Search for: Search

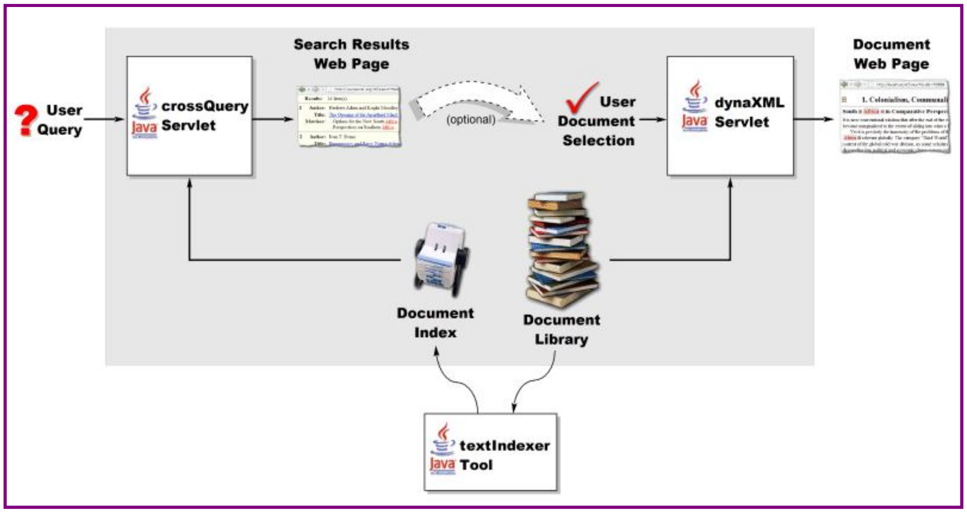
- Home
- About
- XTF Implementations
- XTF Community
- Tutorial
  - Quick Start
  - Fundamental Concepts
  - First Steps
  - The Essentials
  - The Exercises
    - Exercise 1: Add new content
    - Exercise 2: Change metadata
    - Exercise 3: Change logo/colors
    - Exercise 4: Results ranking
    - Exercise 5: Customize search
    - Exercise 6: Modify results
    - Exercise 7: Strucutral searching
    - Exercise 8: Hierarchical facets
    - Exercise 9: Change footnotes
- Documentation
  - Change Log
  - Deployment Guide
  - Programming Guide
  - Tag Reference
  - Tips & Tricks
  - Under the Hood
  - Experimental Features
  - Undocumented Features
  - Resources
    - Rowan Brownlee’s Beginner’s Guide to XTF
    - XTF Stylesheet Hierarchy
- FAQ
- Download
- Support

Deployment Guide

Introduction

This guide describes the installation and configuration of the E**X**tensible **T**ext **F**ramework (XTF).

The ***XTF*** system consists of Java Servlets and tools that permit users to perform Web-based searching and retrieval of electronic documents. Its basic organization can be illustrated as follows:



In this diagram, the basic flow of information is left to right. Document retrieval begins with a Web-based user search query. The ***crossQuery*** servlet checks the query against an index of available documents, and produces a list of matching documents for display in a web browser. Selecting a document from the search results page invokes the ***dynaXML*** servlet, which retrieves and formats the actual document for display in a web browser. The ***textIndexer*** tool shown at the bottom is used to update the document search index whenever documents in the library are added, removed, or updated.

This guide describes the steps that must be taken to deploy a working XTF system. These steps include installing and configuring the ***crossQuery*** and ***dynaXML*** servlets, the ***textIndexer*** tool, and the run-time environment that they depend on.

Installation

This section describes how to get a sample XTF system up and running with minimal effort. The sample system uses a pre-selected document library and component configurations for the sake of illustration. It can be modified if desired to create a fully customized system.

Installation / Quick Start Running on Windows

Configuration

These sections provide detailed descriptions of how to install and configure each component in the system and the run-time environment that it depends on. The sections are arranged so that later sections build on earlier ones. Consequently, following the instructions in these sections in the order they appear will produce a fully customized XTF installation.

The XTF Base Directory (XTF_HOME) Configuring the textIndexer Tool Configuring the dynaXML Servlet Configuring the crossQuery Servlet

Usage Examples

Since the ***crossQuery*** and ***dynaXML*** servlets can be used together or separately, this final section illustrates examples of various stand-alone and combined configurations of the two servlets.

Usage Examples User Authentication (Optional)

Installation/Quick Start

This section provides simplified instructions for installing and configuring an XTF system. By using the pre-created document library, configuration files, and file system tree, this section yields a running XTF system with minimal effort. Be sure to go through the sub-sections in the order they appear, as each new section builds on the previous ones.

Install Run-Time Environment

The XTF servlets and tools depend upon a run-time environment consisting of the following:

- Sun Microsystems Java 2 Platform Standard Edition (J2SE) (via Sun Microsystems), version 1.5.0 or higher.
- A Java servlet container of your choice (e.g., Apache Tomcat , Caucho Resin, etc.)

If you need help installing these run-time tools, follow the steps outlined in the Run-Time Environment Installation/Configuration section.

Install the XTF Package

Installing the XTF Package usually consists of these simple steps:

1. Download the latest XTF package (**`xtf-2.x.war`**).
2. ***Very important:*** rename the file to **`xtf.war`**, so that the servlet container will unpack it to the `xtf` subdirectory (instead of `xtf-2.x`).
3. Place the `xtf.war` file (not `xtf-2.x.war`) inside the `webapps` directory of the servlet container (e.g. Tomcat or Resin).
4. Browse to a URL referencing one of XTF servlets (such as the one below). The servlet container should unpack the archive for you, creating an xtf sub-directory within webapps. In rare cases, you may need to restart the servlet container to get it to recognize the new application.

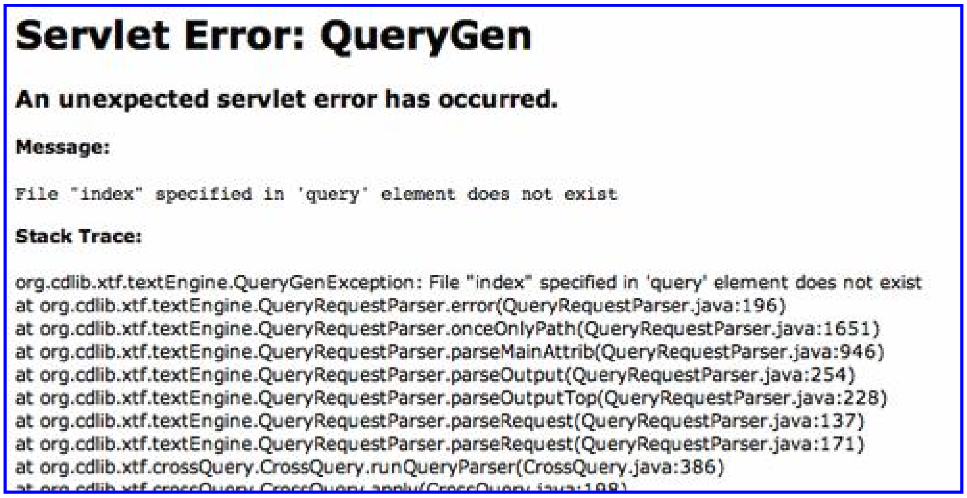
Verify that the Servlets are Correctly Installed

To verify that the servlets are running`,` bring up your web browser and enter the following URL:

```
http://localhost:8080/xtf/search
```

(Note: This URL assumes you’re running the browser on the same machine as XTF, and that the servlet container is configured to listen to port 8080. If not, substitute your server name and/or port for “localhost:8080″).

Accessing this URL should bring up an error page exactly like the following screen-shot (NOTE that the error ***is*** expected at this point):



Servlet Error: QueryGen

An unexpected servlet error has occurred.

Message:

```
File "index" specified in 'query' element does not exist
```

Stack Trace:

```
org.cdlib.xtf.textEngine.QueryGenException: File "index" specified in 'query' element does not exist
at org.cdlib.xtf.textEngine.QueryRequestParser.error(QueryRequestParser.java:196)
at org.cdlib.xtf.textEngine.QueryRequestParser.onceOnlyPath(QueryRequestParser.java:1651)
at org.cdlib.xtf.textEngine.QueryRequestParser.parseMainAttrib(QueryRequestParser.java:946)
at org.cdlib.xtf.textEngine.QueryRequestParser.parseOutput(QueryRequestParser.java:254)
at org.cdlib.xtf.textEngine.QueryRequestParser.parseOutputTop(QueryRequestParser.java:228)
at org.cdlib.xtf.textEngine.QueryRequestParser.parseRequest(QueryRequestParser.java:137)
at org.cdlib.xtf.textEngine.QueryRequestParser.parseRequest(QueryRequestParser.java:171)
at org.cdlib.xtf.crossQuery.CrossQuery.runQueryParser(CrossQuery.java:386)
```

Receiving this error page indicates that the ***crossQuery*** servlet is running; it's okay at this point that it can't access the index. Simply accessing this URL causes the servlet container to unpack the web archive file, creating an `xtf` sub-directory below the container's webapps directory, which in turn automatically installs the crossQuery and dynaXML servlets, the textIndexer tool, and a sample configuration that allows you to work with the default Sample Document Library. The reason we get an error is that we haven't installed and indexed the **Sample Document Library** yet.

If you get any other type of error, including **File Not Found** or **Page Not Found**, then XTF is not properly installed, and you should go back and double-check that you have correctly performed the steps described in the previous section.

Install the Sample Document Library

Once the crossQuery servlet is up and running, its time to install the sample document library. To do so, perform the following steps:

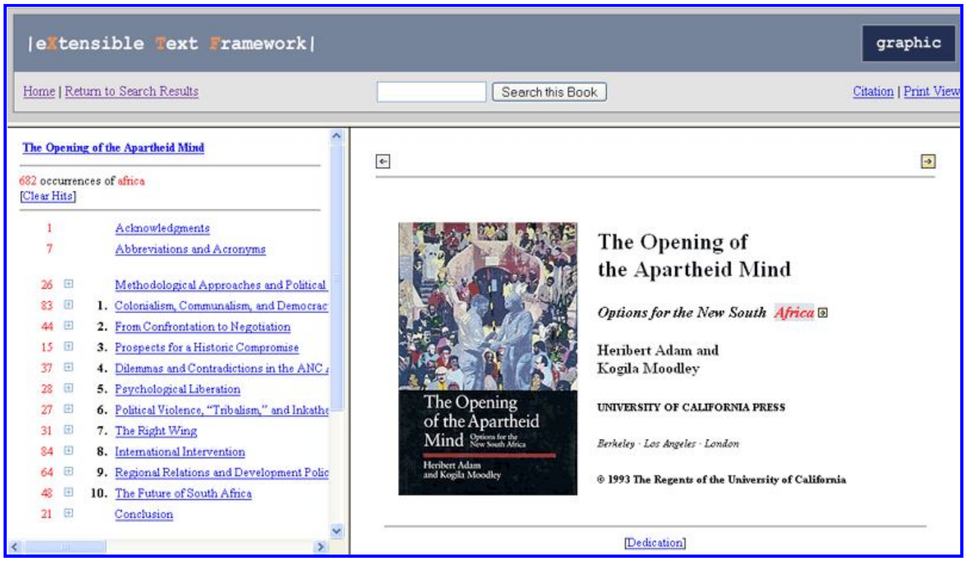
1. Download the **sample** package (`sample-data-2.x.zip`).
2. Using your favorite ZIP file utility, extract the sample document library's `data` directory and move it into into your XTF directory: *servlet-container-directory*`/webapps/xtf`.
3. You should now have the following directory present (among others): *servlet-container-directory*`/webapps/xtf/data/tei/ft958009mm`. This directory contains the document we will attempt to display.

Verify that Document Retrieval Works

Once the Sample Document Library has been installed, you can verify that basic document retrieval works. To do so, bring up your web browser and enter the following URL:

```
http://localhost:8080/xtf/view?docId=tei/ft958009mm/ft958009mm.xml
```

Doing so should bring up the cover for the book ***The Opening of the Apartheid Mind***, looking like this (without the query highlighting — we'll get to that soon):


|eXtensible Text Framework|
graphic
Home | Return to Search Results
Search this Book
Citation | Print View
The Opening of the Apartheid Mind
682 occurrences of africa
[Clear Hits]
1 Acknowledgments
7 Abbreviations and Acronyms
26 Methodological Approaches and Political
83 1. Colonialism, Communalism, and Democrac
44 2. From Confrontation to Negotiation
15 3. Prospects for a Historic Compromise
37 4. Dilemmas and Contradictions in the ANC
28 5. Psychological Liberation
27 6. Political Violence, "Tribalism," and Inkatha
31 7. The Right Wing
84 8. International Intervention
64 9. Regional Relations and Development Polic
48 10. The Future of South Africa
21 Conclusion
The Opening of the Apartheid Mind
Options for the New South Africa
Heribert Adam and Kogila Moodley
UNIVERSITY OF CALIFORNIA PRESS
Berkeley · Los Angeles · London
© 1993 The Regents of the University of California
[Dedication]

Index the Sample Document Library

Before you can verify that basic document searching works, you need to index the document library. To do this, follow the steps below. (*Note:* this procedure depends on a Unix-type shell and a Perl command interpreter. If you are running XTF on **Microsoft Windows**, please see the section Running on Windows.)

1. Switch to the `bin` sub-directory under the XTF installation directory:

```
$ cdservlet-container-directory/webapps/xtf/bin
```

1. Once in the `bin` sub-directory, make the textIndexer script runnable, then run it:

```
$ chmod +x textIndexer
```

```
$ ./textIndexer -index default
```

The ***textIndexer*** tool will now index the document library that you downloaded earlier. This should only take a couple of minutes, and you'll see output something like this:

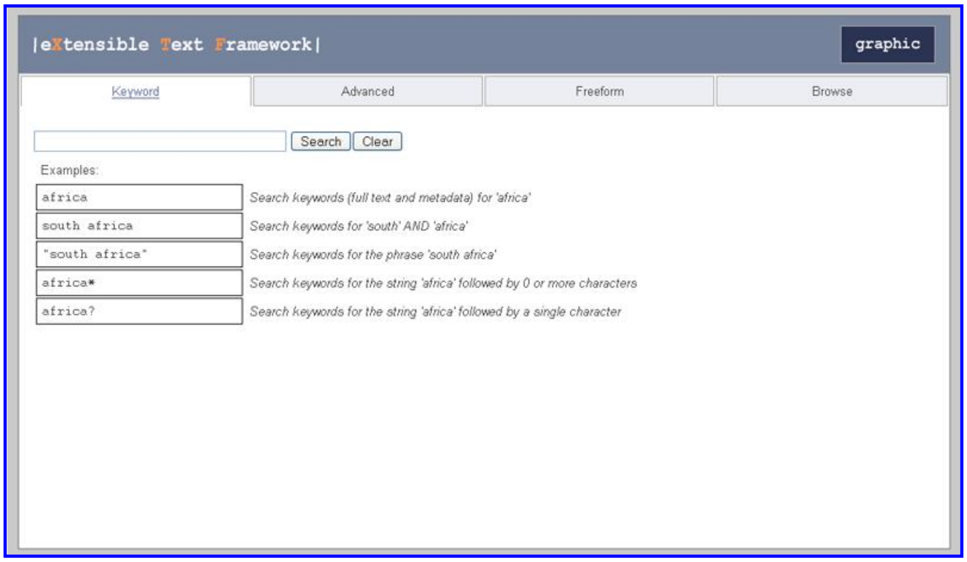
```
  TextIndexer v2.x

  Purging Incomplete Documents From Indexes:
  Done.

  Indexing New/Updated Documents:
    Index: "default"
      Scanning Data Directories....
      (5%)   Indexing [ead/bell/bell.xml] ... (1 stored key) ... Done.
      (6%)   Indexing [ead/clay/clay.xml] ... (1 stored key) ... Done.
      ...
    Done.
  Done.

  Optimizing Index:
    Index: [/Users/mhaye/Software/tomcat/webapps/xtf/index/] ... Done.
  Done.

  Updating Spellcheck Dictionary:
    ...
  Done.

Total time: 1 minute, 8 seconds.
Indexing complete.
```

Verify that Document Searching (crossQuery Servlet) Works

Once the document library has been indexed, you can verify that basic document searching using the ***crossQuery*** servlet works. To do this, bring up your web browser and enter the following URL:

```
http://localhost:8080/xtf/search
```

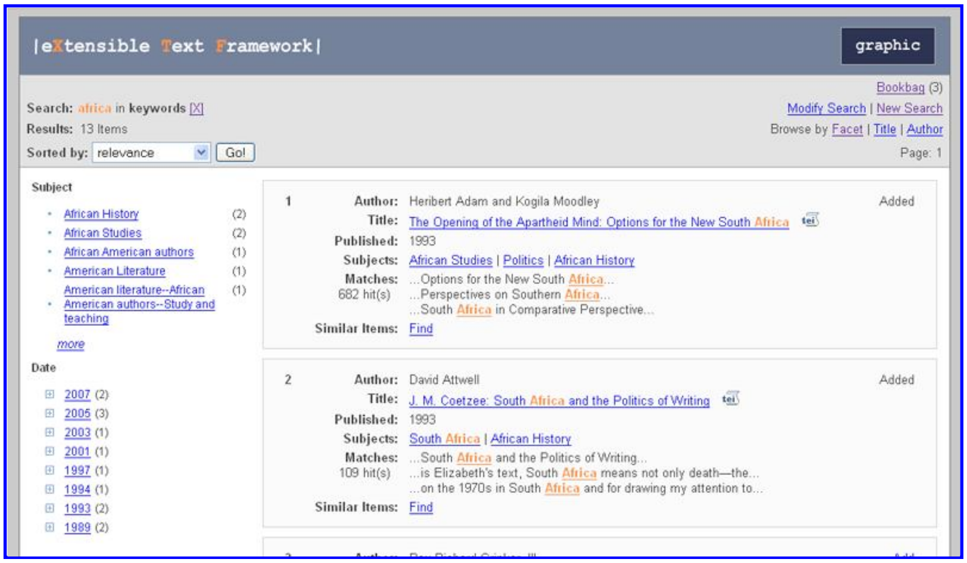
Doing so should bring up a page where you can enter information to search for, like this:



Examples:

africa	Search keywords (full text and metadata) for 'africa'
south africa	Search keywords for 'south' AND 'africa'
"south africa"	Search keywords for the phrase 'south africa'
africa*	Search keywords for the string 'africa' followed by 0 or more characters
africa?	Search keywords for the string 'africa' followed by a single character

Entering the word **africa** in the keyword field and pressing the **Search** button should yield a list of matching documents that looks like this:



|eXtensible Text Framework|
graphic
Bookbag (3)
Modify Search | New Search
Browse by Facet | Title | Author
Go!

Search: africa in keywords [X]
Results: 13 Items
Sorted by: relevance
Page: 1

Subject
- African History (2)
- African Studies (2)
- African American authors (1)
- American Literature (1)
- American literature--African American authors--Study and teaching (1)

more

Date
- 2007 (2)
- 2005 (3)
- 2003 (1)
- 2001 (1)
- 1997 (1)
- 1994 (1)
- 1993 (2)
- 1989 (2)

1
Author: Heribert Adam and Kogila Moodley
Title: The Opening of the Apartheid Mind: Options for the New South Africa
Published: 1993
Subjects: African Studies | Politics | African History
Matches: ...Options for the New South Africa...
682 hit(s) ...Perspectives on Southern Africa...
...South Africa in Comparative Perspective...
Similar Items: Find
Added

2
Author: David Attwell
Title: J. M. Coetzee: South Africa and the Politics of Writing
Published: 1993
Subjects: South Africa | African History
Matches: ...South Africa and the Politics of Writing...
109 hit(s) ...is Elizabeth's text, South Africa means not only death—the...
...on the 1970s in South Africa and for drawing my attention to...
Similar Items: Find
Added

Note that the first document listed is the book entitled ***The Opening of the Apartheid Mind***, which we selected manually in our initial test of the document retrieval (dynaXML) servlet. Clicking on the book title effectively passes a URL to the ***dynaXML*** servlet that looks like the one we entered manually before, and will result in the cover page for the book being shown. In contrast, clicking on the word **africa** in the search result blurbs will take you to the actual place in the book where the word “africa” appears.

Done

You're ready to start adding data to and/or customizing XTF. To add data in one of the formats already supported, simply put it into the `data` directory and re-run the ***textIndexer*** tool. To make the sample installation better suit your needs, continue with one of the following sections: Configuring the textIndexer Tool, Configuring the dynaXML Servlet, and Configuring the crossQuery Servlet.

Running on Windows

The base XTF distribution works well on a number of Unix variants, including Solaris, MacOS, and Linux. XTF also works under Microsoft Windows, but there are some special considerations for the ***textIndexer*** tool (the servlets crossQuery and dynaXML don't require anything special for Windows).

Instead of the usual Perl script to run the textIndexer, a batch file front-end has been provided: `bin/textIndexer.bat`. Using this batch file, there's no need for external tools such as Perl. However, you do need to set the `XTF_HOME` environment variable. Here are the steps:

1. From the Windows command prompt, set the `XTF_HOME` environment variable to the directory where the XTF package is installed. This environment variable is used by the `textIndexer.bat` driver to locate various XTF components. For example:

   ```
   C:...\> set XTF_HOME=C:\servlet-container-directory\webapps\xtf
   ```

   (Warning: Do not put quotes around the path when setting `XTF_HOME`, even if the path contains spaces.)
   (Note: To avoid having to repeatedly set the `XTF_HOME` environment variable, you may wish to add it to the list of system environment variables in Windows. On **Windows XP**, this can be done using the **System** control panel, under **Advanced** -> **Environment Variables**.)

1. Once `XTF_HOME` has been set, you can simply run textIndexer.bat as per the normal instructions. For example:

   ```
   C:\> cd %XTF_HOME%\bin

   C:...\xtf\bin> textIndexer -index default
   ```

Upgrading a Prior XTF Installation

If you have previously installed XTF and want to retain your existing stylesheets and configuration files, this section provides instructions for upgrading to a new release of XTF.

Upgrading usually consists of these steps:

- Shut down the servlet container.
- Make a backup of your existing XTF directory, in case something goes wrong during the upgrade.
- Download the latest XTF-Core package (`xtf-core-2.x.x.zip`). The core package contains the Java source code and libraries for XTF, but omits stylesheets and configuration files.
- Unzip the contents of the package into your XTF directory.
- *Optional:* Review changes to the default stylesheets and apply them to your own. You can do this by downloading `xtf-example-2.x.x.zip` and diffing the stylesheets and config files against the previous version or against your current files.
- Restart the servlet container and test to ensure that XTF is operating correctly.

Run-Time Environment Installation/Configuration

Successfully running the XTF tools and servlets depends on the prior installation and configuration of a minimum run-time environment. This environment consists of the **Java** run-time environment and a servlet container to run the ***dynaXML*** and ***crossQuery*** servlets.

The installation and configuration of each of these run-time environment components is discussed in turn in the following sub-sections. Once these items are installed and configured, the XTF system may be installed and run.

Java

Since the dynaXML and crossQuery servlets and textIndexer are all Java programs, you need to install the Sun Microsystems **Java 2 Platform Standard Edition (J2SE)** , version 1.5.0 or higher. The J2SE run-time can be downloaded directly from Sun via this page.

Since the J2SE package is a separate package maintained by Sun Microsystems, its installation and configuration is beyond the scope of this document. Please refer to Sun's own installation and configuration instructions for the J2SE.

Servlet Container

For the ***dynaXML*** and ***crossQuery*** servlets to be accessible on-line, they must be run under a **Servlet Container** application. At this time, the ***XTF system*** is known to work with the Apache Tomcat and Caucho Resin servlet containers. Please download one or the other of these servlet containers, and follow the accompanying instructions on how to install and configure the selected container.

Note: it has been reported that Tomcat version 4.1.27 is *not* compatible with XTF. If you are running this version or a version very close to it, it is recommended that you upgrade to Tomcat 5.x.

Included Libraries (no installation required)

The XTF distribution also includes two libraries, **Saxon** and **Lucene**, needed for proper operation. Generally no configuration will be required for these, unless other servlets are to be run in the same servlet container that require different versions of Saxon and/or Lucene. The code in XTF is customized to work very closely with both of these libraries, so it is not currently possible for XTF to use different versions than those included with the distribution.

The source text documents that are indexed and queried by the XTF system are **XML** based documents. Consequently, dynaXML, crossQuery, and textIndexer all use Michael Kay's **Saxon Library** for Java to parse the structure of the source documents and run stylesheets to transform them in various ways.

The textIndexer tool uses a library called **Lucene** to construct a search index of the XML source text documents in an XTF system. The crossQuery servlet then uses the resulting index to locate matching documents based on user generated queries. If queries are used in dynaXML, it also utilizes Lucene to perform them.

The XTF Base Directory (XTF_HOME)

The ***XTF*** base directory is critical to understanding where resources are located in your XTF installation. In general, the base directory is the top level of the XTF installation, and contains important sub-directories such as conf, style, and WEB-INF. Every component of XTF needs to locate this base directory to enable it to find configuration files, stylesheets, and data files.

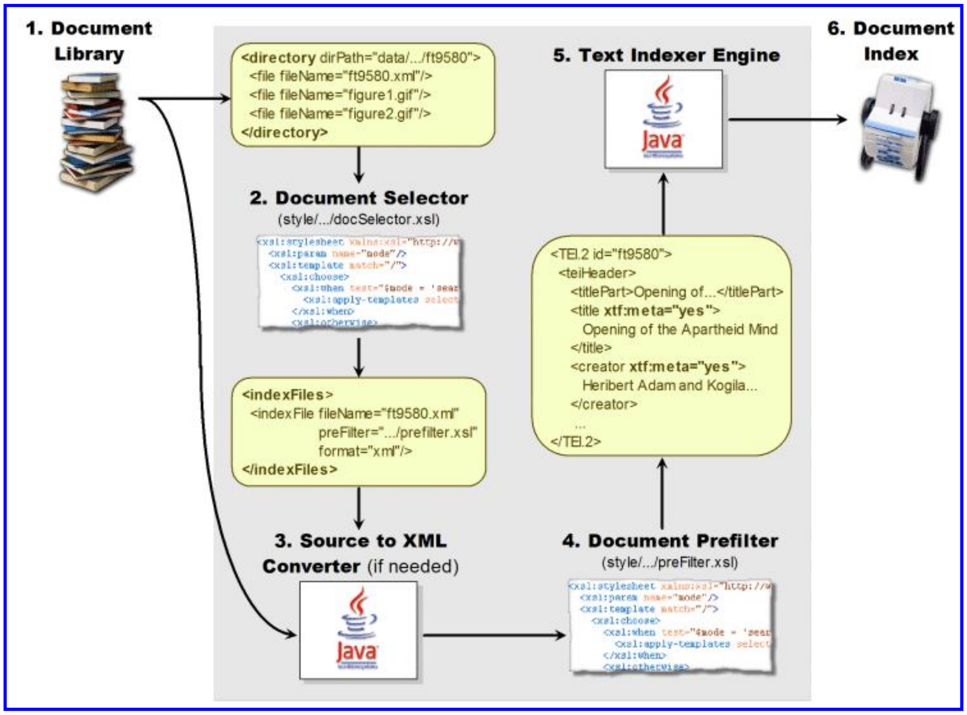
In the case of the servlets, the servlet container (typically **Tomcat** or **Resin**) provides the base directory, and it's usually: `servlet-directory/webapps/xtf/`. This can be overridden in the servlet configuration by specifying a `base-dir` configuration parameter; how to do this depends on the particular servlet container.

The command-line tools such as ***textIndexer*** cannot rely on the servlet container. The way this is handled depends on whether you're running XTF on Unix or Windows:

- On Unix systems, the `textIndexer` perl script in the XTF `bin` directory can detect where it's running from and set `XTF_HOME` automatically. However, if you are running multiple instances of XTF on one machine, you'll need to explicitly set the `XTF_HOME` variable to point to the correct instance.
- On Windows systems, you run `bin/textIndexer.bat`, which is not able to detect where it's running and set `XTF_HOME`. Thus, you need to set it explicitly. See the Running on Windows section for details.

Configuring the textIndexer Tool

The textIndexer tool is a command line tool that indexes the XML, PDF, and other documents in an XTF document library. The index it creates is used by the crossQuery servlet to search for documents based on user queries. The organization of the textIndexer tool can be illustrated as follows:



```
<directory dirPath="data/.../ft9580">
  <file fileName="ft9580.xml"/>
  <file fileName="figure1.gif"/>
  <file fileName="figure2.gif"/>
</directory>
```

```
<xsl:stylesheet xmlns:xsl="http://w
  <xsl:param name="mode"/>
  <xsl:template match="/">
    <xsl:choose>
      <xsl:when test="$mode = 'sear
        <xsl:apply-templates select
      </xsl:when>
      <xsl:otherwise>
```

```
<indexFiles>
  <indexFile fileName="ft9580.xml"
             preFilter=".../prefilter.xsl"
             format="xml"/>
</indexFiles>
```

```
<TEI.2 id="ft9580">
  <teiHeader>
    <titlePart>Opening of...</titlePart>
    <title xtf:meta="yes">
      Opening of the Apartheid Mind
    </title>
    <creator xtf:meta="yes">
      Heribert Adam and Kogila...
    </creator>
    ...
</TEI.2>
```

Running the textIndexer

Running the text indexer from the command line is accomplished by switching to the bin subdirectory just below the main XTF directory, and using the following command. Note: make sure you've set the `XTF_HOME` environment variable correctly first (see above).

```
textIndexer {-config config-file-path}
            {-incremental | -clean}
            {-optimize    | -nooptimize}
            {-updatespell | -noupdatespell}
            {-buildlazy   | -nobuildlazy}
            {-validate    | -novalidate}
            {-rotate      | -norotate}
            {-trace trace-level}
            {-dir subdirectory}
            {-dirList file-of-directories}
            {-force}
            -index index-name
```

Only the last argument, the `-index` *index-name* argument, is required. The remaining arguments (shown in braces) are optional arguments, and in the case of paired options (e.g. `-optimize` vs `-nooptimize`) the first is the default. Note that the order of the arguments is important, since each occurrence of `-index` *index-name* causes the specified index to be updated using the previously set argument values.

The arguments passed to the textIndexer have the following meanings:

`-config` *config-file-path*	This argument is an optional argument that identifies an XML configuration file to use when indexing. If this argument is omitted, the textIndexer will use the `textIndexer.conf` file found in the `XTF_HOME/conf directory`. If used, this argument must be the first argument passed to the text indexer. For a complete description of the contents of the configuration file, see the following section.
`-incremental` | `-clean`	This is an optional argument that can be set to either `-clean` or `-incremental`. This flag tells the text indexer whether the document indexes should be rebuilt from scratch (`-clean`) or should be updated incrementally (`-incremental`). If this argument is not specified, the default behavior is to incrementally update the index.
`-optimize` | `-nooptimize`	This is an optional argument that specifies whether or not the indexer should optimize the indexes after they are built. Optimization improves query speed, but can take a very long time to complete if an index is large. If this argument is not specified, the default behavior is to optimize indexes after they are updated.
`-updatespell` | `-noupdatespell`	This is an optional argument that if not specified defaults to `-updatespell`. The `-updatespell` argument tells the textIndexer to update the spelling correction dictionary once all of the documents have been indexed. Specifying `-noupdatespell` avoids updating the spelling dictionary; these updates will be made the next time the indexer is run with `-updatespell`. Typically this option is rarely used, as spelling updates are quite fast.
`-buildlazy` | `-nobuildlazy`	This is an optional argument that if not specified defaults to `-buildlazy`. The `-buildlazy` argument tells the textIndexer to pre-build lazy tree files. Lazy tree files speed the retrieval of documents by the dynaXML servlet, and also allow resulting matches within documents to be highlighted in context. Pre-building lazy tree files at index time ensures the fastest possible retrieval time for any document in the library, but at the expense of longer indexing times and additional hard-disk space usage. To avoid pre-building lazy tree files for all documents in a library, pass the `-nobuildlazy` flag instead. Doing so will cause the dynaXML servlet to build lazy tree files for documents only when the documents are requested. While this approach conserves hard-disk space, it comes at the expense of longer processing times when dynaXML retrieves a document for which it has not yet built lazy tree information. **WARNING**: If the `-nobuildlazy` flag is used, the **docSelctor.xsl** stylesheet used by the textIndexer and the **docReqParser.xsl** stylesheet used by the dynaXML servlet must both specify the same pre-filter, or else chaos may ensue.
`-validate` | `-novalidate`	This is an optional argument that if not specified defaults to `-validate`. The `-validate` argument tells the textIndexer to run index validation steps at the end of the run (just before rotation). This *only* applies if index validation has been enabled in textIndexer.conf (see the section on Index Validation & Rotation for details.) To avoid validation pass the `-novalidate` flag instead.
`-rotate` | `-norotate`	This is an optional argument that if not specified defaults to `-rotate`. The `-rotate` argument tells the textIndexer to rotate in a new index after it's been completed (and validated if validation is enabled) by making a copy and naming it "index-pending". This *only* applies if index rotation has been enabled in textIndexer.conf (see the section on Index Validation & Rotation for details.) To avoid rotation pass the `-norotate` flag instead.
`-trace` *trace-level*	This is an optional argument that sets the level of output displayed by the textIndexer. The possible output levels are defined as follows: `errors` Only error messages are displayed. `warnings` Both error and warning messages are displayed. `info` Error, warning, and informational messages are displayed. `debug` Low level debug output is displayed in addition to error, warning, and informational messages. If this argument is not specified, the textIndexer

	defaults to displaying informational (`info`) level messages.
`-dir` *subdirectory*	This argument is an optional, repeatable argument, and identifies one or more sub-directories of an index tree to re-index. The primary purpose of this argument is to allow only portions of an index to be updated to save time. When specified, the value of this argument should be path to the sub-directory relative to the base directory set for the index specified in the `-index` *index-name* argument that immediately follows. The `-dir` argument can be especially useful when combined with `-force` to re-index a certain directory regardless of whether its files have changed or not.
`-dirList` *file-of-directories*	This argument is an optional argument identify a file listing a set of directories to index. The primary purpose of this argument is to allow only portions of an index to be updated to save time. Paths in the file should be to the sub-directories relative to the base directory set for the index specified in the `-index` *index-name* argument that immediately follows. The `-dirlist` argument can be especially useful when combined with `-force` to re-index a set of directories regardless of whether their files have changed or not.
`-force`	This is an optional argument that, if specified, circumvents time/date checking in the textIndexer. Any directories specified by `-dir` or `-dirlist` (or all directories if neither are given) will be re-indexed regardless of whether their files have changed since the last index run.
`-index` *index-name*	This argument is a required argument that identifies the name of the index to be created/updated. The name must be one of the index names contained in the configuration file specified as the first argument. Note that this argument also starts the (re)indexing process for the given index using the values set by the arguments that precede it.

A simple example of command line parameters for a text indexer run might look like this:

```
$ textIndexer -clean -index default
```

This example uses the default config file, `textIndexer.conf`, located in the `XTF_HOME/conf` subdirectory. It assumes that the config file contains an entry for an index called **default**, and that the user wants the index to be rebuilt from scratch and then optimized (because of the presence of the `-clean` argument, and the absence of the `-nooptimize` argument.)

As mentioned above, the `-config` *config-file-path* argument (if used) must be specified first. After that, all the arguments may be used as a set one or more times to update multiple indexes. For example:

```
$ textIndexer -config conf/myIndexes.conf -clean -index default -incremental -nooptimize -inde
```

In this example, the `myTextIndexer.conf` file defines two indexes called **default** and **abstracts**, with the **default** index created from scratch and optimized at the end, while the **abstracts** index is updated incrementally and not optimized.

The textIndexer Configuration File

As noted above, first command line argument passed to the ***textIndexer*** tool is the name of a configuration file. This file contains the name of one or more indexes and their associated parameters. The format of the configuration files is as follows:

```
<?xml version="1.0" encoding="utf-8"?>

<textIndexer-config>

    <index name="NameOfIndex"/>

        <src path="LocationOfSourceDocuments" scan="all|pruned"/>
        <db path="LocationOfIndexDatabase"/>
        <chunk size="TextChunkSize" overlap="TextChunkOverlap"/>
        <stopwords path="LocationOfStopWordFile"/>
        <docselector path="LocationOfDocSelector"/>

        <pluralmap path="LocationOfPluralMapFile"/>

        <accentmap path="LocationOfAccentMapFile"/>
        <spellcheck createdict="YesOrNo"/>
    </index>
        .
    <index name="NameOfIndexN"/>
        .
    </index>

</textIndexer-config>
```

As the listing shows, the textIndexer configuration file is an XML file containing a configuration type tag (the tag `<textIndexer-config>`), followed by one or more index definition blocks. For each index block in the file, the following attributes are defined:

`<index name="`*NameOfIndex*`"/>`	This element specifies a name for an index definition block. The name may be any combination of digits and/or letters and the underscore (_) character. Punctuation and other symbols are not permitted, and neither is the use of the space character. Also, the index name may only be used for one index block in any given configuration (if it appears more than once, the first occurrence is used, and the remaining ones are ignored.) This index name is the name passed on the command line to the textIndexer to identify which indexes need to be processed.
`<src path="`*LocationOfSourceDocuments*`" scan="`*all|pruned*`"/>`	This element specifies the file-system path where the documents to be indexed are located. The path specified for an index must be a valid path for the operating system on which the tool is being run (e.g., Windows, Mac, Linux, etc.) If a relative path is used, it is considered to be relative to the base XTF installation directory (i.e., `XTF_HOME`.) If not specified, *scan* defaults to “pruned,” but the distribution configuration file is set to “all.”
`<db path="`*LocationOfIndexDatabase*`"/>`	This element specifies the file-system path where the database for the named index should be located. If the path does not exist or there are no databases files located there, the textIndexer will automatically create the necessary directories and database files. As with the source path, if a relative path is used, it is considered to be relative to the base XTF installation directory (i.e., `XTF_HOME`.)
`<chunk size="`*TextChunkSize*`" overlap="`*TextChunkOverlap*`"/>`	The textIndexer tool splits source documents into smaller chunks of text when adding the document to its index. Doing so makes proximity searches and the display of their resulting summary “blurbs” faster by limiting how much of the source document must be read into memory. See the discussion of chunk sizing below.
`<docselector path="`*LocationOfDocSelector*`"/>`	This tag identifies the location of an XSLT stylesheet that describes which source documents should be indexed, what pre-filter (if any) should be used on the selected documents, and what display formatter should be used to display the document when it is retrieved by the user. The path specified for the document selector must be a valid path and file name for the operating system on which the tool is being run (e.g., Windows, Mac, Linux, etc.) If a relative path is used, it is considered to be relative to the base XTF installation directory (i.e., `XTF_HOME`.) (Note: The document selector must be specified for an index. If it is not, an error is generated, and no document indexing will be performed.)

`<stopwords path="`*StopWordFileLocation*`"/>`
or
`<stopwords list="`*StopWordList*`"/>`
	The first variant of this attribute specifies the path to a file containing a list of words that the textIndexer should not add to the index. As with other paths, if a relative path is used, it is considered to be relative to the base XTF installation directory (i.e., `XTF_HOME`.) The second variant simply accepts the list of stop words as a string, with stop words separated from each other by spaces (i.e., “a an and the”.) See the discussion of stop words below for more information.

`<pluralmap path="`*LocationOfPluralMapFile*`"/>`	This tag identifies the location of a file containing a list of plural words and their corresponding singular forms that should be considered the same by the textIndexer. This file is primarily used to improve search results by finding plural forms of words when only the singular form has been specified (or vice versa.) See the discussion of plural mapping below.
`<accentmap path="`*LocationOfAccentMapFile*`"/>`	This tag identifies the location of a file containing a list of accented characters and their corresponding un-accented forms that should be considered the same by the textIndexer. This file is primarily used to map accented characters to un-accented ones so that searches can still be performed on words with characters not implemented on localized keyboards. See the discussion of accent mapping below.
`<spellcheck createDict="`*YesOrNo*`"/>`	This tag tells the indexer whether to create/update the spelling correction dictionary when the main indexing phase is complete. If no dictionary is created, then spelling correction will not be available in crossQuery. For more information on how to produce spelling suggestions, please see the Programming Guide. Note that dictionary creation does take some time, though it is generally a small proportion of the total indexing time.

Chunk sizing

The `<chunk>` element’s `size` attribute defines (as a number of words) how large the chunk size should be. Note that if the selected chunk size is too large, then time will be wasted reading too much text from disk. Inversely, if the chunk size is too small, too much time will be spent assembling the summary “blurb” from its component chunks. As a guideline, a chunk size of about 200 words was found through experimentation to give good overall performance with “blurbs” of 80 characters (about 15 words) in length. (Note: The chunk size specified for this attribute should be equal to or more than two words. If it is not, the textIndexer will force it to be two.)

The `overlap` attribute controls the amount of overlap between chunks. By overlapping adjacent chunks of text in the index, proximity searches can still be performed, even though the source document doesn’t appear in the index in one contiguous piece. This attribute defines (as a number of words) how large the chunk overlap should be. It should be mentioned that the selected chunk overlap effectively defines the maximum distance (in words) that can exist between two words in a document and still produce a search match. Consequently if you have a chunk overlap of five words, the maximum distance between two words that will result in a proximity match is five words. As a guideline, a chunk overlap of about 20 words for a chunk size of 200 words gives fairly good results. (Note: The chunk overlap specified for this attribute should be equal to or less than half the chunk size. If it is not, the textIndexer will force it to be half.)

Stop words

Eliminating stop-words from an index improves search speed for an index. This is because the search doesn’t need to sift through all the occurrences of the stop-words in the document library. Consequently, adding words like **a**, **an**, **the**, **and**, etc. to the stop-word list, which occur frequently in documents but are relatively uninteresting to search for, can speed up the search for more interesting words enormously.

The one caveat is that searches for any single stop-word by itself will yield **no matches**, so it is important to pick stop-words that people aren't usually interested in finding. Note however that due to an internal process called *bi-gramming*, stop words will still be found as part of larger phrases, like **of** in **Man of War**, or **the** in **The Terminator**.

Finally, if a stop-word file is used, the file itself may be stored in GZIP compressed format (and correspondingly labeled with a `.gz` extension) if desired.

Plural mapping

Since the XTF was designed to be language independent, the textIndexer relies on the Plural Map file to supply all plural information used during indexing, including simple plurals (i.e, words ending in s or es.) Consequently, if this file is not specified or if a plural mapping is not found in the file, the singular and plural forms of a word ***will not both be matched in a search.***

The contents of the file should consist of the plural form of a word followed by the 'pipe' character (|) and then by the singular form of the word. Like this:

```
cacti|cactus
cactuses|cactus
houses|house
mice|mouse
```

There should be only one plural to singular mapping per line in the file (although multiple plural mappings may exist for a singular form), and no white-space should be present within a definition line.

The path specified for the document selector must be a valid path and file name for the operating system on which the tool is being run (e.g., Windows, Mac, Linux, etc.) If a relative path is used, it is considered to be relative to the base XTF installation directory (i.e., `XTF_HOME`).

Finally, the file itself may be stored in GZIP compressed format (and correspondingly labeled with a `.gz` extension) if desired.

Accent mapping

Since the XTF was designed to be language independent, the textIndexer relies on the Accent Map file to supply all accent mapping information used during indexing. Consequently, if this file is not specified or if an accent mapping is not found in the file, the accented and un-accented forms of a letter ***will not both be matched in a search.***

The contents of the file should consist of the *16-bit Unicode value* for an accented form of a letter followed by the 'pipe' character (|) and then by the *16-bit Unicode value* for the un-accented form of the letter. Optionally, you can also add a comment at the end of the line, where a comment starts with a semicolon. Like so:

```
00C4|0041  ; Latin Capital Letter A With Diaeresis|Latin Capital Letter A
00C5|0041  ; Latin Capital Letter A With Ring Above|Latin Capital Letter A
00C6|0041 0045  ; Latin Capital Letter AE|Latin Capital Letters A E
00C7|0043  ; Latin Capital Letter C With Cedilla|Latin Capital Letter C
00C8|0045  ; Latin Capital Letter E With Grave|Latin Capital Letter E
```

There should be only one accented to unaccented mapping per line in the file (although multiple accent mappings may exist for an un-accented form), and no white-space should be present within a definition line. Note that single to multiple letter mappings are supported (such as mapping **Ã¶**> to **oe**).

The path specified for the document selector must be a valid path and file name for the operating system on which the tool is being run (e.g., Windows, Mac, Linux, etc.) If a relative path is used, it is considered to be relative to the base XTF installation directory (i.e., `XTF_HOME`.) Finally, the file itself may be stored in GZIP compressed format (and correspondingly labeled with a .gz extension) if desired.

Example configuration file

An example of a working ***textIndexer*** configuration file is the sample textIndexer.conf file found in the conf subdirectory immediately below the base XTF install directory. At the time of this writing it looked like this:

```
<?xml version="1.0" encoding="utf-8"?>

<textIndexer-config>

    <index name="default"/>
        <src path="./data"/>
        <db path="./index"/>
        <chunk size="200" overlap="20"/>
        <docselector path="./style/textIndexer/docSelector.xsl"/>

        <stopwords list="a an and are as at be but by for if in into is it
                         no not of on or s such t that the their then
                         there these they this to was will with"/>
        <pluralmap path="./conf/pluralFolding/pluralMap.txt.gz"/>
        <accentmap path="./conf/accentFolding/accentMap.txt"/>
        <spellcheck createDict="yes"/>
    </index>

</textIndexer-config>
```

In this example, we see that the index name is “default”, that the source and index paths are `../data` and `../index` respectively (relative to the directory in which the textIndexer configuration file resides), the chunk size and overlap are the recommended sizes of 200 and 20 words, a list of stop-words is active for the index, and the example document selector has been specified.

The Document Selector

XTF provides out-of-the-box support for the following types of documents

- Microsoft Word
- PDF
- Web pages (html/htm)
- XML encoded
- plain text
- Scanned books from Internet Archive and HathiTrust

As the previously mentioned, the TextIndexer uses an XSLT based ***document selector*** stylesheet to identify which documents in the source tree should be indexed. Using an XSLT stylesheet allows arbitrary selection criteria to determine which source files to index, and which ones to ignore.

The default document selector provided with the default XTF installation will index any document files whose names end in `.xml`, `.htm` / `.html`, `.pdf`, `.txt`, or `.doc`. You can use the provided stylesheet as the basis for defining your own document selection logic if you wish. However, writing and maintaining the XSLT document selector is beyond the scope of this deployment guide, and will not be discussed here. Please refer to the Document Selector Programming section of the XTF Programming Guide for more information about writing your own document selector.

The Document Pre-Filter

The TextIndexer can make use of an XSLT based document pre-filter stylesheet to restructure the source document just prior to indexing without changing the stored source document. Normally this feature is used to insert or mark meta-data for a document, or to insert additional sectioning attributes before indexing is performed.

Once a pre-filter has been defined, it is used on any source documents identified by the document selector stylesheet for an index. The default sample pre-filters are a good starting place. Here are some you may find interesting:

teiPreFilter.xsl

eadPreFilter.xsl

Because writing and maintaining the XSLT pre-filter is beyond the scope of this deployment guide, its format will not be discussed here. Please refer to the XTF Programming Guide for more information about document pre-filters.

Adding/Deleting/Updating Source Documents

When new source documents are added to the document library or when existing documents are updated or deleted, the ***textIndexer*** tool must be used to update the index. Fortunately, nothing special needs to be done to inform the textIndexer about which documents have been updated, added, or deleted. The program automatically detects the changes and takes the appropriate actions. It has enough smarts to avoid re-indexing documents that have not changed since the last time the index was generated.

Updating the index can occur any time after a document has been added, deleted or changed. You might wish to run the textIndexer tool manually or as part of a script whenever a change to the document library is made. Alternately, on a Unix system you could schedule a daily or weekly *cron* job that made any accumulated changes to the document library and then re-indexed afterwards. Regardless of the method selected, it is best to re-index as soon as possible after the document library is updated so as to minimize the chance of a user performing a search with an index that doesn’t match the actual document set.

Generating Index Summary Reports

There may be times when you want more information about an index created by the ***textIndexer***, and the **indexStats** tool can be used to do this. The indexStats tool is a command-line tool that is invoked as follows:

```
$ indexStats -config config-file-path -index index-name
```

The `-config`
config-file-path argument is an optional argument that identifies an index configuration file to use. If none is specified, the default `XTF_HOME/conf/textIndexer.conf` file is used. The `-index`
index-name argument is required, and identifies the index for which to generate summary information.

For the sample document library provided, the command line to generate a summary report would be:

```
$ indexStats -index default
```

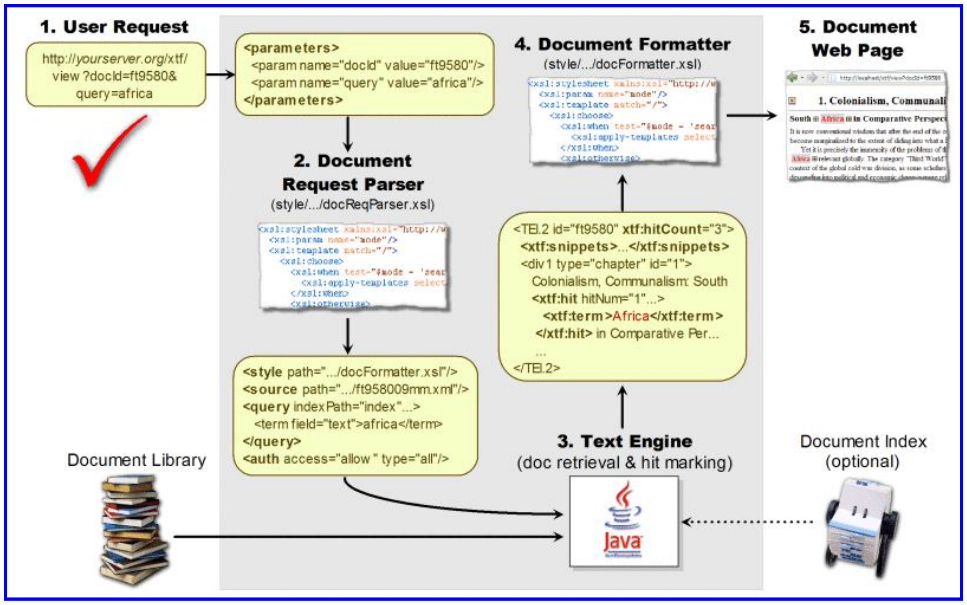
The output generated by the indexStats for the sample index would look something like this:

```
IndexStats v2.x

Index: "default"

  Configuration Info...
    Chunk Size = 200, Overlap = 20
    Index Path = C:/UCOP/Tomcat/webapps/xtf/index/
    Data Path  = C:/UCOP/Tomcat/webapps/xtf/data/
    Stop Words = a an and are as at be but by for if in into is it no not of
      on or s such t that the their then there these they this to was will with
  Statistics...
    Total Documents (Records)  = 15
    Total Chunks               = 16537
    Avg Chunks Per Doc/Rec     = 1,033.6
    Total Number of Src Files  = 15
    Avg Docs/Recs Per File     = 1
    Size of Lucene Index       = 25.31 Mb
    Size of Source Files       = 17.81 Mb
    Size of Lazy Trees         = 14.17 Mb
    Total Index Size           = 39.48 Mb  (Lucene + Lazy)

Done.
```

In this report, the **Avg Chunks Per Doc/Rec** entry reports the average number of chunks that a document was sub-divided into when it was added into the index. The **Size of Lazy Trees** entry indicates how large the current Lazy Tree is for the index, and identifies how much space is currently being used to maintain information to speed access to documents retrieved from the index. To learn more about document chunks and lazy tree functionality, see the XTF Under the Hood Guide.

Configuring the dynaXML Servlet

The dynaXML servlet is responsible for taking a request for a document, converting it to XML if necessary, optionally authenticating the user, and finally presenting the document as a web page. The organization of the dynaXML servlet can be illustrated as follows:



1. User Request
http://yourserver.org/xtf/view?docId=ft9580&query=africa
2. Document Request Parser
(style/.../docReqParser.xsl)
3. Text Engine
(doc retrieval & hit marking)
4. Document Formatter
(style/.../docFormatter.xsl)
5. Document Web Page
1. Colonialism, Communali
Document Library
Document Index
(optional)
Java

```
<parameters>
  <param name="docId" value="ft9580"/>
  <param name="query" value="africa"/>
</parameters>
```

```
<style path=".../docFormatter.xsl"/>
<source path=".../ft958009mm.xml"/>
<query indexPath="index"...>
  <term field="text">africa</term>
</query>
<auth access="allow" type="all"/>
```

```
<TEI.2 id="ft9580" xtf:hitCount="3">
  <xtf:snippets>...</xtf:snippets>
  <div1 type="chapter" id="1">
    Colonialism, Communalism: South
    <xtf:hit hitNum="1"...>
      <xtf:term>Africa</xtf:term>
    </xtf:hit> in Comparative Per...
    ...
</TEI.2>
```

The dynaXML servlet is configured by modifying one or more of the following files:

- The `dynaXML.conf`
  Configuration File
- The `docReqParser.xsl`
  Document Request Parser
- The `errorGen.xsl`**Error Page Generator**
- One or more **Document Formatter** stylesheets

Each of these files is discussed in turn in the following subsections.

The dynaXML.conf Configuration File

The `dynaXML.conf` configuration file is used by the ***dynaXML*** servlet to locate its **Document Request Parser** and **Error Page Generator** XSLT templates. It is also used to adjust the behavior of the servlet itself. Each tag and its associated attributes are discussed here, and a sample `dynaXML.conf` can be found in the `conf` subdirectory below the servlet base directory (i.e., the `XTF_HOME` directory.).

```
<dynaXML-config>

    <logging level="LoggingLevel"/>

    <docReqParser path="LocationOfRequestParser" params="ParamsToPass"/>
    <errorGen path="LocationOfErrorPageGenerator"/>

    <reverseProxy IP="ddd.ddd.ddd.ddd" marker="MarkerString"/>

    <stylesheetCache size="Entries" expire="Seconds"/>
    <ipListCache size="Entries" expire="Seconds"/>
    <authCache size="Entries" expire="Seconds"/>
    <loginCache size="Entries" expire="Seconds"/>

    <dependencyChecking check="YesOrNo"/>

    <reportlatency report="YesOrNo" {cutoffSize="NumBytes"}/>
    <stylesheetProfiling profile="YesOrNo"/>
    <runawaytimer {normalTime="Seconds"} {killTime="Seconds"}/>
    <trackSessions track="YesOrNo"/>
    <cacheControl allowBrowserCaching="YesOrNo"/>
    <lazyFiles use="YesOrNo" buildAlone="YesOrNo"/>

    <!-- Put pass-through tags here, if you need any. -->

</dynaXML-config>
```

These tags and attributes have the following meanings:

`<logging level="`*LoggingLevel*`"/>`	This optional tag sets the level of logging output by the dynaXML servlet. The possible output levels are defined as follows: **`errors`**: Only error messages are displayed. **`warnings`**: Both error and warning messages are displayed. **`info`**: Error, warning, and informational messages are displayed. **`debug`**: Low level debug output is displayed in addition to error, warning, and informational messages. If this argument is not specified, the dynaXML servlet defaults to displaying informational (info) level messages.
`<docReqParser path="`*LocationOfDocReqestParser*`"/>`	This is a required tag. The `path` attribute specifies the name and path to the XSLT template used by the dynaXML servlet to convert incoming document request URLs to the internal XML format actually used by the servlet. The path is interpreted relative to the servlet's base (i.e., the directory assigned to `XTF_HOME`.) For a more detailed description of this tag, see the comments in the sample `docReqParser.xsl` file. *Note*: the old `params` attribute is obsolete and is silently ignored if specified.
`<errorGen path="`*LocationOfErrorPageGenerator*`"/>`	This is a required tag. When an error occurs (either user authentication or an internal error), the XSLT template specified by this configuration tag is used to produce a nicely formatted error page for the requestor. The path is interpreted relative to the servlet's base (i.e., the directory assigned to `XTF_HOME`.) For a more detailed description, see the comments in the sample errorGen.xsl files.

`<reverseProxy IP="`*ddd.ddd.ddd.ddd*`"`
`marker="`*MarkerString*`"/>`
	This optional tag is only required if IP-based authentication uses a reverse proxy server. Using this tag tells dynaXML servlet that the incoming IP address belongs to the reverse proxy server, and that the actual requestor's IP address must be read from the `X-Forwarded-For` marker in the HTTP Request Header. Set the `IP` attribute to the IP address for the reverse proxy server. Use the `marker` attribute if you wish to tell the dynaXML servlet to use a forwarding marker other than the default `X-Forwarded-For` marker.

`<stylesheetCache size="`*Entries*`"` `expire="`*Seconds*`"/>`, etc.	These optional tags override the defaults number of entries and the expiration time (in seconds) for the various caches maintained by the servlet. Usually, these values need not be changed. `styleSheetCache`: Compiled versions of stylesheets (request parser, doc formatter, etc.). `ipListCache`: Compiled versions of IP lists (used only for IP-address based authentication.) `authCache`: Session IDs of successful authentication attempts (LDAP or external.) When expired, the user will be forced to log in again. `loginCache`: Security info for external logins, to verify authentication when user returns from the external page. If it expires while user is logging in, they will be forced to try again.

`<dependencyChecking`
`check="`*YesOrNo*`"/>`
	This optional tag specifies whether dynaXML should perform dependency checking. Whenever consulting one of its internal caches, the dynaXML servlet can optionally check if the source file(s) have been changed and if so ignore the cache entry and reload the file(s). This hurts performance slightly, but makes testing and development much easier. Set this to `no` for a production system where every last little bit of speed is critical. Otherwise, set it to `yes`.

`<reportlatency report="`*YesOrNo*`"`
`cutoffSize="`*NumBytes*`"/>`
	This optional tag controls whether the servlet will report the latency of each request, that is, how long each request took to process and deliver to the client. See the discussion of latency reporting below.

`<stylesheetProfiling`
`profile="`*YesOrNo*`"/>`
	Controls whether a profile is generated for each document request. See the discussion of stylesheet profiling below.

`<runawaytimer normalTime="`*Seconds*`"`
`killTime="`*Seconds*`"/>`
	Controls whether the servlet tracks, reports, and optionally kills potential "runaway" requests. See extended discussion below.

`<trackSessions track="`*YesOrNo*`"/>`	Controls whether the servlet associates a session with each request. If enabled, a session will be created the first time a given user connects to the servlet. The session is assigned a unique identifier which is passed through on subsequent page views either through a cookie or by encoding the session identifier in the URL. The main purpose of sessions is to store long-lasting or large data items that would be inconvenient to pass as URL parameters. The data can be stored and retrieved using XSL extension functions available in XTF. See Tracking Session State in the XTF Programmer's Guide for details. This should generally be enabled.

`<cacheControl`
`allowBrowserCaching="`*YesOrNo*`"/>`
	Since XTF pages are by nature dynamic, it typically does not make sense for browsers to cache them, and thus allowBrowserCaching should generally be set to "no". However, if you have particular performance concerns and are prepared to address caching problems (e.g. by adding extra URL parameters to AJAX requests) then set it to "yes". When set to "no" as recommended, XTF will insert an HTTP header in each response telling browsers that the

	page has expired (meaning it should not be cached.) Other cache control HTTP parameters aren't as safe to use and cause subtle problems (for instance refer to this article: see http://edn.embarcadero.com/print/39141)
`<lazyTrees use="`*YesOrNo*`" buildAlone="`*YesOrNo*`"/>`	Controls whether the servlet uses (and optionally builds by itself) lazy trees. See extended discussion below.

Latency reporting

When the `<reportLatency>` tag's `report` is set to `yes`, the servlet will record and log how many milliseconds were spent processing each request (regardless of whether the request succeeded or produced an error page.) This information can be useful for accumulating statistical data, or to target optimization efforts on those requests that took longest to serve. Set it to `no` if you want to avoid logging and don't need the information.

Depending on the data you're looking for, you might want to exclude latency of large requests (e.g. a whole book). To do this, set the `cutoffSize` attribute to a number of bytes; when any request has output that much data, the servlet will report the latency immediately, labeled "Latency (cutoff)". When the request finally completes, the total latency will be reported with the label "Latency (final)". Leaving out the `cutoffSize` attribute, or setting it to zero, disables cutoff reporting.

Latency reporting has very little impact on system performance, so is safe to enable in production environments.

Note: The reports are logged at the "info" level, so if the `<logging>` level above is set to `warnings` or `errors`, nothing will appear.

Stylesheet profiling

A profile identifies the stylesheet and line number of each XSLT command that was executed for the query, and a relative count that identifies roughly how long it took to execute that command. Within the profile, the executed commands are ordered such that the most used commands appear first, and the least used commands appear last. This provides clues as to where the stylesheet is spending its time and thus could possibly be optimized. When enabled, profiles are written to the dynaXML servlet's activity log. Set this to `no` for general use, since collecting the profile information can be time-consuming.

Note: The profile information is logged at the "info" level, so if the `<logging>` level above is set to `warnings` or `errors`, nothing will appear.

Runaway timer

The `<runawaytimer>` tag's `normalTime` attribute specifies the maximum amount of time, in seconds, that requests are generally expected to take. Any requests that exceed this time will be logged (with a call stack trace on Java 5.0 and above.) If set to zero or not specified, no logging of potential runaway requests will be performed. The `killTime` attribute specifies the time after which a request is truly considered runaway and should be killed. After this time is exceeded a flag is set, and many of the major processing loops in XTF will check the flag and kill off the request. However, some parts (deeply buried in third-party libraries) do not check this flag and thus the request might not be successfully killed. If set to zero, threads will never be killed before they complete.

Runaway timing can introduce significant overhead, and should be enabled only if one is experiencing problems that might be due to runaway requests. Set these to zero for general use.

Note: Runaway reports are logged at the "info" level, so if the `<logging>` level above is set to `warnings` or `errors`, nothing will appear.

Lazy trees

The `<lazyTrees>` tag's `use` attribute controls whether dynaXML will use "lazy trees." These files are generally speed processing. If not specified, this attribute defaults to "yes" For detailed information on lazy trees see XTF Under the Hood.

The `buildAlone` attribute controls whether dynaXML will **create** lazy trees if they're not already built by the ***textIndexer***. If not specified, this attribute defaults to "no". Most users will want to leave this at its default "no" to ensure that the indexer keeps the lazy trees in sync with the index at all times. However, if you are using dynaXML completely independently, without using textIndexer, then enable `buildAlone` to build lazy trees in dynaXML.

The docReqParser.xsl Request Parser

The docReqParser.xsl template describes how the ***dynaXML*** servlet should translate external document request URLs into internal XML based document requests. As part of the translation, the template can also optionally request authentication of the user that made the document request. A sample `docReqParser.xsl` is provided in the `conf` subdirectory below the `XTF_HOME` base directory under which the XTF system was installed. A full discussion of the internal workings of `docReqParser.xsl` is contained in the XTF Programming Guide.

The errorGen.xsl Error Page Generator

The errorGen.xsl template describes how the ***dynaXML*** servlet should generate an HTML page whenever there is an error that needs to be reported back to the user. A sample `errorGen.xsl` is provided in the `style/dynaXML` subdirectory below the `XTF_HOME` base directory under which the XTF system was installed. A full discussion of the internal workings of `errorGen.xsl` is contained in the XTF Programming Guide.

The dynaXML Document Formatters

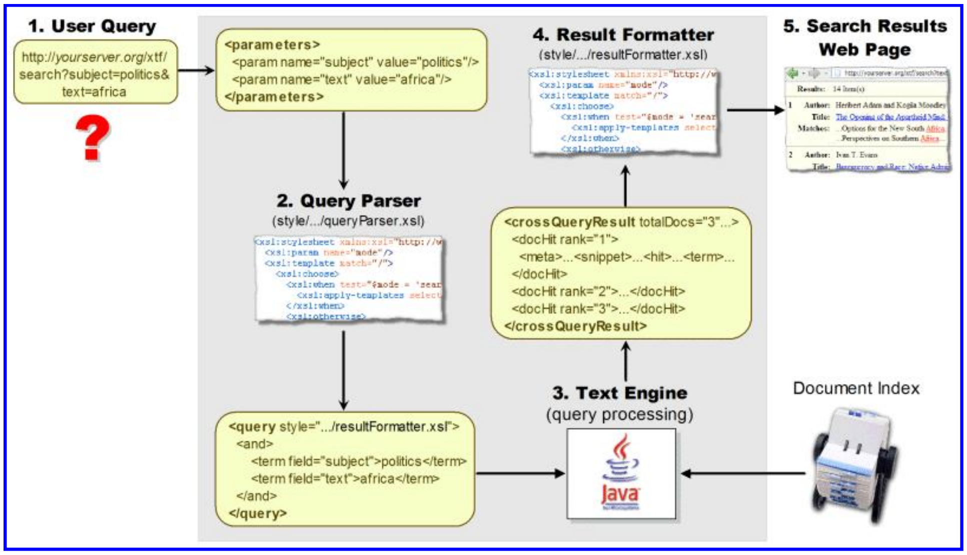
Any number of templates can exist to describe how the ***dynaXML*** servlet should translate the XML file for a document in the library into HTML that can be displayed in the user's browser. Which document formatter to use is determined by a tag in the `docReqParser.xsl` template. Sample document formatters are provided in the `style/dynaXML/docFormatter/tei` and `style/dynaXML/docFormatter/ead` subdirectories below the `XTF_HOME` base directory under which the XTF system was installed. The document formatters are similar but implement different output formatting schemes. A full discussion of the internal workings of the document formatter templates is contained in the XTF Programming Guide.

Running the dynaXML Servlet

Assuming that the servlet container you are using has been configured to invoke it, running the ***dynaXML*** servlet consists of simply issuing a valid document request URL. A sample request was illustrated in the Quick Start section entitled *Verify that Document Retrieval (dynaXML Servlet) Works.*

Configuring the crossQuery Servlet

The ***crossQuery*** servlet is responsible for taking a user search query and locating all files in the document library that match the specified search criteria. The resulting matches are then displayed in an HTML web page for the user to browse. The organization of the crossQuery servlet can be illustrated as follows:


1. User Query
http://yourserver.org/xtf/search?subject=politics&text=africa
?
2. Query Parser
(style/.../queryParser.xsl)
3. Text Engine
(query processing)
Java
4. Result Formatter
(style/.../resultFormatter.xsl)
5. Search Results Web Page
Document Index

```
<parameters>
  <param name="subject" value="politics"/>
  <param name="text" value="africa"/>
</parameters>
```

```
<xsl:stylesheet xmlns:xsl="http://w
  <xsl:param name="mode"/>
  <xsl:template match="/">
    <xsl:choose>
      <xsl:when test="$mode = 'sear
        <xsl:apply-templates select
      </xsl:when>
      <xsl:otherwise>
```

```
<query style=".../resultFormatter.xsl">
  <and>
    <term field="subject">politics</term>
    <term field="text">africa</term>
  </and>
</query>
```

```
<crossQueryResult totalDocs="3"...>
  <docHit rank="1">
    <meta>...<snippet>...<hit>...<term>...
  </docHit>
  <docHit rank="2">...</docHit>
  <docHit rank="3">...</docHit>
</crossQueryResult>
```

The crossQuery servlet is configured by modifying one or more of the following files:

- The `crossQuery.conf` **Configuration File**
- The `queryGen.xsl` **Request Parser**
- The `errorGen.xsl` **Error Page Generator**
- One or more **Document Formatter** stylesheets

Each of these stylesheet templates is discussed in turn in the following subsections.

The crossQuery.conf Configuration File

The `crossQuery.conf` configuration file is used by the ***crossQuery*** servlet to locate its **Query Generator** and **Error Page Generator** XSLT templates. It is also used to adjust the behavior of the servlet itself. Each tag and its associated attributes are discussed here, and a sample `crossQuery.conf` can be found in the `conf` subdirectory below the servlet base directory (i.e., the `XTF_HOME` directory).

```
<crossQuery-config>

    <logging level="LoggingLevel"/>

    <queryParser path="LocationOfQueryParser"/>
    <errorGen path="LocationOfErrorPageGenerator"/>

    <stylesheetCache size="Entries" expire="Seconds"/>

    <dependencyChecking check="YesOrNo"/>
    <reportlatency enable="YesOrNo" {cutoffSize="NumBytes"}/>
    <stylesheetProfiling profile="YesOrNo"/>
    <runawaytimer {normalTime="Seconds"} {killTime="Seconds"}/>
    <cacheControl allowBrowserCaching="YesOrNo"/>
    <trackSessions track="YesOrNo"/>

    <!-- Your tags go here -->

</crossQuery-config>
```

These tags and attributes have the following meanings:

`<logging level="`*LoggingLevel*`"/>`	This tag is an optional tag that sets the level of logging output by the crossQuery servlet. The possible output levels are defined as

	follows: `errors`: Only error messages are displayed. `warnings`: Both error and warning messages are displayed. `info`: Error, warning, and informational messages are displayed. `debug`: Low level debug output is displayed in addition to error, warning, and informational messages. If this argument is not specified, the crossQuery servlet defaults to displaying informational (info) level messages.
`<queryParser path="LocationOfQueryParser"/>`	This tag specifies the name and path to the XSLT template used by the crossQuery servlet to convert incoming search query URLs to the internal XML search query format actually used by the servlet. The path is interpreted relative to the servlet's base (i.e., the directory assigned to `XTF_HOME`.) For a more detailed description, see the comments in the sample queryParser.xsl file.
`<errorGen path="LocationOfErrorPageGenerator"/>`	When an error occurs (either authentication or an internal error), the XSLT template specified by this configuration tag is used to produce a nicely formatted error page for the requestor. The path is interpreted relative to the servlet's base (i.e., the directory assigned to `XTF_HOME`.) For a more detailed description, see the comments in the sample errorGen.xsl file.
`<stylesheetCache size="Entries" expire="Seconds"/>`	This tag (if specified) overrides the defaults number of entries and the expiration time (in seconds) for the **styleSheetCache**, which maintains compiled versions of the XSLT templates used by the crossQuery servlet. Usually, these attributes need not be changed.
`<dependencyChecking check="YesOrNo"/>`	Whenever consulting one of its internal caches, the crossQuery servlet can optionally check if the source file(s) have been changed and if so ignore the cache entry and reload the file(s). This hurts performance slightly, but makes testing and development much easier. Set this to "no" for a production system where every last little bit of speed is critical. Otherwise, set it to "yes".
`<reportlatency report="YesOrNo" cutoffSize="NumBytes"/>`	Controls whether the servlet will report the latency of each request, that is, how long each request took to process and deliver to the client. See the discussion of latency reporting below.
`<stylesheetProfiling profile="YesOrNo"/>`	Controls whether a profile is generated for each search request. See the discussion of stylesheet profiling below.
`<runawaytimer normalTime="Seconds" killTime="Seconds"/>`	Controls whether the servlet tracks, reports, and optionally kills potential "runaway" requests. See the discussion of runaway timing below.
`<cacheControl allowBrowserCaching="YesOrNo"/>`	Since XTF pages are by nature dynamic, it typically does not make sense for browsers to cache them, and thus allowBrowserCaching should generally be set to "no". However, if you have particular performance concerns and are prepared to address caching problems (e.g. by adding extra URL parameters to AJAX requests) then set it to "yes". When set to "no" as recommended, XTF will insert an HTTP header in each response telling browsers that the page has expired (meaning it should not be cached.) Other cache control HTTP parameters aren't as safe to use and cause subtle problems (for instance refer to this article: see http://edn.embarcadero.com/print/39141)
`<trackSessions track="YesOrNo">`	Controls whether the servlet associates a session with each request. If enabled, a session will be created the first time a given user connects to the servlet. The session is assigned a unique identifier

which is passed through on subsequent page views either through a cookie or by encoding the session identifier in the URL. The main purpose of sessions is to store long-lasting or large data items that would be inconvenient to pass as URL parameters. The data can be stored and retrieved using XSL extension functions available in XTF. See Tracking Session State in the XTF Programmer's Guide for details. Generally, this should only be enabled if you plan to store and use session data.

Latency reporting

When the `<reportLatency>` element's `report` attribute is set to `yes`, the servlet will record and log how many milliseconds were spent processing each request (regardless of whether the request succeeded or produced an error page.) This information can be useful for accumulating statistical data, or to target optimization efforts on those requests that took longest to serve. Set it to `no` if you want to avoid logging and don't need the information.

Depending on the data you're looking for, you might want to exclude latency of large requests (e.g. a whole book). To do this, set the `cutoffSize` attribute to a number of bytes; when any request has output that much data, the servlet will report the latency immediately, labeled "**Latency (cutoff)**". When the request finally completes, the total latency will be reported with the label "**Latency (final)**". Leaving out the `cutoffSize` attribute, or setting it to zero, disables cutoff reporting.

Latency reporting has very little impact on system performance, so is safe to enable in production environments.

Note: The reports are logged at the "info" level, so if the `<logging>` level above is set to `warnings` or `errors`, nothing will appear.

Stylesheet profiling

A profile identifies the stylesheet and line number of each XSLT command that was executed for the query, and a relative count that identifies roughly how long it took to execute that command. Within the profile, the executed commands are ordered such that the most used commands appear first, and the least used commands appear last. This provides clues as to where the stylesheet is spending its time and thus could possibly be optimized. When enabled, profiles are written to the crossQuery servlet's activity log. Set this to `no` for general use, since collecting the profile information can be time-consuming.

Note: The profile information is logged at the "info" level, so if the `<logging>` level above is set to `warnings` or `errors`, nothing will appear.

Runaway timing

The `<runawayTimer>` element's `normalTime` attribute specifies the maximum amount of time, in seconds, that requests are generally expected to take. Any requests that exceed this time will be logged (with a call stack trace on Java 5.0 and above.) If set to zero or not specified, no logging of potential runaway requests will be performed.

The `killTime` attribute specifies the time after which a request is truly considered runaway and should be killed. After this time is exceeded a flag is set, and many of the major processing loops in XTF will check the flag and kill off the request. However, some parts (deeply buried in third-party libraries) do not check this flag and thus the request might not be successfully killed. If set to zero, threads will never be killed before they complete.

Runaway timing can introduce significant overhead, and should be enabled only if one is experiencing problems that might be due to runaway requests. Set these to zero for general use.

Note: Runaway reports are logged at the "info" level, so if the `<logging>` level above is set to `warnings` or `errors`, nothing will appear.

The queryParser.xsl Query Parser

The `queryParser.xsl` template describes how the ***crossQuery*** servlet should translate search query URLs into internal XML based search queries. A sample `queryParser.xsl` is provided in the `style/crossQuery/queryParser/default` subdirectory below the `XTF_HOME` base directory under which the XTF system was installed. A full discussion of the internal workings of the Query Parser is contained in the XTF Programming Guide.

The errorGen.xsl Error Page Generator

The `errorGen.xsl` template describes how the ***crossQuery*** servlet should generate an HTML page whenever there is an error that needs to be reported back to the user. A sample `errorGen.xsl` is provided in the `style/crossQuery` subdirectory below the `XTF_HOME` base directory under which the XTF system was installed. A full discussion of the internal workings of the Error Generator is contained in the XTF Programming Guide.

The Search Result Formatters

Any number of templates can exist to describe how the ***crossQuery*** servlet should translate the XML search results into HTML that can be displayed in the user’s browser. Which formatter to use is determined by a tag in the **Query Parser** stylesheet. A sample search result formatter is provided in the `style/crossQuery/resultFormatter/default` subdirectory below the `XTF_HOME` base directory under which the XTF system was installed. The formatter is called `resultFormatter.xsl`. A full discussion of the internal workings of the search result formatter is contained in the XTF Programming Guide.

Running the crossQuery Servlet

As with the ***dynaXML*** servlet, once your servlet container has been configured to invoke it, running the ***crossQuery*** servlet consists of simply issuing a valid search query URL.

Usage Examples

The ***crossQuery*** and ***dynaXML*** servlets are independent servlets that can be used in a stand-alone or combined fashion. The following subsections provide sample stand-alone and combined configurations to help you create your own on-line document libraries.

Stand-Alone dynaXML System

Using the ***dynaXML*** servlet by itself without the ***crossQuery*** servlet would give a site the ability to read on-line documents without any search capabilities. This kind of arrangement may be useful if storage space is at a premium. Eliminating the search feature for a 2GB document set will save eliminate the need for an additional 4GB of storage required to maintain the document search index and its associated partial-file retrieval database.

In a system where the indexing overhead is unacceptable, the site would simply provide one or more web pages with links for each document available in the on-line document library. The links on the web page would simply issue dynaXML style document request URLs when clicked. For example, using the sample document from the Quick Start guide, an HTML code snippet for retrieving the document might look as follows:

```
<a href="http://yourserver:8080/xtf/view?docId=tei/ft958009mm/ft958009mm.xml&doc.view=frames"
    The Opening of the Apartheid Mind
</a>
```

Using this approach, a site could provide a few different kinds of web pages (e.g., organized alphabetically, by subject, by author, etc.), and achieve reasonably good access to documents without incurring the overhead of maintaining a full-text search index.

crossQuery Stand-Alone System

Using the ***crossQuery*** servlet by itself without the ***dynaXML*** servlet would give a site the ability to search for on-line documents without the ability to read them. This kind of arrangement may be useful when creating a catalog for the purchase of on-line documents. The search engine would allow customers to search for relevant documents, and would show them text samples, but not the entire document.

In a simple on-line catalog system, the site would simply provide a query page for entering the author, publisher, or title of the book, and would return results along with prices and purchase options. The query page would generate crossQuery style query URLs from the user input and pass them on to the crossQuery servlet for processing. The XSLT **Search Result Formatter** template would then be changed from the sample provided to simply report the pricing information along with a link for ordering. A slightly more advanced system would permit text searches, and would display matches in short blurbs like the sample along with book information and purchase links, but no links to bring up the document itself.

Since this kind of a system requires the documents to be indexed, the ***textIndexer*** tool would need to be run once initially to create the index for the site, and then whenever documents were added to or removed from the site.

Combined crossQuery and dynaXML System

Using the ***crossQuery*** servlet in conjunction with the ***dynaXML*** servlet would give a site the ability to both search for and read documents. In this case, the site would provide an initial query page which would generate crossQuery style query URLs. The results page produced yield dynaXML style document request URLs as links that when clicked would invoke the dynaXML servlet to bring up the selected document in a web page.

The sample configuration and XSLT templates included with the XTF `.jar` file implement this type of combined system, and are good starting points for creating a similar, more customized arrangement. In fact, a web site would only need to provide a link to an initial query page to make the sample system available to on-line users.

Since this kind of a system requires the documents to be indexed, the ***textIndexer*** tool would need to be run once initially to create the index for the site, and then whenever documents were added to or removed from the site.

User Authentication (optional)

The ***dynaXML*** servlet can be configured to take advantage of user authentication to limit access to retrieved documents. The types of authentication available are IP List, LDAP, and External. An brief overview of each of these methods is provided in the following sections.

IP List Authentication

With this kind of authentication, the user’s IP address is compared to a list of known IP addresses, and access permitted only if the user’s IP address appears in the list. Installation of this kind of authentication simply consists of creating a text file whose contents are as follows:

```
; Authorize a single address.
168.192.12.8

; Authorize an address range.
123.89.122.5 - 123.89.122.73
```

```
; Authorize an address range using wildcards.
63.79.*.*

; Exclude an address from the previously authorized range.
exclude 63.79.123.42

; Exclude an address sub-range from the previously authorized range.
exclude 63.79.124.*
```

In the this example, the first line authorizes a single IP address to access the document library through the dynaXML servlet. The second and third lines both authorize a range of IP addresses to access the document library, with the second example making use of the `*` wildcard character. The final two lines illustrate how to exclude a single address or an address range from a previously selected IP range (in this case, from the range specified by the `63.79.*.*` line.) In all cases, lines that don't begin with an address or the word `exclude` are treated as comments, and blank lines are ignored.

Once an "Authorized IP List" file has been created, the XSLT based dynaXML **Document Request Parser** instructs the dynaXML servlet to authenticate users via IP List Authentication by inserting `<auth>` tags in its document retrieval requests. For more about modifying the Document Request Parser to generate IP List `<auth>` tags, see the XTF Programming Guide.

(Note: IP-address authentication is a relatively insecure method, as IP addresses can be spoofed. Yet it's often sufficient, and is certainly simple to implement.)

LDAP Authentication

The ***dynaXML*** servlet can also consult an LDAP database to authenticate users. Using this method to authenticate users requires that a working LDAP database exists on your server, and a description of how create, configure and install one is well beyond the scope of this document. For further assistance, please refer to your LDAP server documentation, or ask your system administrator.

As with **IP List Authentication** above, the dynaXML **Document Request Parser** instructs the dynaXML servlet to authenticate users with LDAP by inserting `<auth>` tags in its document retrieval requests. For more about modifying the Document Request Parser to generate LDAP `<auth>` tags, see the XTF Programming Guide.

(Note: dynaXML will use the HTTP "basic" protocol to obtain a user name and password. This method is insecure in that the password is transmitted "in the clear" and thus can easily be snooped. If security is of paramount concern, use the **External Authentication** mechanism described in the next section.)

External Authentication

The "external" authentication mechanism uses an external web page to check the user's permission, and uses a special protocol to ensure security during the transaction. External authentication provides great flexibility, allowing form-based login, `.htpasswd` apache-style checking, **Shibboleth**, and any other method that uses a web page.

To use External Authentication you must have an HTTP server capable of receiving a URL request for user verification. The URL sent to the server will have the form:

```
http://your-server/your-authentication-page?returnto=returnpage;nonce=somestring
```

where

returnpage is the dynaXML output page to return to once authentication is accepted.
somestring is a nonsense string that dynaXML generates as a unique ID specifically for this login attempt.

The web-page specified by *your-authentication-page* is responsible for identifying and authenticating the user via a login script or some other method. Once your web-page has authenticated the user, it should redirect the user's web browser to the URL specified by the `returnto` parameter. The redirect URL should have the form:

> *returnpage*`;nonce=`*somestring*`;hash=`*hashvalue*

where

returnpage	is the `returnpage` received in the original external authentication URL, and
hashvalue	is a hex-encoded MD5 hash of the string *somestring*`:`*key*, with *key* being a secret key string known by both the external login page and the dynaXML servlet.

Use of the *somestring* string (also known as a "nonce" value) makes each attempt unique and defends against replay attacks. Hashing the nonsense string with the secret key ensures that the secret key cannot be practically guessed by an outsider.

Like the previous two authentication methods, the dynaXML **Document Request Parser** instructs the dynaXML servlet to authenticate users via External Authentication by inserting `<auth>` tags in its document retrieval requests. For more about modifying the Document Request Parser to generate External `<auth>` tags, see the XTF Programming Guide.

Adding Language Options to the Interface (Optional)

XTF 3.0 ships with default interface translations for English, French, Spanish, German and Italian. A link appears on each page, which when clicked, generates a form allowing the user to choose from these languages, or others if they have been added according to the instructions below.

Four tasks are involved in adding a new language option:

- Creating a mapping template
- Translating the English terms into the <to> element
- Expanding with additional words and phrases
- Adding a new radio button to the language form

Creating a new mapping template

One mapping translation file is required for each language that will be supported. Copy the translation template file, xtf/style/xtfCommon/g10n/translation_pl.xml and rename it using an appropriate language acronym. For instance, to create a mapping translation file for "Pig Latin", one would copy "translation_template.xml" and rename it "translation_pl.xml"

Translating the English terms into the <to> element

Open the new translation mapping file in an editor, and provide a translated phrase for each element you want to translate. If there is no translation, the default language, English is used. Below is a chunk from the Pig Latin mapping file:

```
<trans>
    <from>Add</from>
    <to>Add-ay</to>
</trans>
<trans>
    <from>Advanced</from>
    <to>Advanced-ay</to>
</trans>
```

```
<trans>
    <from>author</from>
    <to>author-ay</to>
</trans>
<trans>
    <from>Author</from>
    <to>Author-ay</to>
</trans>
```

Expanding with additional words and phrases

Any new language strings added to the interface as part of customization can also be added to this translation file. Simply include additional elements following this format:

```
<trans>
    <from>Term</from>
    <to>Translated_Term</to>
</trans>
```

Adding a new radio button to the language form

After creating the new translation file, the next step is to create a way for users to choose that language. To do this, open up the following style sheet in an editor:

xtf/style/xtfCommon/xtfCommon.xsl

Within `xtfCommon.xsl` search for “French” to find the language chooser section. It should be at about line #231.

Add a new `<input>` element like the one below, substituting your language acronym for the “pl” value:

```
   <input type="radio" name="lang" value="pl">
      <xsl:if test="$lang='pl'"><xsl:attribute name="checked" select="'checked'"/></xsl:if>
      <xsl:text>Atin-lay Ig-pay</xsl:text>
   </input>&#160;
```

Working with Large Indexes: Index Validation & Rotation

Putting a large XTF index into production involves additional work that might not be required for small installations. There are two main problems that need to be addressed:

- Because large indexes take a long time to build, care should be taken to ensure that an incomplete index is never shown to end-users. To address this you can optionally configure **index validation** to apply some basic sanity checks to an index before it is seen.
- As new objects come in and are incrementally added to an index, the time penalty incurred to load each new version of the index increases. XTF supports **index rotation** and **background warming** to address this.

The following sections will describe these features and tell you how to configure them.

Index Validation

If your collection has 40,000 items, and the end-user interface only contains 100 items, something is wrong. This can happen in a number of ways, perhaps by accidentally doing a `-clean` re-index but specifying a sub-directory. Or, putting an index into production that is still being built or aborted due to an error. These preventable problems are the reasons to add index validation to your XTF toolbox.

Validation is used in conjunction with index rotation (below). You set up one or more validation checks to perform at the end of an index run. The checks can be either ***crossQuery*** or ***dynaXML*** queries and you can ensure that a certain number of hits (across the collection or within a document, respectively) are present before allowing a new index to be rotated into production.

To enable index validation, follow these steps:

1. Add a validation tag to your `conf/textIndexer.conf` file that this tells the textIndexer to perform validation by default, and to look in the `indexValidation.xml` file for specific instructions. The tag looks like this:

```
<validation path="./conf/indexValidation.xml"/>
```

2. Again in the `conf` directory, create an `indexValidation.xml` file. You specify a number of crossQuery URLs (the server name will actually be ignored, so you can put anything). The indexer will simulate access to each URL, and verify that the number of results returned is at least the minHits that you specified. Similarly you can specify a number of dynaXML queries on specific documents you know to be in your collection, and the indexer will verify that the minimum number of hits within that document were produced. Here's a sample of what its contents could contain (you'll need to customize to your specific case):

```
<?xml version="1.0" encoding="utf-8"?>
<index-validation>
    <crossQuery minHits="28000">http://yourserver.org/xtf/search?browse-all=yes</crossQuery>
    <crossQuery minHits="2000">http://yourserver.org/xtf/search?keyword=italy</crossQuery>
    <dynaXML minHits="10">http://yourserver.org/xtf/view?docId=0001d5b1;query=karok</dynaXML>
</index-validation>
```

3. Enable index rotation (see the next section), and validation will be automatically part of the process.

An additional benefit of adding validation to your process is that the servlets (crossQuery and/or dynaXML) will automatically "warm up" a new index in the background using the validation URLs. This way, when a new index is rotated in and made available to users for searching, it has already had a few URLs queried and thus subsequent user queries should be fast. For this reason, you should try to make the validation queries hit most of the major features of your user interface (e.g. facets, browse pages, etc.) to prevent lag when users access those features for the first time after a new index is rotated in.

If for testing or any other reason you wish to skip the validation step at index time, you can specify the "-novalidate" command line flag to textIndexer.

Index Rotation

Consider the situation where you have a working index with many items, and lots of people are actively using it. Now you have a pretty big change to your index pre-filter. How do you build a new index in the background without interrupting the work of your users, or worse, showing them an incomplete index? Index rotation is the answer.

When you enable index rotation, the ***textIndexer*** will build new indexes (whether clean or incremental) in a special directory called `index-new` (if your main directory is `index`; otherwise, it'll be your index name with "`-new`" appended to it). When the index is completed (and passes validation if you've enabled it), the indexer will make a lightweight copy (more on that later) of `index-new` and name it `index-pending`.

The servlets are constantly on the lookout for `index-pending`, and when it appears they will "warm it up" by running your validation URLs against the new index (if you've enabled validation), and then make it available by renaming the old `index` directory to `index-spare`, and the new `index-pending` directory to just `index`.

The next time the indexer runs, it'll bring the old `index-spare` directory up to date and recycle it for use as the next `index-new`. This saves time and disk space. The whole process can be seen as rotating between four index directories. Let's start with version 1. At the end of the first index run things will look like this:

`index-new` (v1) `index-pending` (v1)

When the servlets notice the pending directory we'll get:

`index-new` (v1) `index` (v1)

Now let's see what happens when we do an incremental update, producing version 2.

`index-new` (v2) `index-pending` (v2) `index` (v1)

As you can see, version 1 is still what end-users will see when they access the servlets. In the background, the servlets will notice the new `index-pending`, warm it up, and rotate, leaving things like this:

`index-new` (v2) `index` (v2) `index-spare` (v1)

Let's do one more index run to get to version 3.

`index-new` (v3) `index-pending` (v3) `index` (v2)

Here the old version 1 `index-spare` has been recycled, completing the rotation. Another way to look at the process is to see that any specific version of an index goes through four stages:

Version 1: `index-new` -> `index-pending` -> `index` -> `index-spare`

A potential problem is disk space: as you can see, at any given moment there might be four separate copies of the index (new, pending, main, and spare). XTF takes extra care to minimize disk space requirements by sharing as much data as possible between these various versions of the index. This is done by using UNIX "hard links" to allow multiple directories to point to the same file data, eliminating duplication. Thus, XTF makes a "lightweight" copy which is not only smaller than a full copy, but also can be faster to make as very little data need be copied.

Note: One consequence of the lightweight copy strategy relying on UNIX hard links is that it doesn't work on Windows. Thus, XTF's index rotation feature only works on UNIX variants such as Linux, Mac OSX, and Solaris. In particular, the textIndexer uses the UNIX "rsync" utility which must be present at index time for rotation to work properly.

If you want to skip the rotation step at the end of indexing, you can specify the "-norotate" command-line flag to textIndexer.

Edit this entry.

Latest XTF News

- XTF 3.0 beta
- XTF Website Launched
- XTF Community Preview
- XTF 2.2 released

Subscribe to XTF News

- RSS

The **eXtensible Text Framework (XTF)** is supported by the California Digital Library