API.

When performing a query, the document date/time value 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 freshness boost feature is controlled using the Rank Profile feature of the Index Profile and by query parameters. There are two ways of controlling this feature on a per query basis:

- Select rank profile for the query by a query parameter. Multiple rank profiles (defined in the Index Profile) may have different weight on the freshness boost within the total rank.
- Select the time base for calculating the freshness boost. The freshness boost is calculated based on the relative age of each document compared to the given time base. Default time base is the current time when performing the query.

Link Cardinality and Anchor Text Analysis

The hyperlink structure of the World Wide Web provides valuable information about the importance of a web page. A web page to which a high number of other web pages refer to, is assumed to be more important than a web page to which only few or no other web pages refer. In particular, links from so called good pages - pages that are referenced by many other good pages - indicate that the linked page is important.

In addition, anchor texts of referring links may provide valuable meta-information about the referenced page.

The Link Cardinality and Anchor Text Import feature allows you to take advantage of these two types of information. It allows you to import both link cardinality and anchor text information for a specified document so that you can

- modify the relevancy metrics for a document based on the number and type of links pointing to or from it (link cardinality) and
- augment a document with data from anchor texts for links pointing to this document.

Link cardinality and anchor text analysis are automatically enabled for all collections that use either the SiteSearch- or the NewsSearch document processing pipeline. These pipelines contain document processing stages that automatically append information about the a document's link cardinality and anchor text to the document's internal representation.

Relevancy Modifications Based on Business Rules

FAST InStream allows you to impact or override the automatic ranking of documents based on business related rules, such as to direct the end-users to business-generating pages. There are three tools available for this purpose:

- Business Manager's Control Panel (BMCP)
- Rank Tuning Bulk Loader

- Rank Modifications Using the Taxonomy Toolkit

Boosting Mechanisms

Absolute Query Boosting

Suppose you want a document to be consistently displayed at a given position in the result set, for example at position one, when a user searches with a specific query. Then you can specify a document-query-combination and assign a fixed absolute ranking position that the specified document is to get within the result list, whenever a user is searching with the specified query.

Absolute Query Boosting also allows you to exclude individual documents from being displayed at all when a user searches with a specific query.

Relative Query Boosting

Suppose you want to ensure a particular document is always displayed among the first 20 documents in the result list, provided a user searches with a specific query. For all other queries, the ranking position of the document shall not be impacted by any boost.

Thus, you specify a document-query-combination and assign an amount of ranking points with which the document's overall ranking value is to be increased whenever a user is searching with the specified query.

Relative Document Boosting

Suppose you want to ensure a particular document is always displayed within the first 20 documents in the result list, no matter which query a user has submitted. At the same time you do not want to assign a fixed result list position to the document.

For this purpose it is possible to specify that the overall ranking value of the particular document to be ranked higher, be increased with a certain amount of ranking points.

Tools to Modify Rank for Individual Documents

These tools enables you to perform *Absolute Query Boost*, *Relative Query Boost* or *Relative Document Boost* for given documents in the FAST InStream index. An example could be a product database where it may be desired to boost products with highest profit margins, boost products related to campaigns, etc.

The following tools exists for this purpose:

Business Manager's Control Panel (BMCP)

This is an optional, GUI based tool which enables rank tuning on document level. The boost value may also be negative, in order to avoid pages to appear on the top of a result list.

Refer to the *BMCP User Guide* for details.

Rank Tuning Bulk Loader

This is a standard FAST InStream tool that enables you to perform the same rank tuning as the BMCP, using an XML file as input. The XML file contains a specification of the rank modifications to be performed.

This approach is preferred if you have the ability to extract the rank boost information from other data or other applications.

Refer to Chapter 9 *Rank Tuning Bulk Loader* in the *Configuration Guide* for details on how to use this feature.

Rank Modifications Using the Taxonomy Toolkit

The Taxonomy Toolkit enables manipulation of relevance score (rank) related to categorization. This enables soft selection of specific category content.

The category rank boost feature enables rank boost of all documents within a category against all or specified queries. Typical use will be to boost for example, a medical category to queries that is specific to this area of medicine.

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.

It is possible to perform *Absolute Query Boost*, *Relative Query Boost* or *Relative Document Boost* for all documents within a category.

Refer to Chapter 6 *Configuring the Taxonomy Toolkit* in the *Configuration Guide* for details on how to use this feature.

Proximity Ranking and Matching

Overview

The term *proximity* denotes the degree to which a query and a document match, based on the distance between the query terms within a document. The calculated proximity value of a query contributes to the overall ranking value of a document within a result set. In general, a document in which the query terms are located close to each other is expected to be more relevant to the query than a document in which the query terms are located far from each other. A high proximity value boosts the overall ranking value of the respective document.

Proximity has only effect on queries with multiple words, and is likely to have higher impact on the result set the more terms the query includes. Like the overall ranking value, proximity does not change the total number of resulting documents for a query, but will improve the ranking order of the result presentation for searches that return multiple results.

FAST InStream supports proximity as a selection and ranking criteria in two main ways:

- Explicit Proximity
- Implicit Proximity

Explicit Proximity

Explicit proximity denotes the fact that you can restrict a query by combining query terms with special proximity operators.

These operators are:

- NEAR/n:

  This operator returns documents that contain the two terms combined by the NEAR-operator with no more than n words separating them. Each term may be a single word or a phrase (enclosed in double quotes). The order of the query terms does not matter for the matching, only the distance.

  The syntax of the operator is [query term 1] NEAR/n [query term 2], with /n being an optional parameter specifying the distance between the two terms in question. If /n is omitted, the default term distance value is 4.

- ONEAR/n:

The ONEAR operator provides ordered near-functionality. This means that the query terms combined by the ONEAR operator must have the same order in the matching document section as in the query. This query expression returns documents that contain the first and second term with no more than n words separating them. Furthermore, the first term must appear before the second term in the matching section of the document.

The syntax of the operator is [query term 1] ONEAR/n [query term 2], with /n being an optional parameter specifying the distance between the two terms in question. If /n is omitted, the default term distance value is 4.

Adding explicit proximity constraints to a query will improve the precision of the result by eliminating irrelevant results from the result set.

For details about applying explicit proximity, refer to the *Query Integration Guide*.

Implicit Proximity

Overview

Implicit proximity denotes the fact that documents get a higher rank value the closer the query terms they contain are to each other.

Implicit proximity will not change the total result set, but will improve the ranking precision of the result set, as documents that contain the query terms close to each other are ranked higher than documents that contain the query terms less close to each other. As such, implicit proximity provides much of the effect of explicit proximity constraints. But in contrast to explicit proximity, that determines which documents match the query, implicit proximity does not change the total number of hits for a query, but impacts the relevance sorting of the results.

Implicit proximity is only one part of the entire ranking of a matching document. Matching text segments in a document are assessed along the following criteria (in decreasing order of significance):

1 Completeness: The higher the number of query terms present in the same element of a matching document, the higher the document's ranking value gets. In addition, important query terms, that means, words that are not stop words, add a higher boost to the ranking value of the document than stop words.

2 Distance: Query terms occurring very near to each other add more to a document's rank value than query terms that are less near to each other.

3 Position: The earlier a query term occurs in a document, the higher the document's rank value gets.

A built-in feature provides a rank boost if the query terms occur close to each other within the 255 first words of a field or composite field. This feature is applied on index level, that is, based on indexed proximity information. This feature is not configurable.

In FAST InStream, implicit proximity boosting features are per default applied during indexing by the FAST Search Engine. They are therefore also referred to as index-side proximity boosting features, as opposed to the legacy feature called result-side proximity boosting (see *Result-side proximity boosting* below). Proximity boosting features are applied to both AND, OR, NEAR, and ONEAR query expressions. This means for example that a query expression a AND b will give higher relevancy to a document where a appears closer to b than in other documents.

Note! Proximity boosting is not applied to query expressions of the format a ANY b.

Where Implicit Proximity is Configured

Implicit proximity is controlled via the rank profile in the Index Profile. For details, refer to Chapter 2 *Index Profile Features Management*, section *Relevance Tuning Using Rank-Profile* in the *Configuration Guide*.

Legacy Proximity Boosting Features

For backwards compatibility, FAST InStream also supports two proximity emulating features that you might use instead of the standard index-side proximity support:

- Result-side proximity boosting

  Result-side proximity boosting implies that proximity rank boosting and explicit proximity are applied during result generation to a limited part of the result set. Result-side proximity is controlled via the *result-proximity* element in the Index Profile.
- Bigram proximity boosting implies that query terms that occur as bigrams, that means next to each other, will add to the relevancy of that document. Bigram proximity boosting is controlled via the *bigram-boost* element of the Index Profile.

Limitations

- Proximity boosting on lemmatized forms is not supported, except for scope fields. Only an exact match of the query term will lead to a boost.

- Proximity boosting is not performed for substring queries. Only exact hits of the query term will lead to a boost.

Sorting Results

Overview

The term *sorting* denotes the ability to order search results according to a value in one or more index fields. Sorting search results depends only on the fields in a document. The position or frequency which the query term or terms may have in the matching documents do not influence sorting.

Which field to use for sorting is specified as part of the query. For details on how to enable this, refer to the *Query Integration Guide*.

Supported Data Types

FAST InStream supports sorting along the following data types:

- text
- numeric
- datetime

Types of Sorting

Full Text Sorting

FAST InStream supports sorting on full text. That means that you may sort on a configurable number of characters, without any limitations on the text string.

Full text sorting includes national text sorting rules.

Ascending and Descending Sorting

FAST InStream allows you to sort results in ascending or descending order.

Multi-Level Sorting

Overview

Sorting may be defined for either single or multiple fields. Specifying multiple fields allows for multi-level sorting. This way, you may for example sort a result set by product name, then by price, and then by date.

Multi-level sorting enables database-type sorting schemes with a list of fields to be used for sorting.

Combining Ranking and Sorting

Multi-level sorting allows you to combine ranking and pure sorting, as the rank field may be one of the sort levels. Multi-level sorting is supported for any field that has been defined for sorting in the index profile. Ascending and descending sort order is available for all fields as part of multi-level sorting. The sort order is specified at query time.

Field Collapsing

A feature related to sorting is field collapsing. Field collapsing allows for folding of results with identical value for a given result field.

You may for example want to collapse all results with the same product name or code in the result set. The result will be re-sorted in a way that the top ranked (based on the normal static/dynamic rank) results for each product name is presented before the other ones.

Field Collapsing is configured within the result specification area of the Index Profile.

Controlling the Ranking and Sorting of Query Results

There are three ways to control ranking and sorting:

- You may specify multiple rank profiles in the index profile.
- You may specify sorting attributes for individual fields in the index profile. This will define which sorting attributes that are available.
- The result sorting can be controlled on a per query basis. By default the result is sorted based on the default rank profile. Query parameters enable you to specify an alternative rank profile for the query, or a set of fields that the result set is to be sorted by.

For details on specifying rank profiles and sorting attributes in the index profile, refer to the *Configuration Guide*.

How to use the result sorting query parameters is described in further details in the *Query Integration Guide*.

Exact Matching

Overview

FAST InStream provides support for exact, boundary-sensitive matching. With exact matching, a query just leads to results if the query string matches a content string exactly, that means including its boundaries.

Use case examples may be a product name field where the full name of one product is a substring of another product name, or a field containing a list of string values, for example, a list of names. In this case it may be desired to be able to match the exact content of each string, and to avoid query match across string boundaries.

Exact matching is enabled per index profile field.

By enabling the exact matching feature for the field, you enable the following:

- Explicit boundary match queries.
- Avoiding phrases to match across string boundaries. For fields that contain multiple strings, this feature will ensure that a string does not match words before/after a boundary indication.

Single and Multi-Value Strings

Exact matching of normal text fields implies that you may require that a query string match the exact content of a field. It is also possible to match a string that is anchored to the start or end of a field.

It is also possible to apply exact matching to text fields containing multiple strings. One document element may contain a set of strings, for example a set of names. This can be represented in two ways from content source and in the document processing:

- As a set of strings (in data type terminology). In this case the mapping to a set of strings for indexing and navigators is performed without a need for a separator.
- As a single text/string element where each string is separated by a configurable separator character.

For details about document elements and multi-value strings, refer to Chapter 3 *The FAST InStream Document Model*, section *Using Multi-Value String Fields* in the *Content Integration Guide*.

Exact Matching and Navigators

Navigators are aware of the exact setting for the field. A drill-down for a navigator on an exact match- enabled field implies that the drill-down query will return only those documents with the exact same content in the field. For other fields the drill-down will find all documents that include the drill-down terms in the field.

For multi-value string fields the same applies for each string within the field.

Exact Matching and Numeric Fields

Boundary match and separators are not applicable for numeric fields. These will always be subject to exact match or value range match.

Applying Boundary Matching

Applying boundary matching requires you to:

- configure the relevant field in the Index Profile

  Refer to the *Configuration Guide* for details on how to configure the Index Profile accordingly.

- enable the use of ^ and $ on the query side to allow for specifying string boundaries

  Refer to Chapter 3 *The Query Language* in the *Query Language and Parameters Guide* for details.

Duplicate Removal

FAST InStream provides two ways of detecting and removing duplicate documents:

- *Crawler Duplicate Removal* - The FAST Crawler is able to detect duplicates within collections. This duplicate removal may be configured to exclude metadata in the HTML document.
- *Dynamic (Result-side) Duplicate Removal* - A result-side duplicate removal feature may be used to detect and remove duplicates across collections, and also enable a more flexible definition of perceived duplicates.

A related feature is Field Collapsing, which does not remove duplicates, but re-ranks documents based on similar value for a given field. Refer to section *Controlling the Ranking and Sorting of Query Results* on page 64 for details.

Crawler Duplicate Removal

The FAST Crawler is able to detect duplicates within collections. It provides two modes for duplicate detection, controlled by the parameter *Include meta in csum*

If this parameter is enabled, then the FAST Crawler will detect changes in the entire document (content and metadata). If this parameter is disabled, then the FAST Crawler will detect changes in content only. Refer to Table 2-4 *Overall Advanced Collection Specific Options* in the *Configuration Guide*) for details.

Crawler duplicate handling will only apply within collections. This limitation is required, as one cannot know in advance which collections a user will have as the scope of a query.

A set of documents with different URL (true duplicates or redirects) classified as duplicates will be indexed as one document in the index, but the field 'urls' will contain all URLs pointing to this document. You may look at this field in the result to verify if the duplicate detection works as expected.

Note! There may be several reasons why two documents that look identical may not be classified as duplicates. An example is pages with identical visible content, but where selection choices from drop-down menus are different. If you think two pages should have been classified as duplicates, it is recommended that you compare the two HTML files and look for 'invisible' differences.

Dynamic (Result-side) Duplicate Removal

The result-side duplicate removal feature may be used to detect duplicates across collections, and also enable a more flexible definition of perceived duplicates. This feature is called dynamic duplicate removal.

The dynamic duplicate removal feature ensures that, within a result set, duplicate documents are represented only by one single document. This document is the one that has the highest relevancy ranking within the set of duplicate documents.

Dynamic duplicate removal is enabled in the index profile. For details, refer to the *Configuration Guide*. The result set to a query will then only list those documents that have different values in this field.

A field to which you typically would apply dynamic duplicate removal is the field of a document containing its URI. When this is specified in the index profile, only documents with different URIs will be included in the result set.

In general, basic duplicate removal based on URI is performed prior to indexing by the data sources. However, in certain cases the same document, that means one specific URI, may appear in different collections within your FAST InStream installation. As queries may be applied to selected collections only, it is not possible to detect or remove such duplicates prior to indexing. In such cases, dynamic duplicate removal allows you to filter out duplicates from the current result set.

The duplicate removal supports a maximum of two slots. A slot indicates an individual duplicate removal criteria. If two slots are defined, at least one of the slots must be identical in order to define two documents as duplicates.

Each slot can consist of more than one field, i.e slot1 can be the concatenation of f1 and f2, while slot2 can be the concatenation of f3, f4 and f5. In this case all of the indicated fields within one of the slots must be equal for two documents to be treated as duplicates.

The dynamic duplicate removal may be resource-consuming if the fields that are basis for the duplicate removal are large (for example, the body field). In this case it is possible to optimize result-side duplicate removal by creating a check-sum field over a set of other text fields during document processing. Refer to the description of the 'Checksummer' document processor in Chapter 3 *Configuring the FAST Document Processing Engine* in the *Configuration Guide*.

Further Reading

The following parts of the FAST InStream documentation provide related information:

- Chapter 6 *Making Documents Searchable*
- Chapter 3 *Index Profile Management* and related appendices in the *Configuration Guide*

Chapter 8

Processing Queries and Results

About this Chapter

This chapter introduces you to the basic concepts of processing queries and results.

The chapter includes:

- Overview
- Query Concepts
- Query Processing
- Result Processing
- The Default Query Front-End

Overview

The FAST Query and Result Engine provides query and result processing prior to submitting the queries to the Search Engines and presenting the result list on the search interface.

It receives search queries from the Query API, analyses them, and, if required, transforms them. It distributes them to the appropriate Search Engine nodes and creates a feedback about what the query analysis has brought up and what search results it gives. Depending on configuration, this feedback is sent back to the end-user, ignored, or used for automatic query resubmission.

Furthermore, the Query and Result Engine receives search results from the Search Engine nodes, processes them and forwards them to the Query API. For more details about the Query API refer to the *Query Integration Guide.*

Query Concepts

The FAST Query and Result Engine

The Query and Result Engine contains multiple processors that perform specific query and result processing tasks. There are two types of processors:

- Processors that contribute to query processing. They form the *query transformation framework*. Their names have the format `qtf_*`.
- Processors that contribute to result processing. They form the result processing framework. Their names have the format `rpf_*` or `rff_*`.

The Query and Result Engine provides:

- Linguistic query processing such as spell checking, anti-phrasing, or proper name recognition
- Result Clustering
- Dynamic Drill-down
- Find Similar
- Dynamic Duplicate Removal

The Query and Its Components

A query submitted to the FAST InStream system consists of two parameters:

- a natural language *query string parameter* which is subject to linguistic query processing.
- a structured *filtering parameter* which is not modified during query processing

For example, looking for "apple" in English documents would be issued with "apple" in the query string parameter and "language:en" in the filtering parameter, where "en" will not be subject to any linguistic query processing like spell checking.

The search query must comply to the FAST Query Language (see *The FAST Query Language* on page 75).

The filtering parameter format is a subset of the simple query format requiring operators and index specifications. Filter query parameters are only used for selecting results (precision and recall). Terms given with filter query parameters do not contribute to ranking.

Refer to the *Query Integration Guide* for a detailed description of the FAST Query Language.

Query Processing

Overview

When an end-user sends a search query to the FAST InStream system, it is subjected to query processing for relevancy enhancement in the FAST Query and Result Engine, before it is passed to the FAST Search Engine to perform the original or processed query.

Query processing is based on linguistic analyses of the query string. It includes the following linguistics features:

- Proper name and phrase recognition (see Chapter 10 *Advanced Linguistic Processing*, section *Proper Name and Phrase Recognition*)
- Spell checking (see Chapter 10 *Advanced Linguistic Processing*, section *Spell Checking*)
- Anti-Phrasing (see Chapter 10 *Advanced Linguistic Processing*, section *Anti-Phrasing*)

Query Modifications

Query processing may be configured globally and per query. Query modifications may be applied in three ways:

- as an automatic rewrite of the query before execution against the index

  This is most useful for Anti-Phrasing, when common query parts as in "Where do I find information about Japan?" are removed and the query is reduced to the essential query string "information Japan".

- as a suggested rewrite, typically presented as a search tip on the result page.

  This is a more conservative approach avoiding any unexpected query rewrites that the end-user did not intend. It is most useful for proper name recognition, when the query string "World Cup" is detected as a phrase, and a search tip such as 'Did you mean "World Cup"?' is returned. It is also useful for spell checking.

- a combination of the two above: The query first is executed in the original form. In case of no hits, the query is automatically resubmitted using the automatic rewrite option, and the new result is presented to the user.

  This is an approach that is transparent to the end-user. The resubmission parameter is set per query and the result received on the API will also indicate the transformed query.

Query Resubmission

The resubmission parameter is set per query and defines which of these features are to be enabled if the original query returns no hits.

FAST InStream is able to perform a number of automatic or suggested transformations of the user's query, based on advanced linguistics. This includes spell checking, proper name recognition and anti-phrasing.

There are three cases of the query transformation:

- Modify – The query term string is automatically modified using the transformation parameters. The modified query is executed and the result set is returned
- Conditional Modify – The query term string is automatically modified if no hits are returned by the executed original query
- Suggest – The executed query is not transformed, but a suggested transformed query is returned together with the result set, based on the original query. This flexibility allows the application or the user to decide how to modify the query terms entered by the user.

For details, refer to the *Query Integration Guide*.

The FAST Query Language

The FAST Query Language (FQL) is used to express query terms, operators and query modes/options. This is further described in the *Query Language and Query Parameters Guide*.

Note! In addition to the FAST Query Language (FQL) FAST InStream provides two alternative query languages, the *Simple Query Language* and the *Advanced Query Language*. These query languages are included for backwards compatibility and do not support all features provided in FAST InStream.

FAST InStream also provides a Verity Query Language Emulator that enables customers to combine FAST query syntax and Verity query syntax within the same query. This is further described in Chapter 6 *Verity Query Language Emulator* in the *Query Language and Parameters Guide*.

You can use the FAST Query Language to perform exact searches and to narrow the scope of your search to values belonging to a specific FAST InStream field, composite field or scope field.

A query language expression may contain a number of nested sub-expressions of one or more of the following types:

- query term: A query term consists of one or more words, strings or numeric values to search for in a query.
- scope specification: A scope specification limits the possible matching sections of the documents to a specific field, composite field or a scope structure within the field.
- operators: Operators may apply boolean operations (AND, OR, etc.), define certain constraints to the operands (for example, filter()), apply proximity constraints (max word distance between matching terms), apply numeric range operations, or specify data types and attributes to the data (such as linguistics operations).

Result Processing

Overview

After the query sent by the end-user has been processed, it is passed on to the FAST Search Engine, which matches it against the index and returns the list of results to the FAST Query and Result Engine. There, the list of results is processed before being sent to the FAST Query API, which submits the result list along with information on applied or suggested query transformations.

Result processing includes the following features:

- Category result grouping
- Find Similar
- Field-based categorization
- Query highlighting through teasers
- Duplicate removal.

Result Views

A result view includes the information that is returned with each search result. In its simplest form the result view is a short teaser summarizing the content of a document. However, the result view in FAST InStream is completely configurable and may contain a smaller or larger set of fields of the initial document. In certain cases, such as database indexing, it is convenient to provide all the indexed fields of each database record in the result view, so that the customer application may present the data in various ways without the need for retrieving the database record once more.

When defining the index profile for a certain collection, you specify which fields are to be returned as part of result views. This configuration impacts the total size of the index, as this information will reside on disk within the index. Based on that, it is possible to define different result views that can be applied to a query. Which of these specified result views to apply when a result set actually is to be presented, is specified by a query parameter.

The definition and selection of field views impacts the amount of data returned from a query. Therefore, more information in the result view implies more bandwidth used between FAST InStream and the customer application, and will also have some minor performance constraints.

Query Result Highlighting through Teasers

The result view may include a teaser field. Teasers allow you to highlight important parts of a query result. A teaser is a summary field that is generated in order to be used as a general result summary of documents in the result set presentation. Two types of teasers are supported (Table 8-1):

Table 8-1 Teasers

Type of Teaser	Description
static teaser	This is a generated summary field that is convenient to use when presenting results from, for example, web pages or text documents. This teaser is created during document processing, and typically analyzes a HTML document, extracting a few lines of text that reflect the most relevant content of the document.
dynamic teaser	This is a generated summary field that enables presentation of a document extract in context with the search query. The text of the document body is used during result processing in order to retrieve the text segments that include the best matches of the query. In most cases the dynamic teaser provides a more relevant text for the result pages than the static teaser. The relevancy of a text segment is determined by (in decreasing order of significance): **1** phrase matching **2** completeness: The more search terms a text segment contains, the more relevant this text segment is **3** proximity: Text segments that contain query terms that occur near each other are more relevant than others. **4** position: The earlier a text segment containing one or more search terms occurs in the document, the more relevant it is than others.

Which teasers to use in a result view is specified in the result view sections of the index profile for the respective search engine cluster. You can specify one teaser per result view; in addition, you may specify a field to be used as a fallback teaser field in case the generation of the original teaser field fails.

For details about defining teasers in the index profile, refer to the *Configuration Guide*.

The Default Query Front-End

FAST InStream provides a default query front-end, available from the **Search View** tab in the FAST InStream Administrator Interface.

This section gives you an overview of the features available from the query front-end.

For details on the individual features, refer to the *Query Language and Parameters Guide*.

Note! Refer to the FAST InStream Software Development Kit (SDK) for details on how to customize the default query front-end.

The default query front-end consists of three main areas:

- the **Main Search page**, providing simple search features
- the **Advanced Search page**, providing advanced search features
- the **Setup page**, allowing you to set a number of properties for the search page

Features Available from the Main Search Page

The main search page provides the following features:

- a default search box:

  The default search box enables you to insert one or more query terms. Unless you specify a search type in the **Advanced** search settings, these query terms will be linked together by an AND operator. The query will be performed against the default composite field within your index.

- a **Sort By** box:

  The **Sort By** box is based on a configurable set of sorting criteria specified in the Index Profile for fields or rank profiles. It allows you to specify ascending or descending sorting.

- a **Collection** box:

  The **Collection** box allows you to specify whether the query is to be executed on individual or all collections.

- a result list:

  The result list provided after a search shows the query results including a configurable set of document fields and meta fields as defined in the setup page.

It provides the following features:

- support for the *Find Similar* feature for each document as defined in the setup page
- support for *Field Collapse* for each document as defined in the setup page
- a **Navigation** section:

  The **Navigation** section includes an optional taxonomy field as defined in the setup page. This is currently based on the Supervised Clustering feature.
- a **Drill-Down** section:

  The **Drill-Down** section displays all defined navigators.
- a **KeyWords** section:

  The **KeyWords** section displays the top concepts as extracted in the Similarity Vectors in the result set.

Features Available from the Advanced Search Page

The advanced search page provides the following advanced query features in addition to the features provided by the main search page:

- a **Search Type** drop-down list:

  The **Search Type** drop-down list allows you to specify the query language and Boolean operation or search type to be used.

  The default is All Words (boolean AND).

  The following types are supported:

  - **All Words**:

    This is equal to the FAST Query Language (FQL) statement

    ```
    string("<the text added in the search box>", mode="AND")
    ```
  - **Any Words**:

    This is equal to the FAST Query Language (FQL) statement

    ```
    string("<the text added in the search box>", mode="OR")
    ```
  - **Exact Phrase**:

    Equal to the FAST Query Language (FQL) statement

    ```
    string("<the text added in the search box>", mode="PHRASE")
    ```
  - **Structured**:

This type enables you to express any query expression in the FAST Query Language (FQL). If you select this type of search, you will be provided with a multi-line search box which makes it easier to express complex queries.

Refer to the *Query Language and Parameters Guide* for details on the FAST Query Language (FQL) syntax.

- a **Query Language** drop-down list:

  The **Query Language** drop-down list allows you to select any of the natural languages that can be detected by FAST InStream.

  Note! This parameter does not filter the query results by language. It applies the LANGUAGE query parameter that controls language dependant linguistics processing.

- an **Encoding** box:

  The **Encoding** box allows you to specify the language encoding of the submitted query string. This specification controls the query language encoding parameter. Typically, you use this if the query string is not encoded according to UTF-8.

- a **Filter** box:

  The **Filter** box allows you to specify a FILTER query parameter. This parameter is included for backwards compatibility, as the FAST Query Language provides in-line support for filter terms.

- a **Hits** box:

  The **Hits** box enables you to control the number of hits returned per page.

- an **Offset** box:

  The **Offset** box allows you to define the offset to the first result presented

- a **Spellchecking** selection box:

  The **Spellchecking** selection box enables you to switch on or off automatic spell checking as controlled by the SPELL query parameter.

- a **Lemmatization** selection box:

  The **Spellchecking** selection box enables you to switch on or off the Lemmatization feature.

  Note! If lemmatization is not enabled in the Index Profile, applying lemmatization for the query will result in no hits returned.

- an **All Fields** selection box:

The **All Fields** selection box enables you to see the content of all fields returned with the query result. This is useful for testing purposes.

- a **Debug** selection box:

  The **Debug** selection box allows you to switch on or off the output of low-level debug information.

Features Available from the Setup Page

The **Setup** page enables you to set a number of properties for the search page. It includes the following features:

- an **FDS Host** box:

  The FDS Host box indicates the host address of the FAST InStream Configuration Server.

  Note! Do not change the value provided in this box.

- a **Configuration Server Port** box:

  The **Configuration Server Port** box indicates the port number to the Configuration Server.

  Note! Do not change the value provided in this box.

- a **Data Fields** box:

  The **Data Fields** box allows you to specify which of the returned fields to display in the result list. Each field is separated by '|'.

- a **Meta Fields** box:

  The **Meta Fields** box allows you to specify which of the returned meta fields to present to the left of each result below the hit number.

- a **Sort Fields** box:

  The **Sort Fields** box allows you to specify which fields the user may select for result sorting.

- a **Logo** box:

  The **Logo** box allows you to specify the path to the image used as a logo on top of the main search page.

- an **Additional Search Parameters** box:

The **Additional Search Parameters** box allows you to add additional query parameters to the **Advanced** part of the search page. You may use this to customize the query front-end and to test various query parameters. The format is as follows:

name;title[|name;title]*

with

name specifying the name of the query parameter (http query interface format).

title specifying the lead-in text displayed in the query page.

A number of such parameters can be defined, where each name:title pair is separated with a '|'. Each parameter will appear on a separate line in the **Advanced** search page. The associated **Enable** radio button in the advanced search page enables you to set a parameter once and keep the value in subsequent queries. Example:

qtpipeline;Query Pipeline|type;Query type (compatibility)

This enables you to specify an alternative query processing pipeline, and set the 'query type' query parameter. This setting can be used to submit queries using the Advanced/Simple Query Language (for backwards compatibility).

- a **Category Field** box:

  The **Category Field** box allows you to specify the field used for taxonomy information.

- a **Collapsing Field** box:

  The **Collapsing Field** box allows you to specify an integer field used for field collapsing.

- a **Show more hits... string** box:

  The **Show more hits... string** box allows you to specify the actual text string to be displayed to indicate that more hits exist when using field collapsing.

- a **Navigation string display size** box:

  The **Navigation string display size** box allows you to specify the maximum size of each modifier within a *Navigator* (Dynamic Drill-Down) drop-down box. If not set, no limit is imposed. You may use this limit to control the maximum width of the left column in the result presentation.

- a **Show Field Collapsing** check box:

  The **Show Field Collapsing** check box allows you to switch on or off Field Collapsing in the results.

- a **Show Navigation** check box:

  The **Show Navigation** check box allows you to specify whether the box named **Navigation** (containing taxonomy presentation) shall be included.

Note! This is not related to Dynamic Drill Down Navigation.

- a **Show Drill Down** check box:

  The **Show Drill Down** check box allows you to specify whether Dynamic Drill-Down shall be presented in the query results.

- a **Show Keywords** check box:

  The **Show Keywords** check box indicates whether the **KeyWords** box shall be presented in the query results.

Further Reading

The following parts of the FAST InStream documentation provide related information:

- Chapter 10 *Advanced Linguistic Processing*
- Chapter 4 *Configuring Linguistic Processing* in the *Configuration Guide*
- Chapter 5 *Index Profile Management* and related appendices in the *Configuration Guide*
- *Query Integration Guide*
- *Query Language and Parameters Guide*

Chapter 9

Scope Search and Dynamic XML Indexing

About this Chapter

This chapter explains what Scope Search and dynamic XML indexing is and describes its core capabilities.

It includes:

- Overview
- Scope Search Capabilities
- Dynamic XML Indexing

Overview

Scope Search is a feature that enables search in hierarchical content structures without a need to know the schema in advance. Inside the FAST InStream index the hierarchical content is represented as a hierarchy of scopes.

What Scope Search can be Used for

The Scope Search feature may be used for:

- Indexing customer XML content without any knowledge of the DTD/schema. *FAST InStream* includes a dynamic XML pipeline that maps submitted XML to one or more scope fields.
- Indexing a more dynamic field structure using the Scope Search framework. In this case it is possible to change field structure without changing the Index Profile. In this case XML is used as an intermediate format in order to submit structured data to *FAST InStream*.

What Scope Search is Based on

Scope Search in FAST InStream is based on the following features:

- *Scope Indexing* provides a scope-aware indexing of content with hierarchical structure, enabling efficient search in scope structures.
- *Dynamic XML Indexing* provides a mapping from any XML to the internal FAST Scope structure.
- The *FAST Query Language (FQL)* is a new query language introduced in *FAST InStream* which supports scope search queries.

Scopes

Figure 9-1 shows a simple example of a scope field within a searchable document. The indicated scope field in the Index Profile is named *book*, and contains *Authors* elements which in turn contains one or more *Author* elements.

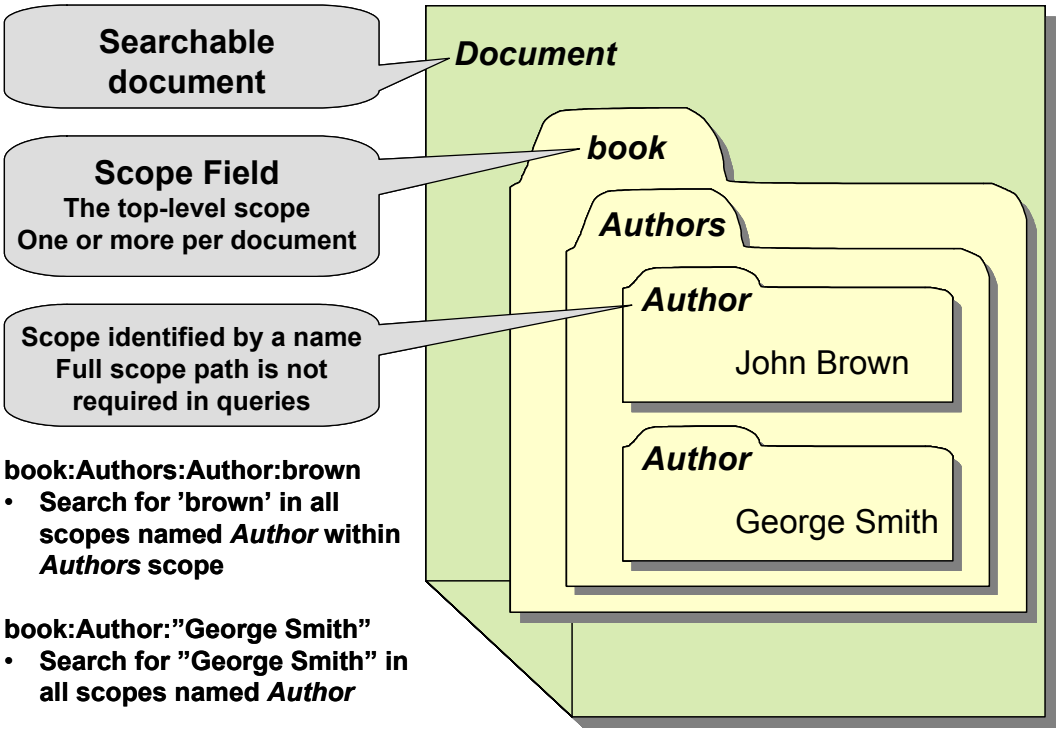

Figure 9-1 Example document scope structure

Scope fields, normal (text/numeric) fields and composite fields may be combined within the same Index Profile.

Scope Search Capabilities

This section describes the core Scope Search capabilities. It includes:

- The Scope Field
- Scope Data Types
- Query Language
- Ranking
- Result Sorting
- Linguistics
- Dynamic Drill-down
- Categorization
- Dynamic Document Summary (Teaser)
- Partial Update

The Scope Field

The FAST Search Engine and its indexer are based on a field structure that defines the schema of the indexed content. The schema is defined using the Index Profile. The Scope Search feature is facilitated by introducing a new field type in the FAST InStream index, named scope-field. Therefore, a scope-enabled index may include the following types of fields:

- *basic fields*. A basic field may be of type string (any textual content), int32 (32 bit signed integer), float, double or datetime (representing a date/time value as a numeric value in the index).
- *composite fields*. A composite field includes a set of basic string fields that can be matched using the built-in dynamic ranking mechanisms in *FAST InStream*.
- *scope field*s. A scope field contains hierarchical scope content. The individual sub-scopes of a scope field may be of any data type supported by *FAST InStream* (string, int32, float, double or datetime). For textual scopes a sub-set of the dynamic ranking mechanisms as provided for composite fields will apply. When defining a scope field, there is no need to define the actual scope structure within the scope field in advance.

> ***Note!*** A scope field can be part of a composite field.

A *FAST InStream* Index Profile may contain a combination of one or more fields, composite fields, and scope fields. Therefore, it is possible to combine schema-based content in pre-defined fields with dynamic scope structured content within the same index.

In the query language you may specify individual fields, composite fields or scopes to limit the scope of a query. For scope queries the scope specification in the query must include the scope field name (also called the root scope) and sub-scopes within the indexed scope structure.

A scope field may include a hierarchy of scopes in arbitrary depth.

Scope Indexing is generic in the sense that it does not require any specific content input format. FAST InStream supports XML input format - other input may be supported by creating custom document processors.

Scope Data Types

Scopes may be of any supported FAST InStream data type. The data types are supported in a similar way as for individual fields. This means that string scopes support dynamic ranking mechanisms, linguistics, phrasing and wildcards as for composite fields.

Numeric scopes support exact matching and range matching mechanisms as for individual numeric fields.

Numeric matching requires that the query term and target numeric scope is of the same type. Matching a query term of type *float* with a scope of type *double* will not return any results. Therefore, it is required to apply consistent data typing in queries. Such data typing can be applied using explicit type conversion (for example, double(24.5)) or implicit default typing based on the term format. Refer to section *Query Operators* on page 28 for details on default types (if explicit type conversion is not used).

When querying non-scope numeric fields the system will know the type for the field, and will perform an automatic type detection based on the indicated field.

Query Language

Scope queries are only supported within the FAST Query Language (FQL). FAST InStream also includes two other query languages, the Simple Query Language and the Advanced Query Language. These two query languages do not support scope queries, but may be used for individual fields and composite fields even if the index also includes scope fields.

When searching content within scope fields, the query term or query expression must include a scope specification. This scope specification must include the name of the scope field and a scope within the scope field. From a query perspective, a scope is defined as any scope that matches the indicated scope name, including all sub-scopes.

A scope specification does not need to include the full path to the scope.

The scope query `root:date:foo` will search for the term 'foo' in all scopes with name *date* within the scope field named *root*, including all sub-scopes to such *date* scopes.

Query term expressions towards scope fields must include data types that are according to the data type of the target scope fields. When querying numeric scopes it is required that the indexed numeric scope is of the same type as indicated for the query term expression. Refer to Chapter 3 *The FAST Query Language*, section *Numeric operators* in the *Query Language and Parameters Guide* for details on literals and explicit type conversion.

Ranking

It is possible to utilize dynamic ranking also on queries involving scope fields. This includes:

- Proximity. The distance between query terms within the scope field impacts the resulting rank score. This only applies to scopes of type 'string', and must be explicitly specified in the query.

  Note! Proximity ranking applies to the scope field as a whole, that means that the distance between terms across sub-scopes also is taken into consideration. Therefore, this feature may not be applicable on highly structured content and metadata. For this reason it is recommended to split visible content and metadata into different scope fields in the index.

- Freshness. Freshness boost ranking is based on dates included in the source document. The freshness/date source must be included in an individual field outside the scope structure.

Refer to section *Mapping XML to one or more scope fields* on page 94 for further details on how to map information located in scopes to non-scope fields.

Note! Applying rank tuning via the BMCP is not supported for scope fields.

Result Sorting

Result Sorting is only available for non-scope fields. If you want to use one or more specific elements from the scope structure as sort criteria, it is possible to extract these elements from the source content (for example, XML elements) during content processing prior to the scope mapping and assign these to individual fields in the index. These fields must be enabled for sorting in the Index Profile. Refer to section *Mapping XML to one or more scope fields* on page 94 for further details.

Linguistics

Scope queries support the following linguistic features:

- Dictionary based automatic phrase and proper name detection
- Dictionary based Spell Check
- Lemmatization
- Synonym and spelling variations. This is a feature that enables expansion with spelling variations (for example, synonyms for product names, etc.). The feature is integrated with the Lemmatization stage and is enabled in the same way as lemmatization.

Dynamic Drill-down

Dynamic Drill-down (Navigation) in *FAST InStream* provides functionality for drilling down into the query result based on value distribution of one or more individual fields. Dynamic Drill-down can only be applied to non-scope textual or numeric fields, that is, it is not possible to apply this feature to scope fields as such.

If a specific element from the scope structure is desired to use for dynamic drill-down, it is possible to extract this element from the source content (for example, an XML element) during content processing prior to the scope mapping and assign this to an individual field in the index, with an associated Navigator specification. In this case the element may still be searchable within the scope structure, but may also be used for drill-down. Refer to section *Mapping XML to one or more scope fields* on page 94 for further details.

Categorization

Categorization in *FAST InStream* provides functionality for associating taxonomy information to each document in the index. The categorization feature requires that a non-scope string field contains the category information.

If a specific element from the scope structure is desired to use as the category source, it is possible to extract this element from the source content (for example, an XML element) during content processing prior to the scope mapping and assign this to an individual field

in the index. This field may then be associated with the category feature in the Index Profile. In this case the element may still be searchable within the scope structure, but may also be used for categorization. Refer to section *Mapping XML to one or more scope fields* on page 94 for further details.

Dynamic Document Summary (Teaser)

The dynamic document summary (teaser) is a short abstract of the matching document where the matching terms are highlighted within the context. *FAST InStream* supports creating dynamic document summaries for non-scope text fields and scope fields.

The 'dynamic' configuration attribute for the Scope field in the Index Profile defines if a dynamic document summary shall be generated or not.

The dynamic document summary feature is scope field aware, but not sub-scope aware. This means that for a document matching a given query:

- the dynamic document summary for a scope field only displays matching contexts with query terms that are within the same scope field
- the dynamic document summary will highlight query matches within all *string* scopes of the scope field. Detailed sub-scope matches will not be considered. This means that the document summary may highlight string scopes that do not match the sub-scope specification but match the query terms.
- The dynamic document summary identifies sub-scope boundaries, so that each unique text segment within a document summary is within one scope.

Metadata within the query (using the filter operator) will not be included in the dynamic teaser.

A 'fallback' configuration attribute may be used to define a non-scope text field as a fallback if it is not possible to create a dynamic document summary for a scope field. This works the same way as when using fallback for non-scope fields. This may for instance be the case if an OR query matches a document, but does not have any match within a particular composite field. Such a fallback is often denoted a *static document summary.* Such a static document summary may also be extracted from the source XML if appropriate. Refer to section *Mapping XML to Non-Scope Fields* on page 95 for details.

Partial Update

FAST InStream supports partial updates on scope fields as a whole, but not sub-trees within the scope field.

Dynamic XML Indexing

Dynamic XML Indexing implies mapping of XML content to the FAST InStream scope Indexing framework.

XML-to-Scope Mapping

FAST InStream provides a document processor (*XMLScopifier*) that can be configured to map any XML structure to a *FAST InStream* scope structure. The document processor can be configured to map one or more input document elements containing XML content to corresponding scope fields.

The document processor does not take into consideration the DTD, but will map all XML elements and attributes to scopes and sub-scopes within the scope field.

By default the scope representation does not differentiate between XML attributes and sub-elements. Both will be represented as sub-scopes. A configuration option in the document processor *XMLScopifier* enables a name differentiation between elements and attributes. If this option is enabled, the attribute names are by default prepended with a leading '@' which must be used if using the attribute name in queries. If you want to use a different symbol to differentiate between the names of elements and attributes, you need to specify this in the *XMLScopifier* configuration. For details on how to configure this, refer to Chapter 2 *Configuring the FAST Document Processing Engine* in the *Configuration Guide*.

A dedicated document processing pipeline template named *XML*, supporting XML to scope mapping, is included. By default the pipeline expects that the XML is included in the *data* document element.

Note! This is also the default option when using the FAST File Traverser.

By default all XML elements and attributes are indexed as text (type '*string*'). It is, however, possible to specify a data type for elements using a pre-defined attribute. The name of this attribute is configured in the *XMLScopifier* document processor. The values for this attribute and the mapping to FAST data types are also configurable.

Assuming that the attribute name is configured to be 'type:

```
<date type="datetime">2004-06-02</date>
```

This default data type support enables typing of elements, not attributes. All attributes will be treated as *'string'*. Other custom data type handling may be implemented by creating a

custom document processing stage. Refer to the *FAST Software Development Kit* for details.

Submitting XML

XML content may be submitted to FAST InStream using the FAST File Traverser or using the FAST Content API.

When using the File Traverser, it is only possible to submit a single XML object per searchable document. This XML object is mapped to a pre-defined document processing element within the FAST InStream processing framework.

When using the Content API, the client may submit any number of document elements that each contains a self-contained XML object. In this case the document processing pipeline may be configured to map each XML object to scope fields.

The term *XML object* in this context refers to a piece of XML content that is associated with a single searchable document.

The File Traverser accepts XML files that contain multiple XML objects, such as an XML file including a set of database records. The File Traverser is then configured with the name of the tag that encloses each XML object.

Mapping XML to one or more scope fields

Often it is preferable to separate the XML content into multiple scope fields. The reasons for doing this may be:

- Partitioning into multiple scope fields will improve search performance in case of large documents
- Dynamic ranking may be improved, for example, proximity. You should not apply dynamic ranking on for example, metadata.
- The dynamic document summary will be more precise. Refer to section *Dynamic Document Summary (Teaser)* on page 92.

The *XMLScopifier* document processor enables mapping of an XML object to one or more scope fields as configured in the Index Profile.

When using the File Traverser and the default XML submit methods in the Content API, the XML object appears within the document processing pipeline enclosed in the '*data*' element. Using the default configuration the *XMLScopifier* document processor will map this element to the scope field named *xmlcontent*.

There are two ways to map XML to multiple scope fields:

- Submitting multiple XML objects using the API. In this case you must submit a document consisting of more than one text element where each element contains an XML object. You must then create one instance of the *XMLScopifier* document processor associated with each XML object.
- Submitting one XML object that will be mapped to more than one scope field. In this case the *XMLScopifier* processor must be configured with an XPath specification that defines which sub-path is mapped to which scope field.

Mapping XML to Non-Scope Fields

Components from the same XML object may at the same time be mapped to scope fields and non-scope fields within the index. There may be several reasons for doing that:

- One or more scope elements are desired to use for dynamic drill-down (navigation from query results)
- The source for category information related to a document can be found within the XML object
- It is desired to extract a piece of text as the document title from the XML object
- The source for freshness (XML element containing a document date) or authority (an XML element containing citation references to this document) can be found within the XML object
- It shall be possible to perform explicit result sorting based on elements within the XML object

In this case the document processing pipeline must include two independent mapping/extraction stages:

1. A stage based on the XMLMapper document processor can be used to map components of the XML to individual string fields. The identification of XML components is specified using XPath format. One stage may be configured to map from a single XML object to multiple string fields.
2. The scope mapper stage will map the XML object to a hierarchical scope structure.

An example configuration file for XMLMapper is listed below. This extracts text to be used as title for the document from an XML file:

```
<XMLMappings>
  <SubTree base-path="/PLAY">
    <Mapping attr="title" path="MAINTITLE" />
  </SubTree>
  <SubTree base-path="/Research">
    <Mapping attr="title" path="//Title" />
  </SubTree>
  <SubTree base-path="/music">
    <Mapping attr="title" path="//title" />
  </SubTree>
</XMLMappings>
```

base-path defines what attribute(s) will be used as the base node for all mapping contained within this SubTree tag. This base-path is an XPath 1.0 path defining a single tag in the source XML document.

The Mapping element indicates the destination field (attr) and the actual XML path where the title text shall be extracted from.

Further Reading

The following parts of the FAST InStream documentation provide related information:

- Chapter 2 *Configuring the FAST Document Processing Engine* in the *Configuration Guide*
- *Appendix D* in the *Configuration Guide*

Chapter 10

Advanced Linguistic Processing

About this Chapter

This chapter introduces you to the basic concepts of advanced linguistic processing.

This chapter includes:

- Linguistics and Relevancy
- Dictionaries
- Automatic Language Detection
- Lemmatization
- Synonym Expansion and Spell Variations
- Proper Name and Phrase Recognition
- Spell Checking
- Anti-Phrasing
- Sub-String Search
- Wildcard Support
- Handling Special Characters

Linguistics and Relevancy

A fundamental issue in information retrieval is the relationship between queries and documents. From a linguistic perspective, it is useful to abstract from the exact words and focus on the information needs expressed within the query and the semantic content of the documents. The document's relevancy with respect to a query is not necessarily decided on the basis of words common to both query and document, but rather the extent that its content satisfies the user's need for information.

In order to achieve this, linguistic processing is performed both at the document level – during document processing – and at the query level – during query and result processing. On the query side linguistic processing results in a query transformation, on the document side, linguistic processing results in document expansion prior to indexing in order to cover grammatical forms and synonyms.

FAST InStream provides several linguistic features. They are described in the following sections.

Dictionaries

Some linguistic features depend on dictionaries. By default, FAST InStream provides dictionaries for lemmatization, entity extraction, spell checking including proper name and phrase recognition, synonym expansion, and variation expansion.

Table 10-1 lists these linguistic features, and to what extent they need custom dictionary configuration.

Table 10-1 Advanced Linguistic Features and Dictionaries

Linguistic Feature	Required Custom Dictionary Configuration
Lemmatization	Customization optional. Refer to Chapter 4 *Configuring Linguistic Processing* in the *Configuration Guide* for details.
Entity Extraction	Customization optional. Contact FAST Professional Services for support.
Spell Checking	Customization recommended. Refer to Chapter 4 *Configuring Linguistic Processing* in the *Configuration Guide* for details.
Synonym and Spelling Variation	Customization recommended. Refer to Chapter 4 *Configuring Linguistic Processing* in the *Configuration Guide* for details.

For details on how to edit dictionaries, refer to Chapter 4 *Configuring Linguistic Processing* in the *Configuration Guide*.

Automatic Language Detection

Overview

During document processing, documents can be analyzed to detect the language in which they are written. This is provided by the Automatic Language Detection feature.

Detecting the language of a document is essential to all other linguistic analysis features, as the resulting language information is used to select language-specific dictionaries and algorithms during document processing and query processing.

During language detection, a given document is analyzed for all supported languages. For each language, a certain score is calculated, based on the occurrences, number, and length of certain test strings. The language that reaches the highest score is specified as the document's language.

Note! For queries, the language has to be explicitly set by the end-user or search application, as the query itself generally provides too little context for determining the language it is written in.

Applying Automatic Language Detection

Overview

Automatic Language Detection is applied during document processing. It is automatically executed on every document that passes one of the following document processing pipelines:

- NewsSearch
- SiteSearch
- Generic

These document processing pipelines contain document processing stages that check each document for the HTML language metadata, for the language used in the document's HTML text, and the URL. If the language given in the HTML language metadata turns out to be different from the language detected through the language analysis of the HTML text, then the automatically detected language is chosen.

When the document is analyzed, the following document elements are appended to it:

- `language`, specifying the primary language detected in the document

- `secondarylanguage`, specifying the secondary language detected in the document
- `languages`, specifying the list of all detected languages in the document

These elements are referred to later on when other linguistic features are applied.

Default Language

If the language of the document cannot be determined, a value of "un" for UNKNOWN will be specified for each of the document elements listed above.

Required Custom Dictionaries

Language detection does not require any custom dictionaries.

Functional Characteristics

Supported Languages

Automatic language detection is available for the following languages:

Table 10-2 Automatically Detected Languages

Language	Code	Language	Code
AFRIKAANS	af	ALBANIAN	sq
ARABIC	ar	ARMENIAN	hy
AZERI	az	BANGLA	bn
BASQUE	eu	BOSNIAN	bs
BRETON	br	BULGARIAN	bg
BYELORUSSIAN	be	CATALAN	ca
CHINESE_SIMPLIFIED	zh-simplified	CHINESE_TRADITIONAL	zh-traditional
CROATIAN	hr	CZECH	cs
DANISH	da	DUTCH	nl
ENGLISH	en	ESPERANTO	eo
ESTONIAN	et	FAEROESE	fo
FARSI	fa	FILIPINO	tl
FINNISH	fi	FRENCH	fr

Table 10-2 Automatically Detected Languages

Language	Code	Language	Code
FRISIAN	fy	GALICIAN	gl
GEORGIAN	ka	GERMAN	de
GREEK	el	GREENLANDIC	kl
HAUSA	ha	HEBREW	he
HINDI	hi	HUNGARIAN	hu
ICELANDIC	is	INDONESIAN id	
IRISH_GAELIC	ga	ITALIAN it	
JAPANESE	ja	KAZAKH	kk
KIRGHIZ	ky	KOREAN	ko
KURDISH	ku	LATIN	la
LATVIAN	lv	LETZEBURGESCH	lb
LITHUANIAN	lt	MACEDONIAN	mk
MALAY	ms	MALTESE	mt
MAORI	mi	MONGOLIAN	mn
NORWEGIAN_BOKMAAL	nb	NORWEGIAN_NYNORSK	nn
POLISH	pl	PORTUGUESE	pt
RHAETO_ROMANCE	rm	ROMANIAN	ro
RUSSIAN	ru	SAMI_NORTHERN	se
SERBIAN	sr	SLOVAK	sk
SLOVENIAN	sl	SPANISH	es
SWAHILI	sw	SWEDISH	sv
TAMIL	ta	THAI	th
TURKISH	tr	UKRAINIAN	uk
URDU	ur	UZBEK	uz
VIETNAMESE	vi	WELSH	cy
YIDDISH	yi	ZULU	zu

Handling Languages with Multiple Language Codes

Chinese, English and Norwegian may be represented by several language codes. These codes are not applied identically during document processing and query processing. Therefore, the following considerations apply for language specific tokenization and lemmatization:

- Chinese
  - The two language codes `zh-simplified` and `zh-traditional` will be used during language detection and must also be used when you indicate the query string language on the query side. That means that you must specify one of these two codes in the query parameter `language` in order to get the correct tokenization and match desired documents.
  - If you specify `zh` only in the query parameter `language`, both traditional Chinese and simplified Chinese fields will be retrieved.
  - The ISO-639 language codes `zh-tw` and `zh-cn` will be mapped to `zh-traditional` and `zh-simplified`, respectively.
- Norwegian:

  The language detection document processor is able to detect two variants of Norwegian (`nb` for Bokmål and `nn` for Nynorsk). This differentiation is not applied on the query side. Therefore, you need to use the language code `no` in queries to get the correct lemmatization, provided you are using lemmatization by reduction.

  Note! You must, however, use the specific language codes ("nn" or "nb") if you want to limit the query result to all Norwegian documents. This can be done as follows in FQL:

  ```
  FQL: AND(FILTER(languages:OR("nn”,”nb")), <your ordinary query expressions>)
  ```

- English:

  The ISO-639 language codes for the different types of English, such as `en-us` or `en_UK` are not supported. This applies both to language detection (document processing) and query processing.

Encoding

During language detection, the encoding of the document is detected as well. Encoding information is essential to normalize documents to a Unicode encoding (UTF-8). All other advanced linguistic features operate on Unicode encoded documents and queries.

Document Meta Data and Automatically Detected Language

Meta data declaring the document language and encoding such as the HTML attribute *lang="lang"*, overrides the automatic detection if the meta data language passes a plausibility check. That means that the document is run through automatic language detection to check whether the document actually is written in this language or not. To determine this, a specific, very low acceptance score is used for the language specified by the meta tag. If the result score for this language passes the acceptance score, the language is assigned to the document. If the result score of the language specified in the meta tag does not pass the acceptance score, the language that achieved the highest result score is assigned to the document.

Running such plausibility checks against documents containing meta data about the document's language is necessary in order to handle HTML in which the metadata is copied from another document or for other reasons is completely misleading.

Lemmatization

Overview

Generally speaking, lemmatization means the mapping of a word to its base form and / or all its other inflectional forms, such as

- singular or plural for nouns,
- tense and person for verbs,
- positive, comparative, or superlative forms for adjectives.

The purpose of lemmatization is to enable that a query for one of the possible forms of a word will also match documents that contain a different form of the word.

This allows a user to search for a query like car and to get both documents that contain the word car as well as documents that contain the word cars.

In contrast to stemming or wildcard search, which would match all documents containing words starting with car, such as cared or career, lemmatization allows for recognizing words as matching terms on basis of their being inflectional variations of the query word. With this, lemmatization also takes irregular inflections such as tooth and teeth into account.

Lemmatization by Expansion and Lemmatization By Reduction

Overview

FAST InStream provides support for two types 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. There is no lemmatization in the query.
- 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 the basis for handling lemmatization of Japanese and Korean documents.

Strengths and Weaknesses of the Different Types of Lemmatization

Lemmatization by expansion and lemmatization by reduction have both strengths and weaknesses. Table 10-3 gives you an overview:

Table 10-3 Strengths and Weaknesses of the Different Types of Lemmatization

	Lemmatization by expansion	Lemmatization by reduction
Disk space consumption	High, especially for languages with high count of forms per word, such as Russian or Finnish.	Low due to smaller dictionaries.
Impact on document processing or query processing	Only document processing is affected.	Document processing and query processing must be synchronized.
Language	Independent of the language of the query term.	The language of the query term needs to be known.
Implicit and Explicit Proximity	Lemmatization by expansion supports implicit and explicit proximity for scope field queries. Lemmatization by expansion does not support implicit and explicit proximity for composite field queries where the matching term is an inflected form of the query term.	Lemmatization by reduction fully supports implicit and explicit proximity.

Note! For information on how lemmatization impacts memory consumption, refer to Chapter 7 *Benchmarking*, section *Lemmatization Memory Consumption* in the *Deployment Guide*.

Default: Lemmatization by Reduction

Lemmatization by reduction is the default type of lemmatization.

Refer to Chapter 4 *Configuring Linguistic Processing* in the *Configuration Guide* for details on how to change the lemmatization mode.

Lemmatization and its Influence on the Search Experience

Enabling lemmatization has the following effects:

- Recall is improved: Documents that would not match since they contain a word form that is different from what the end-user sent to the system will now be retrieved as well. For example, an end-user looking for "laptops" on a shopping site will also get documents that only contain the word "laptop".
- The improved recall will also influence the ranking: Documents will be ranked regardless of whether they contain the word the end-user sent to the system or one of its inflected forms. In either way, the documents ranked at the top of the result list will be the most relevant documents.
- These effects are multiplied for multi-word queries where every component might occur in different forms.

Table 10-4 shows the benefits of lemmatization referring to an example search for the term car:

Table 10-4 Benefits of Lemmatization Enabled Searching

Document	Lemmatization disabled	Lemmatization enabled	Benefit
Does not contain car, only cars in body.	no hit	hit with low rank	improved recall
Contains car in body and fast cars in title.	hit with low rank	hit with high rank	improved precision

Dictionaries for Lemmatization

The Core Lemmatization Dictionaries

FAST InStream provides several dictionaries with a varying coverage of categories:

- lemmatization for nouns only (N),
- lemmatization for nouns and adjectives (NA),
- lemmatization for nouns and verbs (NV),
- lemmatization for nouns, adjectives, and verbs (NAV).

The following gives a short example of the effects of the different dictionaries. Assume that the original document contains the phrase:

"This download allows developers to run quick tests on our server."

For this example, the following words will be added to the index by the different dictionaries:

Table 10-5 Lemmatization Dictionaries and their Influence on the Index

Extent of Dictionary	Output
N	downloads developer runs test servers
NA	downloads developer runs quicker quickest test servers
NV	downloaded downloading downloads allowed allowing allow developer running runs ran tested testing test servers
NAV	downloaded downloading downloads allowed allowing allow developer running runs ran quicker quickest tested testing test servers

What Dictionary to Choose

One factor that influences the choice of a dictionary is index size and indexing time. From the example above, using a dictionary with all possible forms will increase the number of index terms.

Another important factor is precision: Using a dictionary with both noun and verb forms may lead to ambiguities, as can be seen in the expansion for download in the above example: Although download is used as a noun in the source phrase, the verb dictionary component will also expand it to downloaded and downloading since morphologically, download could also be a verb in the present tense. The consequence in this example is that a query for downloaded would also match this document, even though the verb form does not really appear in the original phrase. This effect may sometimes be desired, sometimes not; you will need to decide which dictionary fits your purpose best.

As a general rule, it is recommended to use the noun or noun and adjective dictionaries for generic applications. Dictionaries with verbs are most useful for applications whose content is action-centered, that means where verbs may carry a lot of semantic information. An example for this may be a user help-desk, where users often will use verb phrases to describe their problems, like in the query the printer stops to work after filling the paper tray. In this case, it might make sense to also expand the verbs into their alternative forms so that a query for fill paper tray stop working will match the above phrase as well.

The Lemmatization Stopword Dictionary

In addition to the core lemmatization dictionaries, there is a lemmatization stopword dictionary. It contains words that are to be excluded from lemmatization.

FAST InStream provides stopwords lists for the following languages:

- Danish
- Dutch
- English
- Finnish
- French
- German
- Italian
- Japanese
- Norwegian
- Portuguese
- Spanish
- Swedish

Note! Words that are listed in the lemmatization stopword dictionary are excluded from lemmatization, but not from the query string as such. That means that these words are not lemmatized, but still mapped to the search index.

Applying Lemmatization

Overview

Lemmatization is enabled on a search engine cluster and document basis by modifying the index profile (see section *Defining how Documents are Searchable: Index Profiles* on page 40 and Chapter 3 *Index Profile Features Management* in the *Configuration Guide*). In addition, the end-user can enable or disable lemmatization on a per-query basis, provided the search front-end allows for this selection option.

During lemmatization, two indexes are created for the documents to be indexed:

- the usual search index, an index that indexes the documents in their original form,
- and a lemma index, an index that indexes the documents and all inflections of the words the documents contain.

The rank capabilities of the two indexes are identical. This means that, for instance, the same differentiation of dynamic ranking between *title* and *body* is maintained in both cases.

When the end-user sends a query to the system with lemmatization enabled, the query is sent to the lemma index, and the list of results will contain documents that match both the original query term as well as its inflected forms.

When the end-user sends a query to the system with lemmatization disabled, the query is sent to the original search index, and the list of results will contain documents that match only the original query term.

Required Dictionaries

The following dictionary files are involved (required or optional) in lemmatization:

- <2-character language code>_NA<mode> (default)
- <2-character language code>_N<mode> (optional)
- <2-character language code>_NAV<mode> (optional)
- <2-character language code>_stopwords

where <mode> is either '_red' (for lemmatization by reduction) or '_exp' (lemmatization by expansion).

For details about how to configure and apply lemmatization and how to create a custom lemmatization stopword dictionary, refer to Chapter 4 *Configuring Linguistic Processing* in the *Configuration Guide*.

Functional Characteristics

Supported Languages

Lemmatization is available for documents in the following languages:

- English
- French
- German
- Spanish
- Italian
- Portuguese
- Japanese (contact FAST Professional Services)

- Korean (contact FAST Professional Services)
- Polish (contact FAST Professional Services)
- Hungarian (contact FAST Professional Services)
- Hebrew (contact FAST Professional Services)
- Arabic (contact FAST Professional Services)

Level of Lemmatization

The level of lemmatization can be selected by using different dictionaries for the lemmatization document processor. The default level in FAST InStream is normalization of nouns with respect to singular and plural word forms and adjectives with respect to singular, plural, and gender. Dictionaries for other levels of lemmatization may be provided upon request.

Lemmatization and Matching

Matching is done in a case-insensitive fashion, that means all input is converted to lowercase, and all entries in the dictionaries are stored in lowercase.

Lemmatization and Proper Name and Phrase Recognition

Lemmatization and Proper Name Recognition cannot be applied on the same query term at the same time.

Query terms that are recognized as proper names or phrases are not redirected to the lemma index. These terms are matched only against the usual search index. For example, *FAST Search* may be included in the list of proper names, which would exclude the inflections *fasts* and *searches* in the lemmatized index. Likewise, a search for *FAST*, recognized as a proper name, will not be expanded.

This means that in a standard FAST InStream configuration and search front-end, lemmatization is available for the default search index in the `any word` and `all words` query modes, but not in the `exact phrase` mode.

Note! When proper name and phrase recognition and lemmatization are applied simultaneously to a query, proper name and phrase recognition overrides lemmatization. Thus, if your search front-end provides both lemmatization and proper name and phrase recognition not as mutually excluding functionalities, but as options that can be selected simultaneously, proper name and phrase recognition overrides lemmatization. It is therefore recommended to provide these two selections as mutually excluding radio buttons on your search front-end.

Limitations

The following limitations apply to lemmatization:

- Proper names

  If a proper name is identical to an inflected word form, a query for this name will return also documents containing other inflectional forms of the word.

- Dynamic Document Summary (Teaser)

  If the query term only match an inflected form in the document, the matching term will not be highlighted in the Dynamic Document Summary.

Synonym Expansion and Spell Variations

Overview

Synonym and spell variation expansion works similar to lemmatization – a document to be indexed is expanded with a defined list of synonyms or spell variations to the words it originally contains. As with lemmatization, the original document is indexed in the original search index, whereas the expanded document is indexed in a separate expanded index. This allows you to control enabling synonym expansion on a per-query basis: You can decide whether a query is to be executed with synonym expansion, in which case the query is sent to the synonym index, or without synonym expansion, in which case the query is sent only to the original index.

Note! As with lemmatization, synonym expansion can be applied to either documents or queries. Document-side synonym expansion, however, is the default synonym expansion strategy. All further descriptions refer to the document-side type of synonym expansion. For further information on how to apply query-side synonym expansion, contact FAST Professional Services.

Applying Synonym and Spell Variation Expansion

Overview

Synonym and spell variation expansion is applied during document processing. It is part of the document processing pipeline you specify for your collection.

Searching in synonym expanded documents may be enabled on a per query basis.

Involved Processors

The following document processor is involved in synonym expansion:

- *Lemmatizer* to expand selected document attributes based on the language specific dictionary

Required Dictionary Files

Synonym and spell variation expansion requires you to create a synonym dictionary.

Limitations

By default, synonyms and lemmatized forms are placed in the same lemma index. This means that synonym and spell variation expansion and lemmatization are tied together in the query analysis and can only be enabled or disabled together.

For procedural details about how to configure and apply synonym expansion, refer to Chapter 4 *Configuring Linguistic Processing* in the *Configuration Guide*.

Proper Name and Phrase Recognition

Overview

Proper name and phrase recognition is based on a mapping of the query terms against a dictionary of names and phrases, whose content you can modify.

Proper name and phrase recognition includes phrase detection, as phrases are treated as proper names.

Proper names are words and phrases that are important to both a query and the content you want to make searchable. Typical proper names are product names, trademarks, product models, part numbers, promotion codes, or stock keeping units. In general, proper names are not part of a language's usual vocabulary, such as expressions like *CJK-400ex*. Furthermore, proper names or phrases can be words of a language that have a particular semantic value within the content, such as expressions like *Data Search*. In either case, proper names and phrases are protected from lemmatization and anti-phrasing.

The effects on relevancy are improved recall and precision.

Functional Characteristics

Query Transformations

Proper Name and Phrase Recognition applies the following transformations to a query:

- It detects implicit phrases and proper names in the query and phrases them explicitly by adding quotation marks ("the phrase"). This means that the detected phrase is protected from further query transformation. In addition, the query will return phrase matches only. By creating for instance a list of product names, you may ensure that queries are directed to the desired pages that match the implicit product name phrase.
- It detects and corrects misspelled phrases and words. Implicit phrases in the query will be spell checked and corrected. If the dictionary contains the phrase "nissan micra", the queries "nissan macra" will be detected as a misspelling and corrected to "nissan micra". The spell check will even detect the phrase if both terms are misspelled. The phrase "nisan macra", for instance, would then be corrected to "nissan micra".
- It detects and corrects query terms with alternative spell grouping. If the dictionary contains the term "thinkpad", a query "think pad" will be corrected to "thinkpad". If the dictionary contains the term "alpha server", a query "alphaserver" will be corrected to "alpha server".

Customization

The set of proper names and phrases is customizable. In addition, a list of exceptions allows you to fine-tune terms that are close to both proper names and valid words in the supported languages.

Proper name and phrase recognition may be applied in several sequential steps, which may use increasingly broader dictionaries. This way, you can for example apply proper name and phrase recognition starting with a narrow list of company specific product names followed by domain specific terms such as computing, pharmaceutical, or engineering terms.

For details about customizing proper name recognition, contact FAST Professional Services at FAST Support.

Proper Name Recognition and Spell Checking

Proper name recognition is also used for spell checking (see *Spell Checking* on page 118). A query term that is close to a proper name is replaced with the proper name. The spell checking is also applied to phrases, including word splitting and joining. Thus, "datasearch" or "ffast" for example are recognized as "data search" and "fast" respectively if the dictionary of proper names includes data search.

Proper name and phrase recognition provides also a list of exceptions that avoid spell checking of words that are similar to the defined proper names. For example, assuming your content contains the product name eServer, then the English word server should probably not be changed to eServer. In this case, you add the word server is added to the exception list for proper name and phrase recognition.

Note! It is recommended that you modify the exception list on the basis of past queries.

Proper Name and Phrase Recognition and Lemmatization

Proper name and phrase recognition and lemmatization are mutually excluding functionalities. If both functionalities happen to be selected, proper name and phrase recognition overrides lemmatization.

Applying Proper Name Recognition

Proper name recognition is applied as part of the Advanced Spell Checking (see section *Advanced Spell Check* on page 118). The search string the end-user submitted to the system is analyzed. Depending on the configuration of the FAST Query and Result Processing Engine, the result of the query analysis is either sent directly to the FAST Search Engine or sent back to the end-user as feedback.

For details about how to apply proper name recognition, refer to Chapter 4 *Configuring Linguistic Processing* in the *Configuration Guide*.

Spell Checking

Overview

Spell checking works similarly to proper name recognition. The purpose of spell checking is to improve the quality of the queries by comparing each query term against dictionaries and identifying misspelled query terms. As a result of the spell checking process, FAST InStream either replaces the query terms automatically with the correct terms, or it suggests modifications to the query terms to the end-user.

The spell checking algorithm operates on individual query segments. A query segment is a portion of the query that forms a syntactical entity of some kind. For example, if something within the query is put in quotes, that quoted part forms a query segment. Additionally, the - and + modifiers in the simple query language also introduce segment breaks, as do explicit field specifications. This means that implicit phrases will not be detected across query segments.

In contrast to proper name recognition, spell checking does not protect the corrected terms from further processing.

Spell checking a query is executed in two stages: First, an Advanced Spell Check is performed, followed by a Simple Spell Check. The following sections describe these two stages in more detail.

Advanced Spell Check

During the Advanced Spell Check stage, the query terms are run through Proper Name and Phrase Recognition (see *Proper Name and Phrase Recognition* on page 115). FAST InStream supplies a default dictionary containing names of persons, names of places, names of companies, and other common phrases. You can extend this dictionary with your own custom phrases, for instance product names.

The Advanced Spell Check stage enables all query transformation capabilities included in Proper Name and Phrase Recognition (see *Query Transformations* on page 115).

Simple Spell Check

The Simple Spell Check stage supports spell checking of individual terms against language specific dictionaries (see *Supported Languages* on page 122).This spell check stage will only detect misspelling of single words, not for phrases.

Applying Spell Checking

Overview

Spell checking is applied during query processing.

You enable and configure spell checking by

- adapting the required dictionaries' source files to your content and end-users' needs
- compiling the dictionaries
- configuring the appropriate query transformer.

For details, refer to the *Configuration Guide*.

Required Dictionaries

Both Advanced and Simple Spell Checking require a set of dictionaries. Refer to Chapter 4 *Configuring Linguistic Processing* in the *Configuration Guide* for dictionary file locations.

Advanced Spell Check Dictionaries

The following dictionaries support advanced spell checking:

- phrase dictionaries

  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").

  You may modify the supplied phrase dictionary by adding or removing terms. Alternatively, you may create separate phrase dictionaries that 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 Check 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.

- the phrase exception list

FAST InStream supplies a default phrase exception list 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.

You can adapt this phrase exception list to suit your content.

Note! All phrase dictionaries are language-independent. Note however that the default phrase dictionaries supplied with FAST InStream are optimized for English.

Simple Spell Check Dictionaries

The following dictionaries support advanced spell checking:

- single word dictionaries:

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

  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.

  You may modify the supplied single word dictionaries by adding or removing terms.

- 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 exactly matches an entry in the exception list, the term will be protected from Simple Spell Check.

  In contrast to the phrase exception list, the single word exception lists are language specific.

Functional Characteristics

Supported Languages

Dictionaries for spell checking are provided for the following languages:

- English
- French
- German

- Spanish
- Italian
- Portuguese

Note! During installation, you select which of the supported languages is to be enabled. To change this after installation, contact FAST Professional Services.

Spell Checking and Proper Name and Phrase Recognition

Spell checking and proper name and phrase recognition are tied together closely. A query term that is close to a proper name is replaced with the proper name.

The proper name recognizer has a list of exceptions that avoid spell checking of words that are similar to the defined proper names.

Anti-Phrasing

Overview

Anti-phrasing removes common phrases from the query strings. These common phrases are defined in the anti-phrasing dictionary. This way, query strings like "Who is Miles Davis?" are reduced to "Miles Davis", which improves query recall, particularly in the *all words* query mode. In the *any words* query mode, anti-phrasing has less effect on the results. It may still enhance precision as it may reduce result rank for documents with many irrelevant occurrences of *who is* in parts of the document where "Miles Davis" does not appear.

Note! Anti-phrasing is closely related to the concept of stopwords. In contrast to stopwords, however, the anti-phrasing feature does not remove single words, but entire phrases only. Removing single words implies the risk of removing important words that happen to be identical with stop words. Phrases, in contrast, are more unambiguous and can therefore be removed from the query more safely.

Applying Anti-Phrasing

Overview

Anti-phrasing is applied during query processing.

Required Dictionaries

The following dictionary is involved (required or optional) in anti-phrasing:

- default anti-phrasing dictionary

Functional Characteristics

Supported Languages

Anti-phrasing is supported for the following languages:

- English
- French
- German
- Spanish

- Italian
- Portuguese

Note! During installation, you select which of the supported languages is to be enabled. To change this after installation, contact FAST Professional Services.

Sub-String Search

Overview

What Sub-String Search Is

FAST InStream supports sub-string search, that means searching for parts of a string as with a wildcard search ("*term*").

Sub-string search is based on a specific composite field configuration within the index profile. By setting a composite field to be a sub-string field, you enable your end-users to search for sub-strings of arbitrary lengths and at arbitrary positions inside the indexed content.

How Sub-String Search Works

When enabled, sub-string search is applied to both document and query. For a field in the index profile that is specified for sub-string search, each word or token (for Asian language documents) in the field is split up in smaller entities, so called *sub-strings*, consisting of a defined number of signs. As an example, the word "midsummer" is split up into the sub-strings "mids", "idsu", "dsum", "summ", "umme", and "mmer", provided the specified number of characters the sub-strings are supposed to have is four.

This allows the end-user to search for example for the query "summer" and to find a document that actually contains the word "midsummer". For as with the words of the document, the end-user's query is split up into the sub-strings "summ", "umme", and "mmer". During the matching process, the document containing the word "midsummer" with all its sub-strings and the query "summer" will give a match, because both contain common sub-strings.

You may configure the length of the sub-strings into which a word or token is to be split. In addition, you may configure whether white space or other non-word characters functioning as word separators are to be stripped away, so that sub-strings across words are matched as well.

Note! Sub-string search is not supported for scope fields.

Application Scenarios

Sub-string search is useful for application scenarios like the following:

- For certain database applications it may be desirable to be able to search for sub-strings within individual fields, such as product code fields, or name fields.
- Many languages combine several individual words into new words. German, for instance, uses this mechanism a lot. Sub-string search allows your end-user to find documents containing the word "Staatsanwaltschaft" using the query "anwalt".
- In text written in Chinese, Japanese and Korean, there are commonly no spaces separating individual words. To tokenize documents that are written in one of these languages, FAST InStream uses a specific, language sensitive tokenizing document processor in order to find logical places for word boundaries.

  However, sometimes the process of finding word boundaries is ambiguous, as your end-users may want to search for words going beyond what this tokenizer can output. In these cases, the sub-string search functionality enables queries that are not sensitive to this tokenizer, but go across the word boundaries the tokenizer has come up with, thus matching any sequence of characters.
- Besides Chinese, Japanese, and Korean, there are other languages that do not use space between words either, such as Thai, and for which dedicated tokenizers are not provided. In this case, sub-string search still allows the end-user to search for individual words.
- In certain scenarios, documents have insufficient logic in it to allow for useful word splitting. Examples are DNA-strings and musical midi-descriptions. In these cases, sub-string search allows the end-user to search through these types of documents.
- Sometimes separating characters in a word with spaces are used to emphasize the word, as in the phrase "His name was E L V I S". In this case, sub-string search allows the end-user to find a document containing this phrase by searching for "Elvis".
- Some acronyms may have different spellings, like "D.N.A." and "DNA" or "dna". Sub-string search may then allow the end-user to find documents containing "D.N.A." by searching for "dna" and vice-versa.

Note! A side effect of this tokenization is that word boundaries are not detected. This means that a query "*erni*" would also match the text "Midsummer Night". As indicated in the application examples above this may sometimes be desired, but may also create undesired matches. This means, that this implies that sub-string search is not always meaningfully applicable for usual text documents of reasonable size, as the probability for such undesired matches across word boundaries increases with document size. For structured data and Asian language encoded documents, though, sub-string search is a reasonable solution.

Applying Sub-String Search

Sub-string search is enabled by defining the relevant fields as subject to sub-string search in the index profile.

You may configure the length of the sub-strings into which a word or token is to be split. For Western languages, the recommended length is at least four characters. For Asian languages the recommended length is two to three.

In addition, you may configure whether white space or other non-word characters functioning as word separators are to be stripped away, so that sub-strings across words are matched as well.

Wildcards are implicit, meaning that you get the same results by searching for "summer" as you get when you search for "*summer*". Apart from that there is no impact on how you define queries.

Functional Characteristics

Supported Languages

Sub-string search is language independent.

Sub-String Search and Index Size

The generated index size is reasonably larger for sub-string composite fields than for normal composite fields.

Sub-String Search and Dynamic Teasers

Applying dynamic teasers is only possible when searching against fields that are defined as normally searchable fields.

Wildcard Support

Overview

With full wildcard support it is possible to use '*' and '?' when specifying a query-term, where '*' indicates any number of wildcard characters and '?' indicates a single wildcard character. The wildcard characters may be anywhere in the query term. This means that FAST InStream 4.1 supports single character wildcards, prefix, suffix and substring search.

Configuring Wildcard Support

Wildcard search is enabled by defining the relevant fields as subject to wildcard search in the index profile. For details refer to the *Configuration Guide*.

Wildcard support is defined for an individual string field or a composite field in the Index Profile. It must be configured explicitly as it will have impact on disk usage.

Handling Special Characters

Overview

By default, special characters, such as characters with accents or language specific characters, are preserved in both documents, dictionaries, and queries. This means that words that contain special characters, are treated as different words than their normalized variants.

Character Normalization

To enable character normalization, contact FAST Professional Services at FAST Support.

Further Reading

The following parts of the FAST InStream documentation provide related information:

- Chapter 8 *Processing Queries and Results*, section *Processing Queries and Results*
- Chapter 12 *FAST InStream in an Asian Language Environment*
- Chapter 4 *Configuring Linguistic Processing* in the *Configuration Guide*
- *Query Language and Parameters Guide*

Chapter 11

Categorizing Content and Search Results

About this Chapter

This chapter introduces you to the basic concepts of content categorization and explains how you can provide category features such as dynamic drill-down or clustering to your end-user.

This chapter includes:

- Concepts of Categorization
- Categorizing Documents
- Categorizing Results
- Taxonomy Management

Concepts of Categorization

This section introduces you to the basic concepts and terms of categorization.

Categorization

Categorization is the process of assigning documents to specific categories. These categories may derive from a given hierarchy, a so called taxonomy, or be calculated on the fly based on similarities between documents.

Note! A taxonomy is allowed to be flat, meaning that there is no notion of subclasses or super-classes.

The categorization process is performed on the basis of pre-defined rules. Note that a document is allowed to belong to more than one category.

Clustering

Clustering means the automatic detection of groups (clusters) of documents that have somehow similar content.

There are two types of clustering:

- Unsupervised clustering implies that neither the names of the clusters nor any information about the cluster structure is known in advance, but is automatically computed from the set of given documents themselves. Since no set of example documents are used in this process, such clustering is sometimes also referred to as unsupervised learning.
- Supervised clustering implies that existing knowledge of categories within a result set is used for the clustering process. This means that non-categorized documents will be clustered with similar documents within existing categories. In this way the documents are categorized on the fly to existing categories based on document similarities.

Document Similarity

Both categorization and clustering are based on a notion of document similarity: Similarities between the content of documents are computed, and decisions to place documents into classes or clusters are done on the basis of these similarities. Document similarities are computed by comparing the vectors of documents. A document vector is a kind of doc-

ument signature, representing a document's content in a way that allows comparison between documents.

The clustering in FAST InStream operates on the basis of the set of documents returned from the Search Engine. For example, for an incoming query, FAST InStream may be configured to behind-the-scenes retrieve the 100 highest ranked documents for that query. Some percentage of these documents, say 20, are tagged as belonging to some set of predefined classes, while the remaining 80 do not have any such information associated with them. FAST InStream can try to classify these 80 documents into the fragment of the overall taxonomy that the other 20 documents span. Some number of these 80 documents, say 50, will be successfully classified, while a good fit could not be found for the remaining 30 documents. Rather than giving up, these 30 documents can be passed on to a clustering or unsupervised learning algorithm within FAST InStream that detects groups within these 30 documents, and devises suitable names for the clusters.

FAST InStream can be configured to do both categorization and clustering as described, or only categorization or clustering. What is input to the clustering phase is also configurable. For example, if both categorization and clustering are enabled, we can choose to cluster only those documents that could not be classified, or we can apply the clustering algorithm to the full set of retrieved documents.

Document Similarity Vectors

A document vector is a representation of the unstructured textual content that we associate with a document. This representation is used when analyzing similarities between documents. In FAST InStream similarity vectors are used for when performing *Find Similar* queries, and *Unsupervised Clustering*.

Vectorization is performed as part of the document processing, using the *Vectorizer* document processor. This document processor is a standard part of all document related processing pipelines in FAST InStream.

Basically, a document vector is a set of (keyword, weight)-pairs, where keyword is a word or a phrase associated with the document, and weight is a numerical measure of how important keyword is for the document. For efficiency, the extraction of keyword strings for a document can be done at indexing-time. The weight values are also computed at indexing-time, but can be refined at query-time to reflect importance relative to the set of search results, for example.

A document vector for a news story might contain the following pairs:

(Nobel prize, 1), (Nelson Mandela, 0.78), (Oslo, 0.5)

Vectorization is composed of two sub-tasks: How to select the strings s, and how to determine the weights w.

By default, the document vector is computed based on the "title" and "body" field of a document, where "title" is assigned more weight than "body" within the vector computation.

The vectorization process is primarily designed for unstructured textual data. For structured data, for example, database content, it is more relevant to use URI or rule based categorization and the Dynamic Drill-Down feature.

The vectorization process consists of the following main steps:

1. The vectorizer reads the `language` attribute, which enables selection of the correct stop word list and/or token boosting heuristic to use for this language.
2. A simple tokenizer is applied to the text found in specified fields. Which fields the tokenizer is to work on, is specified in the tokenizer's configuration.
3. A weight is assigned to each token based on the configured rules. Stopwords will get lower weight, and words including upper case letters will get higher weight.
4. The default vectorizer algorithm is based on detecting bigrams in the token stream, that is, detecting phrases of length two, with some exceptions for allowing unigrams to slip through. Bigrams are only detected within sentences, and will not include stopwords.
5. The vectorizer extracts strings (by default bigrams) from the token stream, and passes `(string, weight)` pairs to the statistics-collection routines. The same string may be passed many times, possibly with differing weights. The statistics-collection routines accumulate all the scores for all the strings, and are responsible for computing a final weight for each string. A string is considered important within a document based on the weight of each token and is also related to the frequency within the document.
6. The document is updated with a new field containing a textual representation of the document vector. By default, the name of this output field is docvector.

Note that the vectorizer component may be configured to use any other field if a specific application requires this. The field input to the vectorizer can be any textual fields found in the document, or fields that are computed from the document by other document processors that precede the vectorizer in the document-processing pipeline.

It is also possible to configure the vectorizer to use other algorithms, and also to align the weight of the similarity vector across a larger document set. Please consult FAST if you have other requirements within your application.

Categorization Features

FAST InStream offers a variety of functionalities to support and take advantage of categorization. Figure 10-1 illustrates these features and shows to which subsystems they relate.

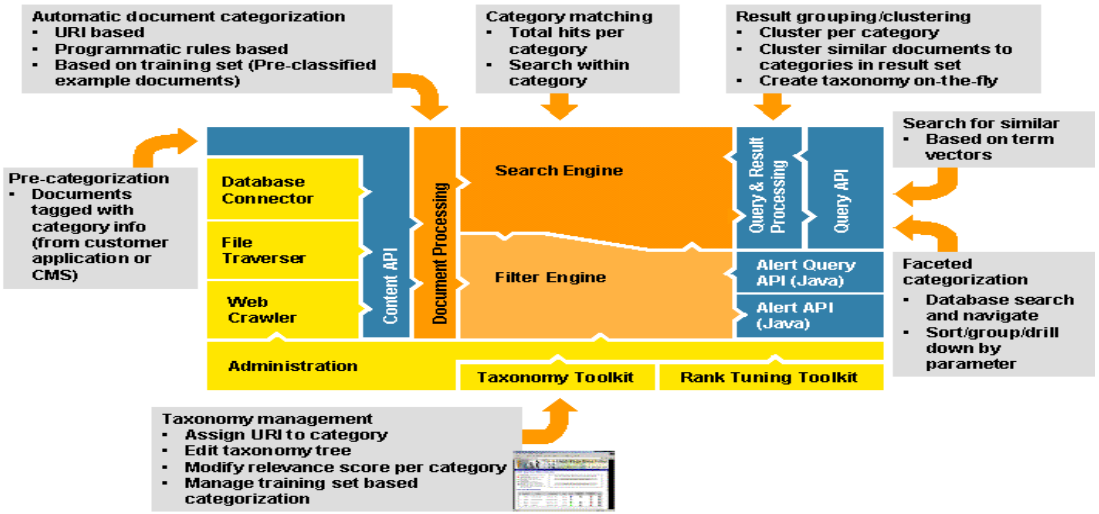

Figure 11-1 FAST InStream Categorization and Taxonomy Features

The features can either be activated independently or used in combination to create the desired user experience.

Categorizing Documents

FAST InStream includes support for several categorization techniques including:

Documents Tagged by Category

The customer application, such as a Content Management System, may tag documents with category information and submit to FAST InStream. Even if the categorization applies to only a sub-set of the documents, FAST InStream may assign non-categorized documents to the pre-defined categories on-the-fly.

Automatic Categorization

Several techniques provide automatic category tagging of documents prior to indexing. In FAST InStream these techniques are implemented within the modular document-processing framework. This includes:

- *URI based categorization*. A document may manually be assigned to one or more categories by its URI. This is accomplished by a configurable list of mappings from URI to category applied using the Taxonomy Toolkit.
- *Rule based categorization*. Categorization is based on programmatic rules specified using the Python programming language. This provides a very powerful tool for creating custom categorization rules. The rule definitions are defined per category using the Taxonomy Toolkit GUI. A set of examples on how to create rules is provided with the toolkit.

Category Matching

Categories may be used within the matching process of FAST InStream. This includes the following capabilities:

- *Search within category*. It is possible to define category filtering on a query in order to match only documents that are included in one or more unique categories, but also match documents within a certain sub-part of a taxonomy tree. This feature may be used for explicit category search as well as category drill-down queries in order to narrow down an initial query that may have returned too many results.
- *Number of hits per category*. A dedicated *Taxonomy Index* enables a query result that includes number of hits per category across the entire searchable index. A particular query returns a result set of the "n" highest ranked documents that match the query. "n" is a configurable parameter, for example, 10. Within the result set only a few categories may be represented. By using the taxonomy index, the total number of hits within all categories (as long as at least one document within the category match the query) can be returned, also those categories that are not represented in the result set. This can be a very efficient tool for category drill-down subsequent queries.

Note! The Taxonomy Index function may be used for taxonomies that are fairly stable. Because each document needs to be indexed with the taxonomy specification, a change of taxonomy may require full or partial re-index of the documents in the index. Whether or not this is acceptable depends on the nature of the customer's taxonomy (to what extent does it change over time), the size of the customer index (how easy can it be re-indexed) and to what extent is it possible to force a full re-index (possible to control in FAST InStream for FAST Crawler, File Traverser, but not Content API).

Categorizing Results

Result Clustering

Categorization is about mapping documents to categories, but also about how the category information is presented in the results. The Result clustering features enable a directory or taxonomy view within a result set.

Result clustering can be achieved at several levels:

Supervised Clustering: Clustering Results per Category

Result documents that have been categorized during document categorization prior to indexing can be clustered along the categories. This is called supervised clustering.

Similarity Clustering: Clustering Similar Documents to Existing Categories

If only a subset of the returned documents is categorized prior to indexing, the result clustering feature may also try to assign non-categorized documents to categories in the result set. This is called similarity clustering.

Similarity clustering is performed by use of *Similarity vectors*. The similarity vectors of a document can be used to determine relative similarities between documents. If a non-categorized document is similar to a document pre-assigned to a category, the clustering feature may cluster these documents to the existing category.

Unsupervised Clustering: Creating Taxonomy on the Fly

For documents that cannot be categorized on either supervised or similarity clustering, the clustering feature may try to define ad-hoc category grouping of the documents based on similarity vectors.

Note! It is important to keep in mind the distinction between the Result Clustering and the Category Matching class of features. Result clustering is applied to a configurable set of top ranked results, while Category matching is applied across the entire index. Which features to use depends on the customer requirements on configuration flexibility and result presentation.

How Clustering Works

The Clustering feature is applied as part of the query result processing and tries to detect groups of documents in the result set that have somehow similar content, where similarity is assessed on the basis of the document vectors (see *Document Similarity Vectors* on page 131). The groups may or may not be hierarchical, and are represented in one or more tree structures.

Clustering is applied on an extended result set. This means that the Search Engine first returns the "n" most relevant documents to a query, based on the static and dynamic rank algorithms. The number "n" is configurable, and depends on your application's needs and performance requirements. Typically, $100 < n < 1000$. The FAST InStream default value is 100.

The clustering may be supervised and unsupervised:

Two clustered trees are returned. One tree holds the supervised clustering results and one holds the unsupervised clustering results.

In brief, this is what happens when a search is performed:

1. The query is dispatched to the Search Engine, and the top "n" results are retrieved.
2. A document vector for each of the "n" documents is provided. In FAST InStream the document vectors are computed during document processing prior to indexing. Therefore, the document vector may be retrieved from the appropriate result field.

   The weights in the document vectors are adjusted to reflect the distribution of strings across the top "n" results. Weights are adjusted according to the principle of inverse document frequency.

   Example: We may want to give a boost to strings that are rare and particular to a small set of documents in the result set. Correspondingly, common strings that occur is all or many of the documents we may want to give relatively less weight since they do not discriminate well.
3. The classification will be performed as follows:
   a. Of the "n" retrieved documents, some may be tagged with information about which category or categories they belong to. A tree that spans these categories is created, and the tagged documents are placed in the appropriate nodes in this tree.

      Example: Assume that $n = 4$, that hit #1 is tagged to belong to sport-tennis and that hit #3 is tagged to belong to categories sport-tennis and business. Furthermore, assume that hits #2 and #4 are untagged, with the document hit #2 is similar to document hit #3. We then have the following tree:

```
root
   sport
        tennis
           1, 3
      business
           3, 2
```

A unique identifier is assigned to each node in the tree. The nodes in the tree from the classification step have identifiers that have the prefix "S".

b An untagged document is placed in a tree node if there is a document in that node that is similar to the untagged document. This is the reason why document hit #2 is placed on the same level of the tree as document hit #3.

The degree of similarity between two documents may range from zero to one. A similarity of one means that the documents are duplicates on the level of their document vectors. Note that this does not mean that the documents are true duplicates.

4 The following unsupervised clustering procedure is then performed:

a The subset of resulting documents that could not be assigned to any node in the preceding classification procedure is applied to the clustering .

b An algorithm is applied to the input documents to form clusters of documents. The clusters are non-overlapping, that is, a document is not member of more than one cluster. Large clusters may be clustered recursively. This process builds up a tree.

c A labeling algorithm is applied to each cluster or node in the tree. This algorithm examines the strings found in the vectors of the cluster's members, and uses these to devise a description of the cluster.

Dynamic Drill-Down

Overview

Dynamic drill-down is a powerful navigation and binning tool for structured data, providing multi-dimensional drill-down in structured data based on facets of the content (for example, database rows, product catalog descriptions).

An integrated part of this is a *Results-based binning* that provides implicit ranking of dimensions (search result fields) based on relevance scores. For each dimension (search result field), the relevance score is computed based on the actual drill-down capabilities of this dimension. The binning performs ad-hoc clustering of results into dynamic bins based on value distribution for this parameter in the results.

Dynamic drill-down is optimized to limit overhead on search environment for e-commerce, Yellow Pages, Supply Chain, CRM, etc. Relevant results can be found faster using a combination of searching and browsing by parametric value and range.

All or the most popular fields from a database or product catalog can be used, and it supports both textual (for example, product name) and numerical (for example, price) attributes.

Figure 11-2 provides an example of a dynamic drill-down search result page created using the FAST Software Development Kit (FAST SDK):

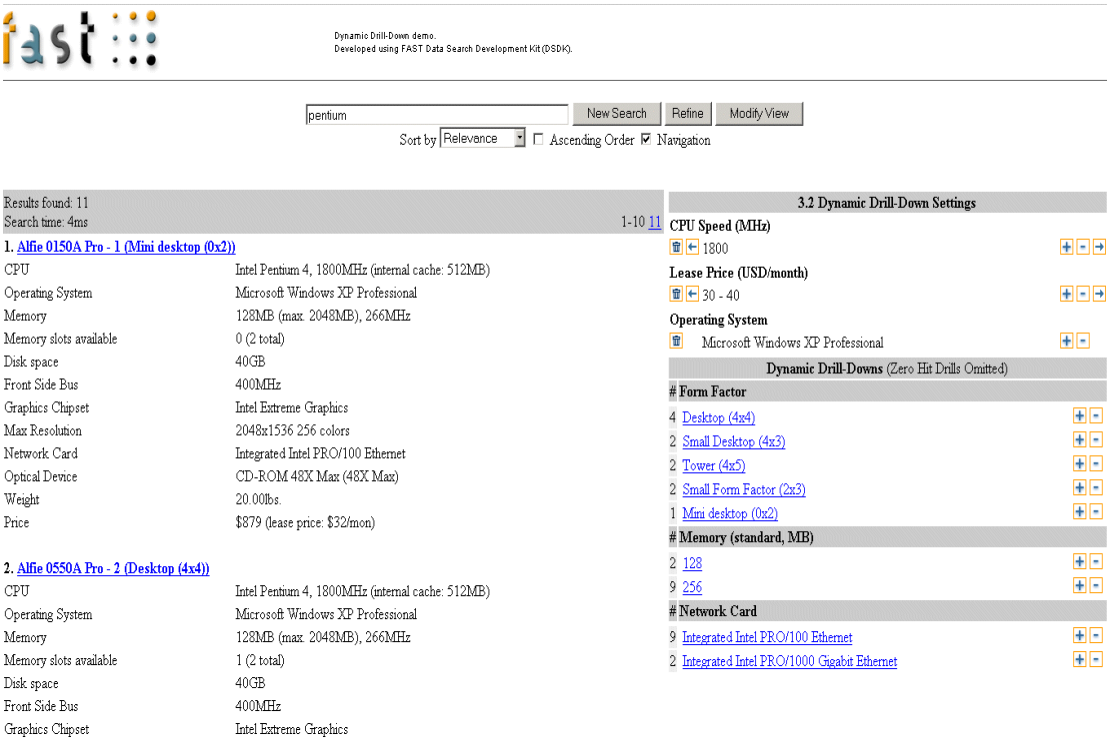

Figure 11-2 Dynamic Drill-down example

The left column displays the results in the usual ranked order, which is based on FAST InStream static and dynamic rank mechanisms, including the relevant parameter fields. The right column displays the drill-down and binning attributes. The feature is dynamic in the following ways:

- Sorted by drill-down relevance. Fields that are expected to provide the best drill-down alternatives are presented first. Typically, these are fields that provide a mean distribution of the values across the result set.
- The range for numeric values per bin is computed on-the-fly, trying to give a mean distribution of values to displayed range categories

For each field range it is provided drill-down links in order to:

- Drill-down within the displayed value range, for example, "Lease Price 30-40"
- Exclude the displayed value range
- Drill-down into all entries with price below or above the displayed value range

It is also possible to reverse an applied drill-down.

The drill-down parameters are sometimes denoted *faceted metadata,* and may apply for applications such as:

- Product databases may have attributes such as price, weight, color, country of origin and product type.
- Music store: songs have attributes such as artist, title, length, genre and date.
- Recipes: cuisine, main ingredients, cooking style and holiday.
- Travel site: articles have authors, dates, places, prices.
- Regulatory documents: product and part codes, machine types and expiration dates.
- Image collection: artist, date, style, type of image, major colors and theme.

An indexed field or attribute can be seen as a dimension in which the query can be refined. The search results are examined on the fly, and data is produced that can be rendered in the form of hyperlinks. This will help the user drill down to find what he or she is looking for by modifying the query.

This is especially relevant in the context of shopping search, where the searchable index is a database or product catalog. The fields indexed for each product may vary according to the type of the product. By supplying a navigational aid on top of the search engine that is adaptive to the search results for the user’s query, relevant results can be found faster. This can be viewed as a special kind of clustering. The results are clustered along one dimension at a time, and a relative priority among the dimensions is computed. The resulting hyperlinks effectively narrow the search along one of these dimensions.

How Dynamic Drill-Down Works

Within the FAST InStream index profile, Dynamic Drill-Down is configured to "monitor" a set of attributes <A>. In brief, this is what happens when a search is done:

1. The query is dispatched to the search engine, and the top "n" results are retrieved.

   The number "n" is configurable, and depends on the application's needs and performance requirements. Typically 100 < n < 1000. Larger "n" gives larger bins in the result presentation, but implies larger query overhead.

2. On the basis of the top "n" hits, the distribution of values is observed and histograms are created:

   Example 1: Assuming that an object can have, at most, one color and that 96 out of the top 100 results have a value for the attribute color, the histogram might look like this: {green/45, red/18, blue/22, yellow/11}.

   Example 2: For a price field the following may be returned:

   - Minimum, maximum and mean value for the price distribution in the results
   - Suitable interval boundaries that defines the width of the histogram's bins

3. The histograms are sorted according to a scoring formula. The scoring formula is configurable.

4. As an appendage to the standard result set, drill-down information is returned for those attributes that have at least one observed value among the top "n" hits. These attributes are returned sorted according to their above-mentioned score.

Find Similar

Overview

Drill-down queries may be defined based on similarity to selected documents in a previously returned result set. This is based on advanced similarity vectors, a set of term-importance pairs that are computed for each document prior to indexing. This method supports creation of similarity queries with very small query overhead. The following comparisons are supported:

- *Find similar*: Find documents similar to the selected document
- *Refine similar*: Refine the query by returning only documents similar to the selected document. Apply this to the documents of the original result set.
- *Exclude similar*: Refine query, return only documents different (not similar) from the selected document

These features use the same similarity vectors as the clustering features, and transform the vector into a query. Using advanced term weight capabilities ensures that different weight

is applied to different terms in the resulting query, enabling retrieval of highly relevant, similar documents.

Document Vectors and Find Similar

The document vectors for each document, sorted by decreasing weights, can be used to build three types of similarity searches for a document <d>, given an original query <Q>. Specific query parameters exists for this purpose, and can easily be applied from a search result set. Within FAST InStream these similarity search requests are transformed to a new unique query, using the following rewrite of the query (shown using a symbolic representation, not the exact query language):

Find similar:

Query = Q OR $<s_1,w_1>$ OR ... OR $<s_m,w_m>$

where the number of terms <m> is configurable, and where <s,w> represent the document's similarity vector.

This means that the similarity vectors are added to the query using an "or" operator. This means that the original query is included in the rewritten query, but the new query may match similar documents even if the original query is not met.

Refine similar:

Query = Q AND ($<s_1,w_1>$ OR ... OR $<s_m,w_m>$)

where <s,w> represent the document's similarity vector and the number of terms m is configurable.

This means that the query will match if the original query conditions *and* the similarity vector conditions are met. That is, refine original query with similar documents as the indicated.

Exclude similar:

Query = Q ANDNOT ($<s_1,w_1>$ OR ... OR $<s_m,w_m>$)

where <s,w> represent the document's similarity vector and the number of terms m is configurable.

This means that the query will match if the original query conditions are met but not the similarity conditions.

Note! The similarity vector weight is also included in the refined query. This implies that the Search Engine can take into consideration the importance of each vector in the rank analysis. Match of terms with lower weight will have less contribution to the overall rank of the found document.

Two ranging mechanisms can be requested for the refined results:

- By rank: The result set is sorted by the rank computed within the Search Engine
- By similarity. An extended result set (by default 100) is first retrieved from the Search Engine. The selection of this extended result set is based on the rank computed within the Search Engine. The Find Similar function will then re-sort these 100 entries by similarity to the original document, and return the most similar documents first in the final result set (for example, 10 first hits).

Taxonomy Management

The FAST Taxonomy Toolkit is a workbench that allows you to configure and maintain taxonomies and the mapping of categories. The Taxonomy Toolkit:

- provides a graphical user interface that presents taxonomies as trees.
- allows you to create multi-level trees.
- allows you to move nodes within a tree.
- allows you to delete, or add nodes to or from trees.
- allows you to add or delete node members, that means documents.
- allows you to define rank boost rules for given categories.
- allows you to define programmatic rules for automatic classification.

The Taxonomy Toolkit is implemented as a standalone tool that works on XML files containing 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.

An existing taxonomy populated with documents can be accessed and edited directly as an XML file or visualized in a graphical user interface as a tree, where nodes can be expanded or collapsed. By clicking on a node, a set of operations applicable to that node becomes available.

The Taxonomy Toolkit enables creation and editing of taxonomy structures and mapping documents to categories. This includes:

- Assign URI to category

  A given document URI is manually assigned to one or more categories.

- Edit taxonomy tree

  Add leaves, edit sub-structures, etc. The edited taxonomy tree will then be basis for the taxonomy mapping features and the taxonomy index.

- Modify relevance score per category.

  The Taxonomy Toolkit enables manipulation of relevance score (rank) related to categorization. This enables soft selection of specific category content. The modification will apply for all documents within a given category. The relevance score modification may be applied for any query or specified queries only.

- Define programmatic categorization rules

  Define rules on a per category basis. If such a rule evaluates to TRUE for a document, the document is assigned to the corresponding category.

Further Reading

The following parts of the FAST InStream documentation provide related information:

- Chapter 6 *Configuring the Taxonomy Toolkit* in the *Configuration Guide*

Chapter 12

FAST InStream in an Asian Language Environment

About this Chapter

This chapter gives you an overview about what to consider when implementing FAST InStream in an Asian Language environment.

The chapter includes:

- The Specifics of Asian Languages
- Asian Languages and Language and Encoding Detection
- Asian Languages and Tokenization
- Asian Languages and Lemmatization
- Asian Languages and Character Normalization
- Asian Languages and Text Sorting
- Asian Languages and Scope Fields

The Specifics of Asian Languages

Some languages require special processing in order for the content to be indexed and searched in a meaningful manner. Traditional and Simplified Chinese, Japanese and Korean are among these.

FAST InStream performs several operations that are unique or specialized for these languages. The following sections will address these operations and other issues that pertain to the Asian languages.

Note! Since Asian languages depend on specialized tokenizers, it is important that the language of the query is identified for Asian languages; otherwise the query will not be processed correctly. This is done by setting the appropriate variable in the search function. Since the Asian tokenizers also know how to tokenize Western language queries, it is normally a good practice to lock the query language to the Asian language in question and have this tokenizer process all queries, including the Western ones. If the query may be in more than one Asian language, for instance both Korean and Japanese, then a more sophisticated approach must be implemented in order to ensure that the queries are processed correctly.

Asian Languages and Language and Encoding Detection

Language and Encoding Detection for Non-Scope Fields

For non-scope fields, the *LanguageAndEncodingDetector* document processor (see Chapter 5 *Processing Documents*) performs both language and encoding detection.

In case the processing stage is unable to come up with reliable language and encoding identification, it will fall back to values given by the stage's "fallback_language" and "fallback_encoding" input parameters. By default, these parameters are set to "unknown" and "ISO-8859-1" (Latin-1), which in case of the encoding is a good guess for most Western deployments. However, for Asian documents this will be the wrong guess in almost all cases. Thus for a FAST InStream deployment that is to operate primarily on Asian data, it is recommended to set the fallback encoding to the most likely one for that particular environment. In Japan for instance, this would often be shift-jis.

Language and Encoding Detection for Scope Fields

For scope fields, language detection is provided by the *LanguageAndEncodingDetector* document processor, and encoding detection is provided by the *XMLParser* document processor.

Language Code Mappings for Chinese

Language detection in FAST InStream distinguishes between traditional Chinese, represented by the language code `zh-traditional`, and simplified Chinese, represented by the language code `zh-simplified`. However, a search specifying only `zh` as the language filter will result in both traditional Chinese and simplified Chinese fields being retrieved.

Furthermore, the language codes `zh-tw` and `zh-cn` will be mapped to `zh-traditional` and `zh-simplified`, respectively.

Asian Languages and Tokenization

Overview

Tokenization is the operation of splitting a stream of text into individual words (tokens) that can then be indexed. For most languages, such as English, this is a rather trivial operation. Characters such as spaces, tabs, periods, commas, dash, question marks, and quotes are examples of word delimiters.

For some languages however, the operation of identifying what constitutes a word can be quite complicated. Some Asian languages, such as Chinese, Japanese, and to some extent Korean, do not use space as a word separator. These languages require specialized and more sophisticated tokenizers that use more complicated rule sets as well as dictionaries.

FAST InStream comes with one default and four specialized tokenizers. The specialized ones are for the following Asian languages: Traditional and Simplified Chinese, Japanese and Korean.

For non-scope fields, tokenization of incoming Asian documents is provided by the *Tokenizer* document processor. For Asian documents, the *Tokenizer* document processor also provides lemmatization. Refer to section *Asian Languages and Lemmatization* on page 151 for further details on lemmatization.

For scope fields, tokenization is provided by the *ScopeTokenizer* document processor.

Customizing the Tokenizers

Since the specialized tokenizers rely on rules and dictionaries, there will always be some words that do not get tokenized in the desired way. This is particularly true for names, new words, and narrow industry terms. To enable the specialized tokenizers to identify such words according to client requirements, each specialized tokenizer has a custom dictionary to which the client can add terms of their own.

The *$FASTSEARCH/etc/tokenizer/userdicts* (UNIX) or *%FASTSEARCH%\etc\tokenizer\userdicts* (Windows) directory contains four empty user dictionary files with instructions on how to add new words to the tokenizer dictionaries. The Korean one is the exception. That file refers to yet another file in which the words are to be added.

Non-Linguistic Tokenization

It is also possible to configure FAST InStream to have indexes that enable search for exact strings independent of word boundaries. For these indexes, the specialized linguistic tokenizers are not used. With this feature enabled, the query "is is is" can get a hit on a document containing the text "This is isolating".

This feature represents an alternative way of indexing Asian text in those cases where finding an exact sequence of characters reliably is more important than whether those same characters in linguistic terms really constitute a word or not.

Asian Languages and Lemmatization

The lemmatization feature enables the end user to search for a document using other grammatical forms of a search term than what appear in an indexed document (see Chapter 10 *Advanced Linguistic Processing*, section *Lemmatization*).

Unlike for Western languages that have incoming documents lemmatized by the Lemmatizer document processor, Asian languages are tokenized and lemmatized in one and the same operation by the Tokenizer document processor. The difference in implementation is due to the cost of processing Asian text. Combining the two operations in one eliminates the need for a second round of costly text parsing.

Lemmatization is not applicable for Simplified and Traditional Chinese as neither form of Chinese has a grammar that causes words to be inflected. With other words, there is no need for a lemmatization feature for Chinese.

As for Western languages, Japanese and Korean are lemmatized by reducing both the words of the documents to be indexed and the query terms to their canonical forms.

Note! For the different methods of lemmatization, see Chapter 10 *Advanced Linguistic Processing*, section *Lemmatization*

Just like for tokenization, it is required that you set the language of the query string to be submitted correctly so that lemmatization for Japanese and Korean works properly.

Asian Languages and Character Normalization

The following Japanese related character normalizations are supported in FAST InStream.

- Full to half width Latin (always enabled)
- Half to full width Katakana (always enabled)
- Katakana to Hiragana (by default disabled)
- Kanji to Latin Numbers (always enabled)

Note! Normalizing Kanji to Latin numbers requires that lemmatization is enabled, as normalized numbers are stored as part of the lemma index.

- Small to regular size Kana (by default disabled)
- Old to modern Kanji, provided by the document processor *CharacterNormalizer*

Note! Katakana to Hiragana and small to regular size Kana normalization are configurable in the file *$FASTSEARCH/etc/NodeConf.xml* (UNIX) or *%FASTSEARCH%\etc\NodeConf.xml* (Windows).
This file contains a global options section where you can enable Katakana to Hiragana or small to regular size Kana normalization by setting the appropriate environment variable to the value 1.

Asian Languages and Text Sorting

A default installation of FAST InStream supports full-string furigana sort (Kana characters) for Japanese only. In order to be able to sort strings written with Hanzi, Kanji, Hanja or Hangul, you need to set "sortValueWidth" in the sorting configuration file *$FASTSEARCH/etc/alphasort/AlphaSortMasterFile.xml* (Unix) *%FASTSEARCH%\etc\alphasort\AlphaSortMasterFile.xml* (Windows) to 16 bit and then make references to files containing the character sets.

For general information about text sorting, refer to Chapter 7 *Concepts of Relevancy*, section *Sorting Results*.

FAST InStream comes with the following configuration files ready to be included in the master file:

- Simplified Hanzi sorted by pronunciation (Pinyin)

- Simplified Hanzi sorted by radicals and then stroke count
- Traditional Hanzi sorted by pronunciation (Pinyin)
- Traditional Hanzi sorted by radicals and then stroke count
- Traditional Hanzi sorted by stroke count
- Kanji sorted by radicals and then stroke count
- Hangul sorted according to UNICODE order

Note! The configuration file for sorting Kanji by radicals and stroke counts is derived from a file which is copyrighted by the Electronic Dictionary Research and Development Group, Monash University, Australia, and is used in accordance with the license stated at:
http://www.csse.monash.edu.au/groups/edrdg/licence.html.
The license conditions stated do not affect the portion of the original file that is extracted by FAST as long as the configuration file is used as intended by FAST.

Asian Languages and Scope Fields

Some of the advanced linguistics features for Asian languages require special considerations or have some limitations. These features are described below.

Non-Linguistic Tokenization in Scope Fields

Non-linguistic tokenization is not supported for scope fields.

Character Normalization in Scope Fields

FAST InStream fully supports the Japanese related character normalizations as described in section *Character Normalization in Scope Fields* on page 153 for scope fields, with the exception of Katakana to Hiragana normalization, which is not supported for scope fields.

Text Sorting

For scope fields containing any of the three Asian languages described here, text sorting is not supported.

Chapter 13

Operation and System Administration

About this Chapter

This chapter introduces you to the basic concepts of operating and administrating a FAST InStream installation.

Note! While this chapter gives you a conceptual view of operation and administration within FAST InStream, the *Operations Guide* and the *Deployment Guide* give you detailed procedural information about individual operational and administrative tasks.

The chapter includes:

- The FAST InStream Administrator Interface
- Logging
- Licensing
- Important Configuration and Log Files
- Sizing and Fault-Tolerance
- Security

The FAST InStream Administrator Interface

Overview

FAST InStream is administrated through the Administrator Interface. This is a graphical user interface that is accessible through a common Web browser.

Main Views

The FAST InStream Administrator Interface has eight main selections:

- Collection Overview
- Document Processing
- Search View
- System Management
- System Overview
- Logs
- Data Sources
- Matching Engines

Collection Overview

The **Collection Overview** selection allows you to monitor, create, configure, and delete collections running on your FAST InStream implementation.

For details on

- the concept of collections, refer to section *Collections* on page 19.
- procedural information about the tasks you can perform within the **Collection Overview** selection, refer to Chapter 1 *Basic Setup* in the *Configuration Guide*.

Document Processing

The **Document Processing** selection allows you to configure the document processing pipelines you want to use to process a collection. It displays statistics, such as host, port, status, stages, or pipelines for each stage of a pipeline. You can view, create, add, edit, or remove stages of the pipeline through this selection.

For details on

- the concepts related to document processing, refer to Chapter 5 *Processing Documents*.

- procedural information about the tasks you can perform within the **Document Processing** selection, refer to Chapter 3 *Configuring the FAST Document Processing Engine* in the *Configuration Guide*.

Search View

The **Search View** selection allows you to view the default search front-end provided with FAST InStream. This search front-end lets you search the documents of your implementation for testing purposes.

Note! You will have to create your own search front-end to serve your end-users' needs.

For detailed information about how to create your own search front-end, refer to the *Query Integration Guide*.

System Management

The **System Management** selection allows you to view status information for any node controller that is configured in the system. Information includes the node name, date and time of creation, general system information such as the name of the home directory and memory currently being used on the disk, and a list of all installed 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 FAST Crawler. You can also stop an entire node from this page.

For procedural information about the tasks you can perform within the **System Management** selection, refer to the *Operations Guide*.

System Overview

The **System Overview** selection gives you a total overview of your FAST InStream installation 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

The **Logs** selection allows you to view all system generated log files by file 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 page.

For details on

- logging in FAST InStream, refer to section *Logging* on page 159.
- information about individual log and error messages, refer to the *Operations Guide*

Data Sources

The **Data Sources** selection provides a list of available data sources and allows you to view the collections a data source is associated with.

For details on

- individual data source modules, refer to Chapter 8 *Processing Queries and Results*.
- how to configure individual data source modules, refer to Chapter 1 *Basic Setup* in the *Configuration Guide*.

Matching Engines

The **Matching Engines** selection allows you to view hostname, port number and type for each Search Engine in your FAST InStream installation, and to add a new Search Engine.

For details on

- the FAST Search Engine, refer to Chapter 8 *Processing Queries and Results*.
- For procedural information about how to configure the Search Engine, refer to Chapter 3 *Index Profile Management* in the *Configuration Guide*.

Logging

Each of the FAST InStream modules provides different log messages. Each log message is of a certain type. Table 13-1 explains these log message types:

Table 13-1 FAST InStream - Types of Log Messages

Type	Explanation
DEBUG	Points to debugging issues. This message type is not shown by default, but can be very useful when testing the external integration to one of the FAST InStream modules, since it enables debug output to be logged.
ERROR or WARNING	Points out that something has gone wrong, but with the entire module still operating properly. Typically, this type of message refers to a command that failed.
FATAL or CRITICAL	Points out that some critical failure has occurred, which prevents the FAST InStream system from operating properly and requires system exit.
INFO	Provides general information about the module or process it is referring to. Typically, this message contains statistics or progress numbers.
STATUS	Provides non-statistical status information and gives you an idea about what is going on within the module or process it is referring to.
VERBOSE	Gives a detailed non-statistical status report about the module or process it is referring to.

Licensing

This section gives a brief introduction to license management in FAST InStream.

Overview

FAST InStream is a system of individually licensed capabilities. These capabilities are either features, modules, or data amount capacities. Some of these capabilities are included in the standard delivery of FAST InStream, while others like advanced linguistics, or the Business Manager's Control Panel (BMCP) are additional modules for which you can purchase separate licenses and include them in your FAST InStream solution.

Based on the agreement with the customer, FAST generates a license file, which - together with the FAST InStream license management system - ensures that the purchased capabilities are enabled. The license management system is based on FLEXlm from Macrovision Corporate.

FAST InStream is provided to you with the understanding that only those components that have been purchased will be used.

Note! Time limited evaluation licenses for FAST InStream are available upon request.

The License Management System

FAST InStream licenses are floating licenses. This means that the capabilities can be shared between several FAST InStream processes running on different machines. The license management system keeps track of both the type and the number of capabilities that are checked out, and registers which processes are using them.

The license management system consists of several sub-components as illustrated in Figure 13-1:

Figure 13-1 License Management System Architecture

The license manager daemon, named *lmgrd*, handles the initial contact with the client application programs. It also starts and manages the FASTSRCH daemon. It must be started before any other FAST InStream module. This is handled automatically by FAST InStream upon system start.

Note! To ensure that FAST InStream can be operative even in case your license server fails, FAST InStream can keep running during four days without the license server being up and running. It is not possible to restart FAST InStream, though.

The FAST vendor daemon, named *FASTSRCH*, grants licenses based on requests from the FAST InStream processes, keeps track of the number of licenses that are checked out, and registers the processes that are using them.

The license file *fastsearch.lic* defines the features and maximum number of capacities that are enabled in your FAST InStream installation. The license file also specifies the machine on which the license manager is running, and the path to the FASTSRCH vendor daemon.

Your FAST InStream application checks out licenses, and checks them in again after use to free the capability for use by other processes.

The processes listed above can run on one machine or on different machines, as defined by the customer during the installation process.

Important Configuration and Log Files

When contacting FAST Support, provide the following configuration and log files:

- all log messages, located in the *$FASTSEARCH/var/log/* (UNIX) or *%FASTSEARCH%\var\log* (Windows) directory with $FASTSEARCH and %FASTSEARCH% being environment variables set to the directory where FAST InStream is installed.
- all configuration files located in *$FASTSEARCH/etc* (UNIX) or *%FASTSEARCH%\etc* (Windows) with $FASTSEARCH and %FASTSEARCH% being environment variables set to the directory where FAST InStream is installed.

Sizing and Fault-Tolerance

FAST InStream is infinitely linearly scalable in three dimensions: volume of data, number of users, and freshness of data.

Fault-tolerance can be achieved both with respect to the query stream and the document input. The Search Engine fault-tolerance ensures that index consistency is maintained after a Search Engine failure.

For detailed deployment considerations about scaling your FAST InStream installation, refer to the *Deployment Guide*.

Security

FAST InStream can easily be deployed behind a firewall. The firewall must be configured in a way that makes a number of FAST InStream interfaces available from outside the firewall depending on the requirements to the system.

For more details about securing your FAST InStream installation, refer to the *Deployment Guide*.

Further Reading

The following parts of the FAST InStream documentation provide related information:

- the *Configuration Guide*
- the *Operations Guide*

Appendix A

Supported Document Formats

The Document Processing Engine can process the following document formats:

Table A-1 Supported Document Formats

Document Format	Version
Access	Versions through 2.0
Adobe Acrobat (PDF)	Versions 2.1, 3.0 – 6.0, Japanese
Adobe FrameMaker (MIF)	Version 6.0
Adobe FrameMaker graphics (FMV)	Vector/raster through 5.0
Adobe Illustrator	Versions through 7.0, 9.0
Adobe Photoshop (PSD)	Version 4.0
Ami Draw (SDW)	Ami Draw
ANSI Text	7 & 8 bit
ASCII Text	7 & 8 bit
AutoCAD Drawing	Versions 2.5 - 2.6, 9.0 - 14.0, 2000i and 2002
AutoCAD Interchange and Native Drawing formats	DXF and DWG
AutoShade Rendering (RND)	Version 2.0
Binary Group 3 Fax	All versions
Bitmap (BMP, RLE, ICO, CUR, OS/2 DIB & WARP)	All versions
CALS Raster (GP4)	Type I and Type II
Computer Graphics Metafile (CGM)	ANSI, CALS NIST version 3.0
Corel Clipart format (CMX)	Versions 5 through 6
Corel Draw (CDR with TIFF header)	Versions 2.x – 9.x
Corel Draw (CDR)	Versions 3.x – 8.x

Table A-1 Supported Document Formats

Document Format	Version
Corel/Novell Presentations	Versions through 11.0
DataEase	Version 4.x
dBASE	Versions through 5.0
dBXL	Version 1.3
DEC WPS Plus (DX)	through 4.0
DEC WPS Plus (WPL)	through 4.1
DisplayWrite 2 & 3 (TXT)	All versions
DisplayWrite 4 & 5	through Release 2.0
EBCDIC	
Enable	3.0, 4.0 and 4.5
Enable	Versions 3.0, 4.0 and 4.5
Enable	Versions 3.0, 4.0 and 4.5
Encapsulated PostScript (EPS)	TIFF header only
Executable (EXE, DLL)	
Executable (Windows) NT	
First Choice	through 3.0
First Choice	Versions through 3.0
First Choice	Versions through 3.0
FoxBase	Version 2.1
Framework	3.0
Framework	Version 3.0
Framework	Version 3.0
Freelance (Windows)	Versions through Millennium 9.6
Freelance for OS/2	Versions through 2.0
GEM Paint (IMG)	No specific version
Graphics Environment Mgr (GEM)	Bitmap & vector
Graphics Interchange Format (GIF)	No specific version
GZIP	All versions
Hangul	Version 97
Harvard Graphics (Windows)	Windows versions
Harvard Graphics for DOS	Versions 2.x & 3.x
Hewlett Packard Graphics Language (HPGL)	Version 2

Table A-1 Supported Document Formats

Document Format	Version
HTML	through 3.0 (some limitations)
IBM FFT	All versions
IBM Graphics Data Format (GDF)	Version 1.0
IBM Picture Interchange Format (PIF)	Version 1.0
IBM Revisable Form Text	All versions
IBM Writing Assistant	1.01
Initial Graphics Exchange Spec (IGES)	Version 5.1
JFIF (JPEG not in TIFF format)	.All versions
JPEG (including EXIF)	All versions
JustSystems Ichitaro	Versions 5.0, 6.0, 8.0 – 12.0
JustWrite	Versions through 3.0
Kodak Flash Pix (FPX)	All versions
Kodak Photo CD (PCD)	Version 1.0
Legacy	Versions through 1.1
Lotus 1-2-3 (DOS & Windows)	Versions through 5.0
Lotus 1-2-3 (OS/2)	Versions through 2.0
Lotus 1-2-3 Charts (DOS & Windows)	Versions through 5.0
Lotus 1-2-3 for SmartSuite	Versions 97 – Millennium 9.6
Lotus AMI/AMI Professional	Versions through 3.1
Lotus Manuscript	Version 2.0
Lotus PIC	All versions
Lotus Snapshot	All versions
Lotus Symphony	Versions 1.0,1.1 and 2.0
Lotus Word Pro	Versions 96 through Millennium Edition 9.6, text only
LZA Self Extracting Compress	All versions
LZH Compress	All versions
Macintosh PICT1 & PICT2	Bitmap only
MacPaint (PNTG)	No specific version
MacWrite II	Version 1.1
MASS11	Versions through 8.0
Micrografx Designer (DRW)	Versions through 3.1
Micrografx Designer(DSF)	Windows 95, version 6.0

Table A-1 Supported Document Formats

Document Format	Version
Micrografx Draw (DRW)	Versions through 4.0
Microsoft Binder	Versions 7.0-97 (only on Windows)
Microsoft Excel (Mac)	Versions 3.0 – 4.0, 98, 2001
Microsoft Excel (Windows)	.Versions 2.2 through 2003
Microsoft Excel Charts	Versions 2.x - 7.0
Microsoft Multiplan	Version 4.0
Microsoft Outlook Folder (PST)	Versions 97, 98, 2002, and 2002
Microsoft Outlook Message (MSG)	All versions
Microsoft PowerPoint (Mac)	Versions 4.0 through 2001
Microsoft PowerPoint (Windows)	Versions 3.0 through 2003
Microsoft Project	Versions 98 - 2002 (text only)
Microsoft Rich Text Format (RTF)	All versions
Microsoft Word	Versions through 6.0
Microsoft Word	Versions through 2003
Microsoft Word (Mac)	Versions 3.0 – 4.0, 98, 2001
Microsoft WordPad	All versions
Microsoft Works	Versions through 2.0
Microsoft Works	Versions through 4.0
Microsoft Works (DOS)	Versions through 2.0
Microsoft Works (DOS)	Versions through 2.0
Microsoft Works (Mac)	Versions through 2.0
Microsoft Works (Mac)	Versions through 2.0
Microsoft Works (Mac)	Versions through 2.0
Microsoft Works (Windows)	.Versions through 4.0
Microsoft Works (Windows)	Versions through 4.0
Microsoft Write	Versions through 3.0
MIME Text Mail	
Mosaic Twin	Version 2.5
MultiMate	Versions through 4.0
Navy DIF	All versions
Nota Bene	Version 3.0
Novell Perfect Works	Version 2.0

Table A-1 Supported Document Formats

Document Format	Version
Novell Perfect Works	Version 2.0
Novell PerfectWorks (Draw)	Version 2.0
Novell WordPerfect	Versions through 6.1
Novell WordPerfect	Versions 1.02 through 3.0
Novell/Corel WordPerfect	Versions through 11.0
Office Writer	Versions 4.0 - 6.0
OS/2 PM Metafile (MET)	Version 3.0
Paint Shop Pro 6 (PSP)	Windows only, versions 5.0 – 6.0
Paradox (DOS)	Versions through 4.0
Paradox (Windows)	Versions through 1.0
PC Paintbrush (PCX and DCX)	All version
PC-File Letter	Versions through 5.0
PC-File+ Letter	Versions through 3.0
Personal R:BASE	Version 1.0
PFS: Professional Plan	Version 1.0
PFS:Write	Versions A, B and C
Portable Bitmap (PBM)	All versions
Portable Graymap (PGM)	No specific version
Portable Network Graphics (PNG)	Version 1.0
Portable Pixmap (PPM)	No specific version
Postscript (PS)	.Level II
Professional Write	Versions through 2.1
Professional Write Plus	Version 1.0
Progressive JPEG	.No specific version
Q & A	Versions through 2.0
Q&A	Version 2.0
Q&A Write	Version 3.0
Quattro Pro (DOS)	Versions through 5.0
Quattro Pro (Windows)	Versions through 11.0
R:BASE 5000	Versions through 3.1
R:BASE System V	Version 1.0
Reflex	Version 2.0

Table A-1 Supported Document Formats

Document Format	Version
Samna Word	Versions through Samna Word IV+
SmartWare II	Version 1.02
SmartWare II	Version 1.02
SmartWare II	Version 1.02
Sprint	Versions through 1.0
Star Office/Open Office Calc	Star Office Versions 5.2, 6.x, and 7.x Open Office version 1.1 (text only)
Star Office/Open Office Draw	Star Office 5.2, 6.x, and 7.x and OpenOffice version 1.1 (text only)
Star Office/Open Office Writer	Star Office Versions 5.2, 6.x, and 7.x Open Office version 1.1 (text only)
StarOffice / OpenOffice Impress	StarOffice 5.2, 6.x, and 7.x OpenOffice 1.1 (text only)
Sun Raster (SRS)	No specific version
SuperCalc 5	Version 4.0
Text Mail (MIME)	
TIFF	Versions through 6
TIFF CCITT Group 3 & 4	Versions through 6
Total Word	Version 1.2
Truevision TGA (TARGA)	Version 2
Unicode Text	All versions
UNIX Compress	
UNIX TAR	
UUEncode	
vCard	Version 2.1
Visio	Versions 5, 2000 and 2002
Visio (preview)	Version 4
Volkswriter 3 & 4	Versions through 1.0
VP Planner 3D	Version 1.0
Wang PC (IWP)	Versions through 2.6
WBMP	No specific version
Windows Enhanced Metafile (EMF)	No specific version

Table A-1 Supported Document Formats

Document Format	Version
Windows Metafile (WMF)	No specific version
WML	Version 5.2
WordMARC	Versions through Composer Plus
WordPerfect Graphics (WPG & WPG2)	Versions through 2.0, 7 and 10
WordStar	Versions through 7.0
WordStar	Version 1.0
WordStar 2000	Versions through 3.0
X-Windows Bitmap (XBM)	x10 compatible
X-Windows Dump (XWD)	x10 compatible
X-Windows Pixmap (XPM)	x10 compatible
XyWrite	Versions through III Plus
ZIP	PKWARE versions through 2.04g

Glossary

This glossary lists FAST InStream specific terms and their definitions.

antiphrasing Identifying word sequences in queries that are irrelevant for the search.

API Application Programming Interface.

approximate matching Matching a query term and a term within a document based on the edit distance between a query term and the document term. The edit distance is given by the number of basic character operations (add, delete, swap) required to transform the query term to a name.

Asian language tokenization Tokenization for Asian languages requires special treatment. As these languages do not allow to split up text into word entities by referring to white-space or other separators, Asian language text needs to be split up into tokens that can be treated as words during document processing and matching.

boosting Increasing the relevancy value of a document, that means the field weight within a field view.

categorization The process of assigning documents to specific categories. These categories may derive from a given hierarchy, a so called taxonomy, or be calculated on the fly based on similarities between documents.

clustering Arranging results in result groups according to characteristics that are external to the result set (supervised clustering) or inherent to the result set (unsupervised clustering).

collection A group of content types. Content types can be grouped by source and by the processing rules that are to be applied to this type of content.

content External data input to the FAST InStream system.

Content API An API that is used to push content to a FAST InStream implementation.

Content Distributor A submodule of the Document Processing Engine. It receives documents and distributes them to the appropriate document processing pipelines.

content entity	A single piece of content.
Crawler	A data source that retrieves external files from Web servers. By following links, the FAST Crawler crawls Web content hierarchies based on a single start URL. As opposed to: File Traverser. The File Traverser goes along directory structures, whereas the FAST Crawler crawls Web servers along URI structures.
data source	A module that retrieves content from external sources. Example: FAST Crawler, File Traverser.
document	A piece of content that is normalized with respect to the FAST InStream document structure. As opposed to: content.
Document Processing Engine	A module that provides document processing pipelines for analyzing documents.
document processing pipeline	A sequential set of document processing stages.
document processing stage	Instantiation of a document processor. Example: A pipeline consists of the document processing stages A - B - A - C. It contains 4 document processing stages, but represents only 3 document processors.
document processing stage	Part of a document processing pipeline that performs a particular processing of a document. A document processing stage may modify, remove or add information to a document, such as adding new meta information for linguistic processing, or extracting information about the language the document is written in.
document summary	Data that is returned to answer a query. A query is answered by a set of document summaries. There is one document summary per document in the result set.
element	Part of a document. A document is divided into elements in order to enable parametric search and to apply different ranking weights on different parts of a document.
entity extraction	A feature provided by the FAST Document Processing Engine. Documents sent through the NewsSearch document processing pipeline are processed to detect geographical, personal, or company names to enhance relevancy.
field	Part of an index profile. Fields specify those elements of a document that are to be searchable or presented in the result.

File Traverser	A data source module that traverses file server directories. As opposed to: FAST Crawler. The File Traverser works along directory structures, whereas the FAST Crawler crawls Web servers along URI structures.
freshness boosting	Ranking results along date or age.
full text sorting	Sorting on a configurable number of characters in the field on which you want to sort your results.
index profile	An XML file that defines the way documents are to be searchable. It contains fields to which the elements of a document are mapped.
lemmatization	Alternative grammatical forms of words are extracted and added to the document in a separate document section to enable searching for other grammatical forms than that given in the query string.
log message type	Type of log message like INFO, VERBOSE, etc.
multilevel sorting	Sorting by multiple fields. Both text and integer fields may be sorted upon. This can be used for improved navigation for end-user.
proper name recognition	Identifying word sequences in text that are defined as proper names or phrases in the appropriate dictionary.
Query and Result Engine	A module that processes and transforms queries and processes results.
Query API	An API that is used to submit search requests from an end-user or customer front-end application to the FAST InStream system.
ranking	Arranging result Documents according to their relevancy value.
result set	Set of document summaries that point to the resulting documents returned for a query.
result view	Defines alternative ways for a query front-end to view the index with relation to queries.
result-based binning	Provides implicit ranking of dimensions (search result fields) based on relevance scores.
Search Engine	Matching engine that enables indexing and search in indexed documents.
search engine cluster	Group of Search Engine instances that share one index profile.
search index	Searchable documents. After having been indexed, documents are searchable and form an index.

search query	Search request sent to the FAST InStream system.
sorting	Sorting results according to a fixed sorting value based on a specified field you wish to sort the results on.
spell-checking	Correcting or proposing corrections to common typing errors in the search query based on a comparison of the query's terms and a dictionary.
sub-string search	Searching for parts of a string as with a wildcard search ("*term*"). A word or token (for Asian language documents) is split up in smaller entities, so called sub-strings, consisting of a defined number of signs.
supervised clustering	Clustering results in result groups by means of a predefined taxonomy whose categories are assigned to documents prior to indexing.
taxonomy	A defined hierarchy of categories.
Taxonomy Toolkit	A module that allows you to configure and maintain taxonomies and the mapping of categories.
tokenization	Splitting up text into word entities. This involves detection of white-space characters and other symbols that separate words from each other and are not relevant for the matching process.
unsupervised clustering	Clustering results in result groups based on the document similarity calculated through vectorization.
vectorization	The process of calculating a document vector for a given document. A document vector is the numerical representation of the unstructured textual content of a document.

Index

S

T

W