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

Tag Reference

XTF Tag Reference

This page provides a reference for the tags, parameters, and utilities usable in XSLT stylesheets in various parts of the eXtensible Text Framework (XTF). This document assumes that you are familiar with the general organization of the XTF system as described in the XTF Programmer's Guide. Here are the available sections:

- textIndexer
  - Document Selector
    - Input Tags
    - Output Tags
  - PreFilter
    - Output Attributes
- crossQuery
  - Common to crossQuery and dynaXML
    - Stylesheet parameters
    - Parser Input Tags
    - Error Generator Stylesheet
    - Utilities
  - Query Router and Query Parser Tags
    - Common Input Tags and Parameters
    - Query Router Output Tags
    - Query Parser Output Tags
  - Search Result Formatter
    - Input Tags
  - Error Generator
    - Input Tags
- dynaXML
  - Common to crossQuery and dynaXML
    - Stylesheet parameters
    - Parser Input Tags
    - Error Generator Stylesheet
    - Utilities
  - Document Request Parser
    - Input Tags and Parameters
    - Output Tags
  - Document Formatter
  - Error Generator

textIndexer

return to top

This section details the XTF specific tags for the textIndexer stylesheets. There are two kinds of tags: 1) Document Selector tags and 2) PreFilter tags.

1. Document Selector

The following input and output tags make up the XML input for the **Document Selector** stylesheet. They constitute a simple XML representation files found in one (sub-) directory of the document library.

- **Input Tags**
  - Directory Tag
    This tag is the outermost tag for the XML input fragment sent to the **Document Selector** stylesheet for translation. It has the form:

```
<directory dirPath="DirectoryPath">
    File
```

```
    File
    …
    File
</directory>
```

where

`dirPath="!DirectoryPath"`	is the absolute file path to the directory on disk.
`File, File...`	is zero or more **File Tags** (see below) , one for each file found in the directory.

- File Tag

  This tag is the input to the **Document Selector** stylesheet for each file found in the containing `<directory...>` tag. It has the form:

```
<file fileName="FileName"/>
```

where

`fileName="!FileName"`	is the name of a file found in the directory identified by the containing `<directory...>` tag. Note that this file name does not contain any path information for the file, but only the file name itself.

- **Output Tags**
  - Index List Tag

    This tag is the outermost tag for the XML output fragment issued by the **Document Selector** stylesheet. It has the form:

```
<indexFiles>

    FileToIndex
    FileToIndex
    …
    FileToIndex

</indexFiles>
```

where

`FileToIndex, FileToIndex...`	is zero or more **File To Index Tags** (see below), one for each file to index in the directory.

  - File To Index Tag

    One copy of this tag should be output by **Document Selector** stylesheet for each file that must be indexed. It should appear within an Index List Tag container (see above). It has the form:

```
<file fileName      = "FileName"
      {format       = "FileFormat"}
      {preFilter    = "PreFilterPath(s)"}
      {displayStyle = "DocumentFormatterPath"}/>
```

where

`fileName="`*FileName*`"`	is a required attribute that specifies the name of a file to be indexed. Note that this file name should not contain any path information for the file, but

	only the file name itself.
`format="`*FileFormat*`"`	is an optional attribute that specifies the format of a file to be indexed. Currently XML, PDF, HTML plain text, and most Microsoft Word files are handled by the textIndexer, and the format attribute should correspondingly be set to `XML`, `PDF`, `HTML`, `Text`, or `MSWord`. If this attribute is omitted, the textIndexer will try to infer the file type based on the file extension.
`preFilter="`*PreFilterPath(s)*`"`	is an optional attribute that specifies the path to the **Pre-Filter** stylesheet to be applied to this file. If this path is not specified as an absolute path, it is assumed to be relative to the XTF base installation directory (i.e., XTF_HOME.) Multiple pre-filters may be specified in a list; they should be separated by ";" or "," characters. The pre-filters will be applied in the order listed (e.g. the original file is sent to the first pre-filter; its output is sent to the second pre-filter, whose output is sent to the third, etc.) If this attribute is omitted, no pre-filter will be applied to the file.
`displayStyle="`*DocumentFormatterPath*`"`	is an optional attribute that specifies path to the **Document Formatter** stylesheet to use for this file. If this path is not specified as an absolute path, it is assumed to be relative to the XTF base installation directory (i.e., XTF_HOME.) If this attribute is present, the textIndexer will create a special cache that is used by the ***dynaXML*** servlet to display the current file more quickly. If this attribute is omitted, the cache is not created. For more details, see the discussion of Lazy Document Handling in the XTF Under the Hood guide.

2. Pre-Filter

- **Output Attributes**

  This section summarizes the attributes defined for the **Pre-Filter** stylesheet to output which have meaning for the textIndexer tool.

  - Index/No-index Attribute

```
<xsl:attribute name="xtf:index" select="'TrueOrFalse'"/>
<xsl:attribute name="xtf:noindex" select="'TrueOrFalse'"/>
```

  This attribute is used to turn on/off indexing for a tag in a source document. The noindex variant is simply a logical inverse of the index variant. Both are provided as a convenience to the programmer.

  The value for either of these tags should be set to either the string '`true`' or the string '`false`'. (Note: If not explicitly set, nested sub-tags for a document inherit the index/noindex state from

the closest parent tag for which an index state is defined.)

This attribute can be used for normal text blocks, and also on blocks marked as metadata using the `xtf:meta` attribute below. In both cases, it controls whether the given block of text, or meta-data field, is added to the index. In the case of meta-data, a field that isn't added to the index will still be made available to the **Result Formatter** stylesheet when crossQuery results are displayed.

- Meta-Data Attribute

```
<xsl:attribute name="xtf:meta" select="'TrueOrFalse'"/>
```

This attribute is used to mark the contents of a tag as being part of the meta-data for a document rather than main-body text for the document.

The select value for this tag should be set to either the string '`true`' (text in tag is meta data) or the string '`false`' (text in tag **is not** meta data.)

The entire tag and its contents will be treated as meta-data and will be added to the index using the element name as the name of the meta-data field. That is, the tag will be indexed separately from the full text of the document.

Other attributes of the tag, and any embedded element tags, will be stored in the index and will be passed verbatim to the **Result Formatter** stylesheet to be used for output purposes. Of course the text of the element and any sub-elements will be searchable, but the actual attributes and element tags themselves cannot be searched for.

Note: If you mark a section of text with the `xtf:meta` attribute, it will not be included in the full text index of that document (accessed by querying the text field). If you want a given piece of text to appear in both the meta-data and full-text indexes, make two copies of it, marking one with `xtf:meta` and not marking the other.

- Store Attribute

```
<xsl:attribute name="xtf:store" select="'TrueOrFalse'"/>
```

This attribute is used to turn on/off whether to store the contents of a meta-data field in the index, and make them available to the **Result Formatter** stylesheet.

The value for either of these tags should be set to either the string '`true`' or the string '`false`'. If not specified, this attribute defaults to '`true`'.

This attribute can only be used on meta-data blocks that also have the `xtf:meta` attribute set. Setting xtf:store to '`false`' can make the final index smaller, and can also speed up processing by the **Result Formatter** stylesheet, since it will have less data to process. A field can be indexed and stored, indexed and not stored, or stored and not indexed; all of these combinations can be useful in certain circumstances.

- Tokenize Attribute

```
<xsl:attribute name="xtf:tokenize" select="'YesOrNo'"/>
```

This attribute is used to indicate whether a meta-data field should be tokenized or not. By default, meta-data fields are tokenized so they can be searched. If you intend to use a meta-data

field for sorting query results instead, set this attribute to ‘no‘ .

- Proximity Break Attribute

```
<xsl:attribute name="xtf:proximitybreak" select="'TrueOrFalse'"/>
```

This attribute introduces a proximity break into a document. A tag marked with a proximity break attribute is considered to be infinitely far away from the previous or containing tag. Using this tag prevents proximity matches that span two adjacent tags from being counted as a valid match.

The select value for this tag should be set to either the string ‘true‘ (introduce a proximity break) or the string ‘false‘ (**do not** introduce a proximity break.)

To de-emphasize rather than disallow proximity matches across sections, use the sectionBump attribute instead (see below).

- Sentence Bump Attribute

```
<xsl:attribute name="xtf:sentenceBump" select="BumpInWords"/>
```

This attribute de-emphasizes proximity searches that span multiple sentences by introducing extra virtual spacing between adjacent sentences. The amount of virtual spacing to add between the end of the previous sentence and the beginning of the current one is specified as a number of virtual words by the BumpInWords argument. This value, if not specified, defaults to five words of added spacing.

(Note: If not explicitly set, nested sub-tags for a document inherit the sentence bump value from the closest parent tag for which a sentence bump value is defined.)

- Section Type Attribute

```
<xsl:attribute name="xtf:sectionType" select="'TypeName'"/>
```

This attribute assigns a section type name to a tag, with the TypeName parameter identifying the section name to use. Assigning a section name to a tag allows grouped searches to be performed on tags that have the same section names, by inserting a Section Type Tag into a query.

(Note: If not explicitly set, nested sub-tags for a document inherit the section type name from the closest parent tag for which a section name is defined.)

- Section Type Add Attribute

```
<xsl:attribute name="xtf:sectionTypeAdd" select="'TypeName'"/>
```

This attribute appends a section type name to the section type already associated with a tag (or one of its ancestors which has a section type), with the TypeName parameter identifying the section name to append. Assigning a section name to a tag allows grouped searches to be performed on tags that have the same section names, by inserting a Section Type Tag into a query. And appending a section type allows child tags to inherit their parent’s `sectionType` and then add additional type information. This can be very useful for representing hierarchical information using section types.

(Note: If not explicitly set, nested sub-tags for a document inherit the section type name from the

closest parent tag for which a section name is defined, including any section type which has been appended to that parent tag.)

- Section Bump Attribute

```
<xsl:attribute name="xtf:sectionBump" select="BumpInWords"/>
```

This attribute de-emphasizes proximity searches that span multiple sections by introducing extra virtual spacing between adjacent sections. The amount of virtual spacing added between the end of the previous section and the beginning of the current one is specified as a number of virtual words by the BumpInWords argument. This value, if not specified, defaults to zero words of added spacing.

- Word Boost Attribute

```
<xsl:attribute name="xtf:wordBoost" select="BoostValue"/>
```

This attribute boosts or de-emphasizes the relevance of text found within a particular tag. To boost the relevance of text in a tag, set the BoostValue parameter to a floating-point number greater than 1.0. To de-emphasis the relevance of a tag's text, set the BoostValue parameter to a floating-point number between 0.0 and 1.0.

(Note: If not explicitly set, nested sub-tags for a document inherit the boost value from the closest parent tag for which a boost value is defined.)

crossQuery

return to top

This section details the parameters, input tags, output tags, and utilities used in programming the **crossQuery** servlet.

1. Common to crossQuery and dynaXML

- **Stylesheet parameters**
  This section summarizes the standard set of XSL parameters available to every stylesheet used by the ***crossQuery*** and ***dynaXML*** servlets. These include the original URL of the current request (in a few handy forms), the XTF base directory, and headers from the HTTP request.
  - Request URL
    This field identifies the full URL passed to the XTF system for the current request. The request URL is always available to servlets (unlike many of the following parameters which are optional). It is accessed via the XSL parameter

```
<xsl:param name="http.URL"/>
```

This parameter contains a URL string of the form: `http://`*`yourserver`*`/`*`yourport`*`/servlet/`*`queryparms`*

where

yourserver	is the name of your XTF server
yourport	is the port through which XTF requests are routed (typically 8080)

servlet	is the name of the servlet to which the request is being sent. Normally, this is either `search` (for crossQuery) or `view` (for dynaXML)
queryparms	is the list of parameters that defines the actual request being sent to the servlet. All URL escape codes (such as %20 for space) will have been translated to normal characters, and UTF-8 octet sequences will have been decoded.

- Request URL with Original Unescaped Percent Codes

  This field identifies the full URL passed to the XTF system for the current request, with all of the percent codes left unescaped. The request URL is always available to servlets (unlike many of the following parameters which are optional). It is accessed via the XSL parameter

  ```
  <xsl:param name="http.rawURL"/>
  ```

  This parameter contains a URL string of the form: `http://`*yourserver*`/`*yourport*`/`*servlet*`/`*queryparms*

  where

yourserver	is the name of your XTF server
yourport	is the port through which XTF requests are routed (typically 8080)
servlet	is the name of the servlet to which the request is being sent. Normally, this is either `search` (for crossQuery) or `view` (for dynaXML)
queryparms	is the list of parameters that defines the actual request being sent to the servlet. All URL escape codes (such as %20 for space) will be left unescaped (i.e. **not** translated to normal characters) and UTF-8 octet sequences will **not** have been decoded.

- Servlet Directory

  This field identifies the filesystem path (directory) of the XTF instance in which the stylesheet is running. It is accessed via the XSL parameter

  ```
  <xsl:param name="servlet.dir"/>
  ```

  Typically this value comes from the servlet container (e.g. Resin or Tomcat), but may be overridden by specifying a base-dir parameter in the servlet container configuration. As this varies by container, check the documentation for your servlet container if you wish to override this value.

- Servlet URL

  This field identifies the URL used to access the servlet. It is accessed via the XSL parameter

  ```
  <xsl:param name="servlet.URL"/>
  ```

  This parameter contains a URL string of the form: `http://`*yourserver*`{:`*yourport*`}/xtf/`*servlet*

  where

yourserver	is the name of your XTF server
yourport	is the port through which XTF requests are routed (typically 8080)

servlet	is the name of the servlet to which the request is being sent. Normally, this is either `search` (for crossQuery) or `view` (for dynaXML)

Note that the query string is *not* included in this parameter; if you need to access the query string, use the **Request URL** parameter above.

- Root URL

This field identifies the URL of this particular instance in the servlet container. The value is useful for accessing icons and other non-servlet resources from the container. It is accessed via the XSL parameter

```
<xsl:param name="root.URL"/>
```

This parameter contains a URL string of the form: `http://`*yourserver*`{:`*yourport*`}/xtf/` where

yourserver	is the name of your XTF server
yourport	is the port through which XTF requests are routed (typically 8080)

Note that the servlet name and query string are *not* included in this parameter; if you need to access these, use the **Request URL** parameter or the **Servlet URL** parameter, above.

- User-Agent Header Field

This optional HTTP header field identifies the browser that issued the current request. It is accessed via the XSL parameter

```
<xsl:param name="http.user-agent"/>
```

This parameter contains a string that identifies the browser that made the current request. Most HTTP requests will provide this field. For example:

```
Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt; empas)
```

Note that the contents of this field vary widely depending on which browser made the request, and a detailed description is beyond the scope of this document.

- Referer Field

This optional HTTP header field identifies the web page from which the request URL was issued. It is accessed via the XSL parameter

```
<xsl:param name="http.referer">
```

This parameter holds a URL string identifying the web page that issued the request. Often the initial request to the servlet will supply this parameter; subsequent requests will not.

- If-Modified-Since Header Field

This optional HTTP header field identifies the last time a particular request URL was issued by a browser. This field can be used by an XTF stylesheet to determine if a request needs to be processed because the XTF database has changed, or whether the requesting browser can use its

previously cached results. This field is accessed via the XSL parameter

```
<xsl:param name="http.if-modified-since"/>
```

Most HTTP responses will not include this parameter, but if they do, the contents of this time string of the form:
weekday, dd-mmm-yy hh:mm:ss timezone
where

weekday	is the day of the week the request was last issued
dd-mmm-yy	is the day, three letter month abbreviation, and year the request was last issued
hh:mm:ss	is the time the request was last issued, represented as a 24 hour GMT based time
timezone	is the offset in hours of the timezone from which the request was last issued.

- Other HTTP Headers

  All HTTP header fields are made available to stylesheets in a similar fashion to the fields detailed above. In general they are accessed via XSL parameters like this:

  ```
  <xsl:param name="http.field-name">
  ```

  where field-name is the name of the HTTP header field. Note that most HTTP header fields will be absent most of the time.

- Pass-Through Configuration Parameters

  Any unrecognized tag present in the configuration file for a given tool (i.e. `textIndexer.conf` for the textIndexer, `dynaXML.conf` for the dynaXML, etc.) will be made available to stylesheets run by that tool. These can be accessed by declaring an XSLT parameter containing the element name and the attribute name:

  ```
  <xsl:param name="ElementName.AttributeName/>
  ```

  For a concrete example, see the Programming Guide.

- **Parser Input Tags**

  The following tags make up the XML input for the **Query Router**, **Query Parser**, and **Document Request Parser** stylesheets. They constitute a simple XML representation of the query URL supplied to the crossQuery or dynaXML servlet.

  - Container Tag

    This tag is the outermost tag for the XML input fragment sent to the router or parser stylesheet for translation. It has the form:

    ```
    <parameters>
        ParameterBlock
        ParameterBlock
        …
    </parameters>
    ```

  - Parameter Block Tag

    This tag is the XML input to the parser for a single parameter in a user query URL. It has the

form:

```
<param name="ParamName" value="ParamValue">
    Token | Phrase
    Token | Phrase
    …
</param>
```

where

`name="ParamName"`	is the name of the parameter extracted from the original query URL.
`value="ParamValue"`	is the original text in the query URL that is assigned to the specified parameter.

Note that each of the parameters from the query URL is also available as a standard XSL parameter with the form:

```
<xsl:param name="ParamName" select="DefaultValueIfNotInURL"/>
```

This allows query parameters to be accessed either through the standard template driven XML or through stylesheet parameters.

- Token Tag

  This tag identifies a single word or token taken from the query URL. It has the form:

  ```
  <token value="Word" isWord="YesOrNo"/>
  ```

  where

`value="Word"`	is the actual word or symbol extracted from the URL.
`isWord="YesOrNo"`	identifies whether the token is a word (`isWord="yes"`) or a punctuation symbol (`isWord="no"`.)

- Phrase Tag

  This tag identifies a literal phrase taken from the query URL. It has the form:

  ```
  <phrase value="StringOfWords">
      Token
      Token
      …
  </phrase>
  ```

  where

`value="StringOfWords"`	is the entire phrase extracted from the URL as a single string.
Token, Token…	is the original phrase broken down into one or more **Token Tags** (see above), one for each word or symbol in the phrase.)

- **Error Generator Stylesheet**

  The purpose of the **Error Generator** stylesheet is to generate a web-page that displays user friendly messages when crossQuery or ***dynaXML*** errors occur. Since its output is simply HTML, the only tags that this reference section covers are the input XML tags passed to the **Error Generator** by the

servlets.For convenience, all of the error information passed into the **Error Generator** stylesheet as XML tags is also available in the following XSLT parameters:

`$exception`	A string containing the name of the exception that occurred. This will be the name of one of the error/exception tags listed below (e.g., `ExcessiveWork`, `TermLimit`, etc.)
`$message`	The descriptive message for the error/exception (if any; may be an empty string.)
`$stackTrace`	The HTML-Formatted Java Stack Trace generated by the exception, (if any; may be an empty string.)

- *Query Format Error Tag*

  This error tag is issued by the **Query Parser** or **Document Request Parser** stylesheet if its code determines that there is an error in the query URL received from the user. It has the form:

```
<QueryFormat>
    <message>Error Message</message>
</QueryFormat>
```

  To generate such an error, the **Query Parser** simply issues a

```
<error message="Error Message"/>
```

  tag instead of a query tag. The error message specified by the parser's error tag is then transferred to the **Query Format Error Tag**, for processing by the servlet's **Error Generator** stylesheet.

- Term Limit Error Tag

  This error tag is issued by the **Query Parser** or **Document Request Parser** stylesheet if the number terms or clauses in a query exceeds the maximum term limit established by the **termLimit** attribute of the query tag produced by the Query Parser. It has the form:

```
<TermLimit>
    <message>Error Message</message>
</TermLimit>
```

  This error is most often generated by the expansion of wildcard and range queries.

- Excessive Work Error Tag

  This error tag is issued by the **Query Parser** or **Document Request Parser** stylesheet if a query exceeds the maximum work limit established by the **workLimit** attribute of the query tag produced by the Query Parser. It has the form:

```
<ExcessiveWork>
    <message>Error Message</message>
</ExcessiveWork>
```

- Invalid Document Error Tag

  This error tag is issued by the dynaXML servlet if a document requested by the **Document Request Parser** stylesheet (in the `<source>` tag) cannot be located. This tag has the form:

```
<InvalidDocument>
    <message>Error Message</message>
```

```
    <docId>Document Identifier</message>
</InvalidDocument>
```

Usually, this error is generated when a request is made for a document that doesn't exist (for instance, it might have been removed, or the document ID might be invalid.)

- No Permission Error Tag

This error tag is issued by the dynaXML servlet if the authentication (specified by the **Document Request Parser** stylesheet) fails for some reason. For instance, if IP-based authentication has been specified, and the requestor's IP address isn't in the IP list, this error will be issued. This tag has the form:

```
<NoPermission>
    <message>Error Message</message>
    <ipAddr>IP Address</message>
</NoPermission>
```

Note that the IP address will only be included for requests that failed during IP-list authentication (and not for LDAP or external authentication, for example.)

- Unsupported Query Error Tag

This error tag is issued by the dynaXML servlet if a document request issued by the **Document Request Parser** stylesheet specifies a text query, and the document being queried is not present in the index. Searching can only be performed on documents that have been indexed. This tag has the form:

```
<UnsupportedQuery>
    <message>Error Message</message>
</UnsupportedQuery>
```

Usually, this error is generated when a query tag has been used in a document request without the required index tag.

- General Exception Error Tag

This error tag is generated whenever an internal exception occurs in any Servlet. This tag looks as follows:

```
<GeneralExceptionName>
    <message>Error Message</message>
    <stackTrace>HTML-Formatted Java Stack Trace</stackTrace>
</GeneralExceptionName>
```

Exceptions are usually generated by anomalous fatal conditions like missing required files, corrupted indexes, files locked by other applications, or bugs in the XTF code itself.

- **Utilities**

The following utilities are available to ***dynaXML*** and ***crossQuery*** stylesheets:

- Raw Dump Mode

The `raw` URL parameter can be helpful when debugging a **Search Result Formatter** or **Document Hit Formatter** stylesheet. When specified, the servlet (either crossQuery or

dynaXML) will dump the formatter input directly to the browser, bypassing the formatter stylesheet. The raw dump can help in formulating templates to correctly format the final output.

This parameter is added to a query URL, and has the form:
`http://`*yourserver{*`:`*yourport}*`/xtf/search?`*queryParameters*`;`**`raw=`***YesOrNo*

or
`http://`*yourserver{*`:`*yourport}*`/xtf/view?`*viewParameters*`;`**`raw=`***YesOrNo*

where

YesOrNo	specifies whether or not raw XML search results are sent to your browser. If set to **yes** , the formatter stylesheet is disabled for the query, and raw XML search results or marked up document contents are sent to your browser. If set to **no** , the XML is sent to the formatter stylesheet for processing, and its output in turn is sent to the browser. If this URL parameter is not specified, it defaults to no.

- Session State

XTF is capable of tracking variables associated with a particular browsing session. This section details the functions available to any crossQuery or dynaXML stylesheet for storing and retrieving session data.

- Store Session Data FunctionThis function can be called within a stylesheet to set a name/value pair in the session data, like this:

```
<xsl:value-of select="session:setData(Name, Value)"/>
```

where

`session:`	is a namespace prefix to differentiate this function from any other. The namespace URI for this prefix must be: **java:org.cdlib.xtf.xslt.Session**.
Name	specifies the name under which to store the value in the session data.
Value	specifies the particular value to store. It may be either a string, or a structured piece of XML with a single outer-level element (and any number of inner elements.)

This function searches the session data for a value with a matching name. If found, the old value is replaced with the specified one. If not found, a new name/value pair is added to the session data.

Note that session tracking must be enabled in the servlet configuration file before calling session:setData; if it not, a runtime error will be generated and processing will stop. By default session tracking is enabled; see the XTF Deployment Guide for more information on enabling or disabling session tracking.

Although this function is typically called within an `<xsl:value-of>` element, it does not return a value. The purpose of using the `<xsl:value-of>` element is to force the XSLT processor to execute the function instead of possibly optimizing out the call.

- Retrieve Session Data Function

This function can be called within a stylesheet to retrieve a value from the session data,

like this:

```
<xsl:variable name="Variable" select="session:getData(Name)"/>
```

where

Variable	specifies the name of an XSLT variable to create.
`session:`	is a namespace prefix to differentiate this function from any other. The namespace URI for this prefix must be: **`java:org.cdlib.xtf.xslt.Session`**.
Name	specifies the name to look up in the session data.

This function searches the session data for a value with a matching name and returns the value (in this case, assigning it to the given XSLT variable.) If not found, `null` is returned (i.e. empty string/empty sequence.)

Note that if session tracking isn't enabled, `null` is always returned. By default session tracking is enabled; see the XTF Deployment Guide for more information on enabling or disabling session tracking.

- Get Session Identifier

  This function can be called within a stylesheet to obtain the current session identifier string, like this:

  ```
  <xsl:variable name="sessionID" select="session:getID()"/>
  ```

  where

`session:`	is a namespace prefix to differentiate this function from any other. The namespace URI for this prefix must be: **`java:org.cdlib.xtf.xslt.Session`**.

  This function returns the unique identifier assigned by the servlet container (e.g. Resin or Tomcat) to the current user session.

- Check Enabled Status FunctionThis function can be called within a stylesheet to check whether session tracking is enabled, like this:

  ```
  <xsl:if test="session:isEnabled()"/>
  ```

  where

`session:`	is a namespace prefix to differentiate this function from any other. The namespace URI for this prefix must be: **`java:org.cdlib.xtf.xslt.Session`**.

  This function returns `true` if session tracking is enabled in the configuration file for the current servlet. It returns `false` if session tracking is not enabled. By default session tracking is enabled; see the XTF Deployment Guide for more information on enabling or disabling session tracking.

- Encode URL Function

  This function can be called within a stylesheet to add the session ID to any URL created by the stylesheet, like this:

  ```
  <xsl:variable name="Variable" select="session:encodeURL(RawURL)"/>
  ```

  where

Variable	specifies the name of an XSLT variable to create. This variable will contain the new, encoded, URL.
`session:`	is a namespace prefix to differentiate this function from any other. The namespace URI for this prefix must be: **`java:org.cdlib.xtf.xslt.Session`**.
RawURL	specifies an expression or other variable containing the URL to be encoded.

  This function adds a session ID to the specified raw URL, if necessary. Generally session IDs are stored in cookies, but if the user has disabled cookies, URLs will have session IDs added to them. Note that if session tracking is disabled, the URL will be returned unchanged.

- session:noCookie()

  This extension function is used to determine if cookies are enabled in the user's browser.

  ```
  <xsl:if test="session:noCookie()">
      <a href="javascript:alert('Cookies are disabled!')">
      Requires Cookie!
      </a>
  </xsl:if>
  ```

  where

`session:`	is a namespace prefix to differentiate this function from any other. The namespace URI for this prefix must be: **`java:org.cdlib.xtf.xslt.Session`**.

- Calling Command-line Tools

  The built-in `exec` extension element can be used in stylesheets to call external command-line tools. See the Programming Guide for a description and examples.

  External Command Extension Element

  This element can be used inside any XTF stylesheet to call a command-line program, like this:

  ```
  <exec:run command  = "CommandName"
            {timeout = "Milliseconds"}
            xsl:extension-element-prefixes="exec">

      {<exec:arg>Argument1</exec:arg>}
      {<exec:arg>Argument2</exec:arg>}
      …

      {<exec:input>
           XmlOrString
  ```

```
    </exec:input>}

</exec:run>
```

where

`exec:`	is a namespace prefix identifying this particular Saxon extension. The namespace URI for this prefix must be: **`java:/org.cdlib.xtf.saxonExt.Exec`**.
`command="CommandName"`	specifies the command-line program to run. In general, this should be an absolute path; if a relative path is given, it will be resolved in an undefined manner.
`timeout="Milliseconds"`	is an optional attribute setting an upper limit, in milliseconds, on the amount of time the external process will be given to finish its work. If this time is exceeded, the process will be forcibly terminated, an a Java exception will be thrown (which terminates stylesheet processing immediately.) If this attribute is not specified, the process will be allowed to run to completion no matter how long it takes.
`<exec:arg>`	is an optional sub-element that can be used repeatedly to specify command-line arguments to be passed to the program. Each argument should be specified in its own `<exec:arg>` element; in particular, pairs should usually be broken up. For example, to perform the command "**`ls -al *`**" one would specify two arguments, the first being "**`-al`**" and the second being "**`*`**".
`<exec:input>`	is an optional sub-element that specifies the input to be send to the external program (input will be sent to the process stdio stream). If the content of the exec:input element is XML data (for instance, a variable holding one or more elements) then the servlet will automatically serialize the data into standard XML format, with UTF-8 character encoding. If the content is not XML, the string will be sent to the tool verbatim.

This extension element calls an external command-line tool. Internally, this creates a new operating system process, and the servlet thread waits for that process to complete.
If the exec:input element is specified, the entire string of input text (converted from XML to text if necessary) will be fed to the process' stdin stream.

Output from the process' stdout stream will be collected, and the servlet checks if the data begins with an XML header. If so, the value of the exec:run element will be actual XML elements from the data. Otherwise, the value will be one long string, being an exact copy of the output from the external tool.

If the process exits with a non-zero code (or the optional timeout is exceeded) then XTF will throw a Java exception, which causes the stylesheet execution to terminate immediately.

- File Utility Functions

XTF provides a number of functions that stylesheets can use to check attributes of a file in the local filesystem. See the XTF Programming Guide for a description and examples.

- File Existence Extension Function

  This extension function can be used within a stylesheet to determine whether a particular file exists on the filesystem, like this:

```
<xsl:if test="FileUtils:exists(FilePath)">
    …
</xsl:if>
```

  where

`FileUtils:`	is a namespace prefix identifying this set of Saxon extension functions. The namespace URI for this prefix must be: **`java:org.cdlib.xtf.xslt.FileUtils`**.
FilePath	specifies the relative or absolute path to the file in question. If a relative path is specified, it will be resolved in relation to the stylesheet that calls the function.

  The extension function returns the boolean value true if the file is readable by the stylesheet, or false if not.

- File Length Function

  This extension function can be used within a stylesheet to determine the length (in bytes) of a particular file exists on the filesystem, like this:

```
<xsl:variable name="myFileLen" select="FileUtils:length(FilePath)">
    …
</xsl:variable>
```

  where

`FileUtils:`	is a namespace prefix identifying this set of Saxon extension functions. The namespace URI for this prefix must be: **`java:org.cdlib.xtf.xslt.FileUtils`**.
FilePath	specifies the relative or absolute path to the file in question. If a relative path is specified, it will be resolved in relation to the stylesheet that calls the function.

  The extension function returns the length, in bytes, of the specified file, or -1 if the file cannot be read.

- File Modification Time Function

  This extension function can be used within a stylesheet to find out when a particular file was last modified, like this:

```
<xsl:variable name="myModTime"
              select="FileUtils:lastModified(FilePath, DateFormat)">
    …
</xsl:variable>
```

  where

`FileUtils:`	is a namespace prefix identifying this set of Saxon extension functions. The namespace URI for this prefix must be: **`java:org.cdlib.xtf.xslt.FileUtils`**.
FilePath	specifies the relative or absolute path to the file in question. If a relative path is specified, it will be resolved in relation to the stylesheet that calls the function.
DateFormat	is a string representing the format that the date and/or time should be returned in. This uses codes from Java's SimpleDateFormat class such as `yyyy-MM-dd:HH:mm:ss`. **Warning**: "mm" and "MM" are different: the former is minutes, the latter is months.

The extension function returns a formatted version of the date and/or time (depending on format) that a given file was last modified. This can be useful for detecting which files are newer than others, processing all files newer than a certain date, etc.

- Current Date & Time Function

  This extension function can be used within a stylesheet to find out the current date and time known to the system, like this:

```
<xsl:variable name="myTime"
              select="FileUtils:curDateTime(DateFormat)">
    …
</xsl:variable>
```

  where

`FileUtils:`	is a namespace prefix identifying this set of Saxon extension functions. The namespace URI for this prefix must be: **`java:org.cdlib.xtf.xslt.FileUtils`**.
DateFormat	is a string representing the format that the date and/or time should be returned in. This uses codes from Java's SimpleDateFormat class such as `yyyy-MM-dd:HH:mm:ss`. **Warning**: "mm" and "MM" are different: the former is minutes, the latter is months.

  The extension function returns a formatted version of the current date and/or time (depending on format). This can be useful for constructing time-based queries, displaying the time or date on a result page, etc.

- readXMLStub

  This extension function provides an efficient way to determine the root element name, public ID, DTD URI, and namespace of the source document.

```
<xsl:variable name="xmlStub"
              select="FileUtils:readXMLStub(FilePath)">
    …
</xsl:variable>
```

  where

`FileUtils:`	is a namespace prefix identifying this set of Saxon extension functions. The

	namespace URI for this prefix must be: **`java:org.cdlib.xtf.xslt.FileUtils`**.
FilePath	specifies the relative or absolute path to the file in question. If a relative path is specified, it will be resolved in relation to the stylesheet that calls the function.

This can be very useful when trying to determine the type of document you are dealing with and which stylesheets to apply to it. *xsl:for-each* must be used to correctly set the context node for the functions used in creating the variables.

```
<xsl:for-each select="FileUtils:readXMLStub($file)">
    <xsl:variable name="rn" select="name(*[1])"/>
    <xsl:variable name="pid" select="unparsed-entity-public-id($rn)"/>
    <xsl:variable name="uri" select="unparsed-entity-uri($rn)"/>
    <xsl:variable name="ns" select="namespace-uri(*[1])"/>
    <xsl:if test="matches($rn,'^TEI') or
                  matches($pid,'TEI') or
                  matches($uri,'tei2\.dtd') or
                  matches($ns,'tei')">
        This must be a TEI document!...
    </xsl:if>
    …
<xsl:for-each>
```

In the example stylesheets you can see it in action in both the docSelector and docReqParser.

- *Redirecting to Another URL*

XTF provides an extension element that immediately redirects the user's browser to a different URL, suppressing further processing of the formatting stylesheet. See the XTF Programming Guide for a description and example.

HTTP Redirection Extension

This element can be used inside any parsing or formatting stylesheet in XTF like this:

```
<redirect:send url="TargetURL"
               xmlns:redirect="java:/org.cdlib.xtf.saxonExt.Redirect"
               xsl:extension-element-prefixes="redirect"/>
```

where

`url`="TargetURL"	is a required attribute specifying the URL that the user's browser should be redirected to. An absolute or relative URL may be specified. If relative, the URL will be resolved by the servlet container to an absolute URL.
`redirect:`	is a namespace prefix identifying this Saxon extension instruction. The namespace URI for this prefix must be: **`java:/org.cdlib.xtf.saxonExt.Redirect`** . In addition, it must be declared in the list of extension-element-prefixes for Saxon. The declarations are most easily done in-line as shown above.

This extension element causes an immediate HTTP redirect (code 302) to be sent to the user's browser. Further processing of the stylesheet is aborted.

No prior output is allowed before this extension instruction is executed. If any output was generated, an exception will be thrown and the redirect will fail.

- *TBD: Dynamic E-mail*
- *TBD: Tidying HTML Files*

Query Router and Query Parser Tags

- Common Input Tags and Parameters
  - Common Parser Input Tags

    The following tags make up the XML input for the Query Router, Query Parser, and Document Request Parser stylesheets. They constitute a simple XML representation of the query URL supplied to the crossQuery or dynaXML servlet.

    - Container TagThis tag is the outermost tag for the XML input fragment sent to the router or parser stylesheet for translation. It has the form:

```
<parameters>
    ParameterBlock
    ParameterBlock
    …
</parameters>
```

    - Parameter Block TagThis tag is the XML input to the parser for a single parameter in a user query URL. It has the form:

```
<param name="ParamName" value="ParamValue">
    Token | Phrase
    Token | Phrase
    …
</param>
```

      where

`name="ParamName"`	is the name of the parameter extracted from the original query URL.
`value="ParamValue"`	is the original text in the query URL that is assigned to the specified parameter.

      Note that each of the parameters from the query URL is also available as a standard XSL parameter with the form:

```
<xsl:param name="ParamName" select="DefaultValueIfNotInURL"/>
```

      This allows query parameters to be accessed either through the standard template driven XML or through stylesheet parameters.

    - Token Tag

      This tag identifies a single word or token taken from the query URL. It has the form:

```
<token value="Word" isWord="YesOrNo"/>
```

      where

`value="Word"`	is the actual word or symbol extracted from the URL.
`isWord="YesOrNo"`	identifies whether the token is a word (`isWord="yes"`) or a punctuation symbol (`isWord="no"`.)

- Phrase Tag

  This tag identifies a literal phrase taken from the query URL. It has the form:

```
<phrase value="StringOfWords">
    Token
    Token
    …
</phrase>
```

  where

`value="StringOfWords"`	is the entire phrase extracted from the URL as a single string.
Token, Token…	is the original phrase broken down into one or more **Token Tags** (see above), one for each word or symbol in the phrase.)

- *Common Stylesheet Parameters*

This section summarizes the standard set of XSL parameters available to every stylesheet used by the crossQuery and dynaXML servlets. These include the original URL of the current request (in a few handy forms), the XTF base directory, and headers from the HTTP request.

- Request URL

  This field identifies the full URL passed to the XTF system for the current request. The request URL is always available to servlets (unlike many of the following parameters which are optional). It is accessed via the XSL parameter

```
<xsl:param name="http.URL"/>
```

  This parameter contains a URL string of the form: `http://`*`yourserver/yourport/servlet/queryparms`*

  where

yourserver	is the name of your XTF server
yourport	is the port through which XTF requests are routed (typically 8080)
servlet	is the name of the servlet to which the request is being sent. Normally, this is either `search` (for crossQuery) or `view` (for dynaXML)
queryparms	is the list of parameters that defines the actual request being sent to the servlet. All URL escape codes (such as %20 for space) will have been translated to normal characters, and UTF-8 octet sequences will have been decoded.

- Servlet Directory

  This field identifies the filesystem path (directory) of the XTF instance in which the stylesheet is running. It is accessed via the XSL parameter

```
<xsl:param name="servlet.dir"/>
```

Typically this value comes from the servlet container (e.g. Resin or Tomcat), but may be overridden by specifying a base-dir parameter in the servlet container configuration. As this varies by container, check the documentation for your servlet container if you wish to override this value.

- Servlet URL

  This field identifies the URL used to access the servlet. It is accessed via the XSL parameter

  ```
  <xsl:param name="servlet.URL"/>
  ```

  This parameter contains a URL string of the form: `http://`*yourserver{*`:`*yourport}* `/xtf/servlet`

  where

yourserver	is the name of your XTF server
yourport	is the port through which XTF requests are routed (typically 8080)
servlet	is the name of the servlet to which the request is being sent. Normally, this is either `search` (for crossQuery) or `view` (for dynaXML)

  Note that the query string is *not* included in this parameter; if you need to access the query string, use the **Request URL** parameter above.

- Root URL

  This field identifies the URL of this particular instance in the servlet container. The value is useful for accessing icons and other non-servlet resources from the container. It is accessed via the XSL parameter

  ```
  <xsl:param name="root.URL"/>
  ```

  This parameter contains a URL string of the form: `http://`*yourserver{*`:`*yourport}*`/xtf/` where

yourserver	is the name of your XTF server
yourport	is the port through which XTF requests are routed (typically 8080)

  Note that the servlet name and query string are *not* included in this parameter; if you need to access these, use the **Request URL** parameter or the **Servlet URL** parameter, above.

- User-Agent Header FieldThis optional HTTP header field identifies the browser that issued the current request. It is accessed via the XSL parameter

  ```
  <xsl:param name="http.user-agent"/>
  ```

  This parameter contains a string that identifies the browser that made the current request. Most HTTP requests will provide this field. For example:

  ```
  Mozilla/4.0 (compatible; MSIE 5.0; Windows 98; DigExt; empas)
  ```

Note that the contents of this field vary widely depending on which browser made the request, and a detailed description is beyond the scope of this document.

- Referer FieldThis optional HTTP header field identifies the web page from which the request URL was issued. It is accessed via the XSL parameter

```
<xsl:param name="http.referer">
```

This parameter holds a URL string identifying the web page that issued the request. Often the initial request to the servlet will supply this parameter; subsequent requests will not.

- If-Modified-Since Header FieldThis optional HTTP header field identifies the last time a particular request URL was issued by a browser. This field can be used by an XTF stylesheet to determine if a request needs to be processed because the XTF database has changed, or whether the requesting browser can use its previously cached results. This field is accessed via the XSL parameter

```
<xsl:param name="http.if-modified-since"/>
```

Most HTTP responses will not include this parameter, but if they do, the contents of this time string of the form:
weekday, dd-mmm-yy hh:mm:ss timezone
where

weekday	is the day of the week the request was last issued
dd-mmm-yy	is the day, three letter month abbreviation, and year the request was last issued
hh:mm:ss	is the time the request was last issued, represented as a 24 hour GMT based time
timezone	is the offset in hours of the timezone from which the request was last issued.

- Other HTTP Headers

All HTTP header fields are made available to stylesheets in a similar fashion to the fields detailed above. In general they are accessed via XSL parameters like this:

```
<xsl:param name="http.field-name">
```

where field-name is the name of the HTTP header field. Note that most HTTP header fields will be absent most of the time.

- Pass-Through Configuration ParametersAny unrecognized tag present in the configuration file for a given tool (i.e. `textIndexer.conf` for the textIndexer, `dynaXML.conf` for the dynaXML, etc.) will be made available to stylesheets run by that tool. These can be accessed by declaring an XSLT parameter containing the element name and the attribute name:

```
<xsl:param name="ElementName.AttributeName/>
```

For a concrete example, see the Programming Guide.

- *Query Router tags*

The tags listed here are XML tags associated with the Query Router stylesheet. The tags are divided

into two categories: input tags (here) and then output tags. The input tags are used by the crossQuery Servlet to translate the original query URL into a simple XML representation that can act as input to the Query Router stylesheet. The output tags are then used by the Query Router stylesheet to form the actual XML query passed to the crossQuery servlet's Text Search Engine for processing.

The XML input to the Query Router stylesheet is a simple XML representation of the query URL supplied to the crossQuery servlet from the user query web-page. The XML input and XSL parameters are the same as those for the Query Parser and Document Request Parser.

- *Query Router Output Tags*

  These tags are used by the Query Router stylesheet to form the XML response telling the crossQuery servlet the particular query parser stylesheet to use. The top-level tag is the Route Tag below.

  - Route Tag

    This tag is outermost tag in the query route sent to the crossQuery servlet's query routing logic. It has the form:

    ```
    <route>

        QueryParserTag
        {ErrorGenTag}

    </route>
    ```

    The **QueryParserTag** (see below) specified within this tag identifies the Query Parser stylesheet to use. If specified, the **ErrorGenTag** (see below) identifies an Error Generator stylesheet to use instead of the default specified in the crossQuery.conf file.

  - QueryParser Tag

    This tag appears directly within a **Route Tag** (see above). It has the form:

    ```
    <queryParser path="QueryParserLocation"/>
    ```

    This tag identifies which Query Parser stylesheet should be utilized by crossQuery to parse the URL parameters and produce an XTF query. If this path is not specified as an absolute path, it is assumed to be relative to the XTF base installation directory (i.e., XTF_HOME.)

  - ErrorGen Tag

    This tag appears directly within a **Route Tag** (see above). It has the form:

    ```
    <errorGen path="ErrorGeneratorLocation"/>
    ```

    This tag identifies which stylesheet should be used in case unexpected errors occur during query parsing or processing. This overrides the default error generator specified in the crossQuery.conf configuration file. If this path is not specified as an absolute path, it is assumed to be relative to the XTF base installation directory (i.e., XTF_HOME.)

- Query Parser Output Tags

The tags listed here are XML tags associated with the Query Parser stylesheet. The tags are divided

into two categories: output tags (on this page) and input tags. The input tags are used by the crossQuery Servlet to translate the original query URL into a simple XML representation that can act as input to the Query Parser stylesheet.The following output tags are used by the Query Parser stylesheet to form the XML query passed to the crossQuery search engine for processing. Since there are so many, each tag has its own page. The top-level tag is the <query> tag below.

<query>
<error>
<term>
<phrase>
<exact>

<and>
<or>
<orNear>
<not>
<near>

<range>
<sectionType>
<facet>
<spellcheck>
<resultData>

<moreLike>
<allDocs>

- Query Tag: <query>

This tag is outermost tag in an XML query sent to the crossQuery servlet's search engine. It has the form:

```
<query indexPath        = "IndexDBLocation"
       style            = "ResultFormatterLocation"
       {sortDocsBy      = "ListOfMetaFields|score"}
       {startDoc        = "FirstDocToReturn"}
       {maxDocs         = "MaxDocsToReturn"}
       {termLimit       = "MaxTermsToAllow"}
       {workLimit       = "MaxWorkToAllow"}
       {maxContext      = "MaxContextChars"}
       {maxSnippets     = "SnippetsToOutput"}
       {termMode        = "TermMarkMode"}
       {field           = "FieldToSearch"}
       {normalizeScores = "TrueOrFalse"}
       {explainScores   = "TrueOrFalse"}>

       QueryElement

</query>
```

where

`indexPath="`*IndexDBLocation*`"`	is the path to the **Index Database** to use when performing the search. If this path is not specified as an absolute path, it is assumed to be relative to the

	XTF base installation directory (i.e., XTF_HOME.)
`style="`*ResultFormatterLocation*`"`	is the path to the **Result Formatter** stylesheet to use to display the results generated by the current query. If this path is not specified as an absolute path, it is assumed to be relative to the XTF base installation directory (i.e., XTF_HOME.)
`sortDocsBy="`*ListOfMetaFields*`|`*score*`"`	is an optional attribute specifying a list of meta fields by which to sort the results. The list should consist of a quoted string containing one or more meta-field names, separated by commas. If multiple meta-fields are specified, the results are sorted first by the left-most meta-field, then sub-sorted by subsequent fields to produce the final output. Optionally, each meta-field name can be preceded by a plus sign (+) or a minus sign (-) to indicate whether the results for that field should be sorted in ascending or descending order. If no plus or minus sign is specified for a meta-field, then the results are sorted in ascending order by default. If this attribute not specified, documents are by default sorted in order of decreasing score (so the most "relevant" documents are first.) With XTF 3.0, this default behavior can now also be explicitly set by providing a value of "score" (or synonym "relevance"). (Note: Meta tags to be used for sorting queries should also have an xtf:tokenize="no" attribute set, or sorting will produce unpredictable results.) (Compatibility note: This attribute was previously called "sortMetaFields", and this old name is still accepted to retain backward compatibility.)
`startDoc="`*FirstDocToReturn*`"`	is an optional attribute specifying the ordinal number of the first matching document to pass on to the **Result Formatter**. If not specified, this attribute defaults to 1, meaning the first document that contains matches for the specified query.
`maxDocs="`*MaxDocsToReturn*`"`	is an optional attribute specifying the number of matching documents to pass on to the **Result Formatter**. If not specified, this attribute defaults to 10, meaning that up to 10 documents with matches will be returned for the specified query. The special value "**`all`**" may be used to indicate that *all* matching documents should be returned.
`termLimit="`*MaxTermsToAllow*`"`	is an optional attribute that limits the number of terms permitted in a query. If not specified, this attribute defaults to 50. This attribute is used primarily to prevent wildcard expansions like `<term>a*</term>` from overloading the crossQuery servlet. If the query does in fact exceed the limit specified by this attribute, a TermLimit error is sent to the **Error Generator**

	stylesheet for the offending query.
`workLimit="`*MaxWorkToAllow*`"`	is an optional attribute that limits the amount of "work" that may be performed in a query. If not specified, this attribute defaults to -1, meaning no limit is enforced. This attribute is used primarily to prevent queries from overloading the crossQuery servlet, which would adversely impact the responsiveness of the XTF system. If a query exceeds the work limit set by this attribute, a ExcessiveWork error is sent to the **Error Generator** stylesheet for the offending query. For the crossQuery servlet, one unit of "work" is equivalent to finding a single matching term in a single document. Experimentally, a value of 500,000 for this attribute seems to work well.
`maxContext="`*MaxContextChars*`"`	identifies the size of a snippet to pass in the **Result Formatter** snippet tag. If not specified, this attribute defaults to 80 characters. Note that the context length is the total number of characters for the snippet, which includes both the matched text and the context text surrounding it.
`maxSnippets="`*SnippetsToOutput*`"`	identifies the number of snippets to pass on to the **Result Formatter** stylesheet for display. A snippet is defined as the matching text found in a document for a particular query, along with some additional text around it for context. The amount of context displayed for each match is defined by the maxContext attribute. If not specified this attribute defaults to 3, meaning snippets for the top three matches for a document are returned. Note that -1 is a special value. If maxSnippets is set to -1, it requests that *all* snippets for a document be returned. If the maxSnippets attribute is set by the `<query>` tag, any occurrences of the maxSnippets attribute set by the inner tags must match the value set by the `<query>` tag. Otherwise, an error will be generated. (Note: Allowing nested copies of the maxSnippets attribute doesn't serve a purpose other than to make it easier to write **docReqParser.xsl** stylesheets in a uniform way. Effectively, the outermost maxSnippets value is always used.)
`termMode="`*TermMarkMode*`"`	is an optional parameter that specifies how terms should be marked in the search results. Valid values for this option are: **`"none"`**: Do not mark matching terms anywhere in the search results. **`"hits"`**: Mark matching terms in the search results only if they appear inside `<hit>` tags. **`"context"`**: Mark matching terms in the search results only if they appear inside `<snippet>` or `<hit>` tags. **`"all"`**: Mark matching terms in search results anywhere they occur. If not

	specified, the default value used is “hits”. Note that when term marking is enabled, matching terms in the search results are placed inside `<term>` … `</term>` tag sets.
`field="`*FieldToSearch*`"`	is an optional parameter that identifies which field in the index to search. This can be set to `text` to indicate that the main text of the document should be searched, or it can name a meta-data field such as `creator` or `subject`. If child elements specify their own `field` attributes, their field name must agree with the parent element. (Note: Allowing nested copies of the field attribute doesn’t serve a purpose other than to make it easier to write **docReqParser.xsl** stylesheets in a uniform way. Effectively, the outermost field name is always used.)
`normalizeScores="`*TrueOrFalse*`"`	is an optional parameter that can disable *score normalization*, the process that converts all document scores to be relative to the highest ranking document (which receives a score of 100). If set to no or false, the scores will be raw floating point numbers. The default, true or yes, will normalize scores to the range of 0..100, and round them to whole numbers. In the default XTF stylesheets, one can simply add “`;normalizeScores=1`” to the query URL, and the default **Query Parser** will set this attribute for you.
`explainScores="`*TrueOrFalse*`"`	is an optional parameter that causes XTF to output a structured, detailed explanation of how the score for each document was calculated. This means that each Document Hit Tag in the query result will contain an Score Explanation Tag, which in turn contains other Score Explanation Tags describing how the components of that score. In the default XTF stylesheets, one can simply add “`;explainScores=1`” to the query URL, and the default **Query Parser** will set this attribute for you. This is an advanced feature, as XTF’s scoring is fairly complex and can be confusing to those just starting out. For an overview of how XTF scores document hits, see the Scoring section of the document XTF Under the Hood. Note that if you enable this attribute, you should generally disable normalizeScores above, as the score explanations describe the non-normalized score for each document.

The primary purpose of the startDoc and maxDocs attributes is to allow search results to be split up into multiple pages by the **Result Formatter**. For example, to display the second page of 50 matching documents, the following query tag could be generated by a link on the current result page:

```
<query … startDoc="51" maxDocs="50">
```

```
                ...
</query>
```

The QueryElement specified within this tag identifies the query to be performed. This can be a term query tag, or a phrase, exact, and, or, orNear, near, range, resultData, spellcheck, or not tag.

Note that the **Query Parser** stylesheet can issue a single top-level error tag instead of a query tag if it encounters any errors.

- Error Tag: <error>

This tag can be issued by the **Query Parser** when it encounters an error. It has the form:

```
<error message="Error Message"/>
```

where Error Message is an error string describing the error.

This tag is a top level tag, and should be issued by itself in place of the normal query tag when an error occurs. Once issued, the error will be routed to the **Error Generator** stylesheet for processing.

- Term Tag: <term>

This tag specifies a single word to search for. This tag has the form:

```
<term {field       = "FieldName"}
      {maxSnippets = "SnippetsToOutput"}
      {boost       = "BoostValue"}>

    WordToFind

    {OptionalSectionTypeQuery}

</term>
```

where

`field=“FieldName”`	is an optional attribute that identifies which field in the index to search. Often this attribute is set to text to indicate that the main text of the document should be searched. It can also be set to the name of a meta field such as author or subject. It should be mentioned that the field name specified by a `<term>` tag must match the field name set by any tags that contain it. Otherwise, an error will be generated.
`maxSnippets=“SnippetsToOutput”`	is an optional attribute that identifies the number of snippets to pass on to the **Result Formatter** stylesheet for display. A snippet is defined as the matching text found in a document for a particular query, along with some additional text around it for context. The amount of context displayed for each match is defined by the maxContext attribute. If not specified this attribute defaults to 3, meaning snippets for the top three matches for a document are returned. Also, this attribute can be set to -1, meaning all the snippets

	for a document are returned. As with the field attribute, the maxSnippets specified by a <term> tag must match the value set by any tags that contain it. Otherwise, an error will be generated.
boost="BoostValue"	is an optional attribute that specifies a relevance boost multiplier for this term in the query. Boost values higher than 1.0 increase the relevance of a term, while boost values between 0.0 and 1.0 decrease the relevance of a term. Boost values less than zero will generate an error.

Note that aside from letters and numbers, only a few select symbols may appear in WordToFind (e.g., apostrophes, periods as part of decimal numbers, etc.) If any other symbols appear in the word to find, no matches will be found in the index. This is because the indexer removes all other types symbols from a term before indexing it. For a complete list of symbols that may appear in a term, see the XTF Under the Hood guide.

Also, the term tag is case-insensitive, so matches with capitalization different from that given in WordToFind may appear in the resulting list of matches.

(Note: Any non-leading or trailing whitespace characters in WordToFind (i.e, space, tab, carriage-return, linefeed, etc.) will produce unpredictable search results.)

If the field is "text", a Section Type sub-query may optionally appear, restricting this term query to particular sections of a document based on section types added by the **Pre-Filter** stylesheet at index time.

- Phrase Search Tag: <phrase>

This tag specifies a phrase for which to search. This tag has the form:

```
<phrase {field       = "FieldName"}
        {maxSnippets = "SnippetsToOutput"}
        {boost       = "BoostValue"}>

    Term | Clause
    Term | Clause
    …

    {OptionalSectionTypeQuery}

</phrase>
```

where

field="FieldName"	is an optional attribute that identifies which field in the index to search. Often this attribute is set to text to indicate that the main text of the document should be searched. It can also be set to the name of a meta field such as author or subject. It should be mentioned that the field name specified by a <phrase> tag must match the field name set by any tags that contain it. Otherwise, an error will be generated. If no parent tags specify a field name, any field name can be used by the <phrase> tag. Similarly, any tags

	within a <phrase> tag must either specify the same field name as the phrase, or specify no field name at all. Otherwise, an error will be generated. (Note: Allowing nested copies of the field attribute doesn't serve a purpose other than to make it easier to write **docReqParser.xsl** stylesheets in a uniform way. Effectively, the outermost field value is always used.)
maxSnippets="SnippetsToOutput"	identifies the number of snippets to pass on to the **Result Formatter** stylesheet for display. A snippet is defined as the matching text found in a document for a particular query, along with some additional text around it for context. The amount of context displayed for each match is defined by the maxContext attribute. If not specified this attribute defaults to 3, meaning snippets for the top three matches for a document are returned. Also, this attribute can be set to -1, meaning all the snippets for a document are returned. As with the field attribute, any maxSnippets value set by the <phrase> tag must match the value set by any tags that contain it. Otherwise, an error will be generated. If no parent tags set a maxSnippets value, then any value may be specified by the <phrase> tag. Similarly, any tags within a <phrase> tag must either specify the same field name as the phrase, or specify no field name at all. Otherwise, an error will be generated. (Note: Allowing nested copies of the maxSnippets attribute doesn't serve a purpose other than to make it easier to write **docReqParser.xsl** stylesheets in a uniform way. Effectively, the outermost maxSnippets value is always used.)
boost="BoostValue"	is an optional attribute that specifies a relevance boost multiplier for this phrase in the query. Boost values higher than 1.0 increase the relevance of a phrase, while boost values between 0.0 and 1.0 decrease the relevance of a phrase. Boost values less than zero will generate an error. Note that boost values multiply. That is, if tags within a <phrase> tag have boost attributes, their individual boost values will be multiplied by the boost set for the containing <phrase> tag.

Within the phrase tag, Term is a term tag, and Clause is a phrase, and, or, orNear, near, range, resultData, or not tag.

Note that text will only match a phrase query if all the terms in the phrase tag appear in the document in same order, with no other intervening words. Note that it does not matter whether the phrase appears at the beginning, middle, or end of a field; it will match in any position. To match the exact contents of an entire field, use the exact tag.

If the field is "text", a Section Type sub-query may optionally appear, restricting this phrase query to particular sections of a document based on section types added by the **Pre-Filter** stylesheet at index time.

- Exact Tag: <exact>

This tag specifies an exact phrase to search for. This tag has the form:

```
<exact {field       = "FieldName"}
       {maxSnippets = "SnippetsToOutput"}
       {boost       = "BoostValue"}>

    Term | Clause
    Term | Clause
    …

    {OptionalSectionTypeQuery}

</exact>
```

where

`field="FieldName"`	is an optional attribute that identifies which field in the index to search. Normally, this attribute is set to the name of a meta field such as author or subject. It may not be set to text. It should be mentioned that the field name specified by a `<exact>` tag must match the field name set by any tags that contain it. Otherwise, an error will be generated. If no parent tags specify a field name, any field name can be used by the `<exact>` tag. Similarly, any tags within an `<exact>` tag must either specify the same field name as the phrase, or specify no field name at all. Otherwise, an error will be generated. (Note: Allowing nested copies of the field attribute doesn't serve a purpose other than to make it easier to write **docReqParser.xsl** stylesheets in a uniform way. Effectively, the outermost field value is always used.)
`maxSnippets="SnippetsToOutput"`	is an optional attribute that identifies the number of snippets to pass on to the **Result Formatter** stylesheet for display. A snippet is defined as the matching text found in a document for a particular query, along with some additional text around it for context. The amount of context displayed for each match is defined by the maxContext attribute. If not specified this attribute defaults to 3, meaning snippets for the top three matches for a document are returned. Also, this attribute can be set to -1, meaning all the snippets for a document are returned. As with the field attribute, any maxSnippets value set by the `<exact>` tag must match the value set by any tags that contain it. Otherwise, an error will be generated. If no parent tags set a maxSnippets value, then any value may be specified by the `<exact>` tag. Similarly, any tags within a `<exact>` tag must either specify the same field name as the phrase, or specify no field name at all. Otherwise, an error will be generated. (Note: Allowing nested copies of the maxSnippets attribute doesn't serve a purpose other than to make it easier to write **docReqParser.xsl** stylesheets in a uniform way. Effectively, the outermost maxSnippets value is always

	used.)
`boost`="BoostValue"	is an optional attribute that specifies a relevance boost multiplier for this phrase in the query. Boost values higher than 1.0 increase the relevance of a phrase, while boost values between 0.0 and 1.0 decrease the relevance of a phrase. Boost values less than zero will generate an error. Note that boost values multiply. That is, if tags within an `<exact>` tag have boost attributes, their individual boost values will be multiplied by the boost set for the containing `<exact>` tag.

Within the exact tag, Term is a term tag, and Clause is a phrase, and, or, orNear, near, range, resultData, or not tag.

Note that text will only match a exact query if all the terms in the exact tag appear in the document in same order, with no other intervening words. In addition, the match must begin at the start of the field and end at the end of the field (in other words, it must match the entire field.)

For example, an exact search for "man on the moon" would not match field contents"put a man on the moon", nor would it match "man on the moon base". Only a field whose contents are exactly the string "man on the moon" (and nothing else) will match.

Note that "exact"-ness refers only to the lack of other terms before and after the match; the normal rules for upper/lower case insensitivity, and accent/plural mapping still apply. So "man on the moon" will still match "Men on the MOON".

If the field is "text", a < a href="#tagRef_crossQuery_QueryParser_Output_SectionType">Section Type sub-query may optionally appear, restricting this query to particular sections of a document based on section types added by the **Pre-Filter** stylesheet at index time.

- And Tag: <and>

This tag defines a search where all of the sub-terms or clauses *must* exist in a document for a match to be made.

If a single field is specified (using the `field` attribute), documents will match if *all* of the sub-terms or clauses occur in that field. If `useProximity` is "yes" (the default), then documents with terms closer together and in the right order will score higher than those in which the terms are farther apart.

If multiple fields are specified (using the `fields` attribute), documents will match if *all* of the search terms occurs in *any* of the listed fields. That is, that is, it doesn't matter which field(s) they appear in, as long as all of them appear somewhere. If in one document multiple terms appear in the same field, the score will be higher, and if the terms are close together, the score will be higher still.

This tag has the form:

```
<and {field        = "FieldName" |
      fields       = "Field1,Field2,..."}
```

```
     {maxSnippets  = "SnippetsToOutput"}
     {boost        = "BoostValue"}
     {useProximity = "YesOrNo"}>

    Term | Clause
    Term | Clause
    …

    {OptionalSectionTypeQuery}

</and>
```

where

`field="`*FieldName*`"`	is an optional attribute that identifies which field in the index to search. Often this attribute is set to text to indicate that the main text of the document should be searched. It can also be set to the name of a meta field such as author or subject. It should be mentioned that if the field attribute appears in any tags outside it, the field name in the `<and>` tag ***must*** match the value set in the outer tags. Otherwise, an error will be generated. However, if the `<and>` tag does not specify a field name, and no tags outside it specify a field name, then the tags directly below the `<and>` tag may have any combination of field names desired. This is how mixed queries of document text and meta data are formed.
`fields="`*Field1*`,`*Field2*`,…"`	is a multi-field alternative to the `field` attribute. It identifies a list of fields in the index to search, instead of a single field. Using a list of fields is an ideal way to perform a "keyword" search, for example searching the title, subject, author, and full text of documents. It should be mentioned that the `fields` attribute is only applicable to `<and>` and `<or>` queries. As with the `field` attribute, any elements nested within the tag are ***not*** allowed to specify a field or fields.
`maxSnippets="`*SnippetsToOutput*`"`	is an optional attribute that identifies the number of snippets to pass on to the **Result Formatter** stylesheet for display. A snippet is defined as the matching text found in a document for a particular query, along with some additional text around it for context. The amount of context displayed for each match is defined by the maxContext attribute. If not specified this attribute defaults to 3, meaning snippets for the top three matches for a document are returned. Also, this attribute can be set to -1, meaning all the snippets for a document are returned. As with the field attribute, if the maxSnippets attribute appears in any containing tags, the value set for it in the `<and>` tag must match the value set in the containing tags. Otherwise, an error will be generated. Similarly, any occurrences of the maxSnippets attribute in tags within the `<and>` tag must have the same value as their containing tags. (Note: Allowing nested copies of the maxSnippets attribute

	doesn't serve a purpose other than to make it easier to write **docReqParser.xsl** stylesheets in a uniform way. Effectively, the outermost maxSnippets value is always used.)
`boost="`*BoostValue*`"`	is an optional attribute that specifies a relevance boost multiplier for this clause in the query. Boost values higher than 1.0 increase the relevance of a clause, while boost values between 0.0 and 1.0 decrease the relevance of a clause. Boost values less than zero will generate an error. Note that boost values multiply. That is, if tags within an `<and>` tag have boost attributes, their individual boost values will be multiplied by the boost set for the containing `<and>` tag.
`useProxmity="`*YesOrNo*`"`	is an optional attribute that specifies whether the AND query should take the proximity of terms into account. Generally it's best to leave this on (the default) as it results in higher quality results for the user. However, turning it off can increase query processing speed, as the Text Engine will have less work to do to calculate which documents match the query. If not specified, this attribute defaults to Yes, that is, proximity will be taken into account. Note that if proximity processing is turned off, individual text hits within document text and meta-data fields will *not* be highlighted, and scores for matching documents will be somewhat different.

Within the `<and>` tag, Term is a term tag, and Clause is a phrase, exact, and, or, orNear, near, range, resultData, or not tag. If the field is `"text"`, a Section Type sub-query may optionally appear, restricting this query to particular sections of a document based on section types added by the **Pre-Filter** stylesheet at index time.

- Or Tag: <or>

This tag defines a search where any one of the sub-terms or clauses must exist in a document for a match to be made. This tag has the form:

```
<or {field       = "FieldName" |
     fields      = "Field1,Field2,..."}
    {slop        = "MaxMatchDistance"}
    {maxSnippets = "SnippetsToOutput"}
    {boost       = "BoostValue"}>

    Term | Clause
    Term | Clause
    …

    {OptionalSectionTypeQuery}

</or>
```

where

`field="`*FieldName*`"`	is an optional attribute that identifies which field in the

	index to search. Normally, this attribute is set to text to indicate that the main text of the document should be searched. It can also be set to the name of a meta field such as author or subject. It should be mentioned that if the field attribute appears in any tags outside it, the field name in the `<or>` tag must match the value set in the outer tags. Otherwise, an error will be generated. However, if the `<or>` tag does not specify a field name, and no tags outside it specify a field name, then the tags directly below the `<or>` tag may have any combination of field names desired. This is how mixed queries of document text and meta data are formed.
`fields="`*Field1*`,`*Field2*`,…"`	is a multi-field alternative to the `field` attribute. It identifies a list of fields in the index to search, instead of a single field. Using a list of fields is an ideal way to perform a "keyword" search, for example searching the title, subject, author, and full text of documents. It should be mentioned that the `fields` attribute is only applicable to `<and>` and `<or>` queries. As with the `field` attribute, any elements nested within the tag are ***not*** allowed to specify a field or fields.
`slop="`*MaxMatchDistance*`"`	is a measure of the "nearness" of the terms or clauses within a given field of a multi-field query. Note that this attribute is *required* for multi-field `<or>` queries, and *not allowed* for single-field `<or>` queries. To get an idea of what `slop` does, see the discussion of this attribute for the related `<orNear>` query, following this query.
`maxSnippets="`*SnippetsToOutput*`"`	is an optional attribute that identifies the number of snippets to pass on to the **Result Formatter** stylesheet for display. A snippet is defined as the matching text found in a document for a particular query, along with some additional text around it for context. The amount of context displayed for each match is defined by the maxContext attribute. If not specified this attribute defaults to 3, meaning snippets for the top three matches for a document are returned. Also, this attribute can be set to -1, meaning all the snippets for a document are returned. As with the field attribute, if the maxSnippets attribute appears in any containing tags, the value set for it in the `<or>` tag must match the value set in the containing tags. Otherwise, an error will be generated. Similarly, any occurrences of the maxSnippets attribute in tags within the `<or>` tag must have the same value as their containing tags. (Note: Allowing nested copies of the maxSnippets attribute doesn't serve a purpose other than to make it easier to write **docReqParser.xsl** stylesheets in a uniform way. Effectively, the outermost maxSnippets value is always used.)

`boost="`*BoostValue*`"`	is an optional attribute that specifies a relevance boost multiplier for this clause in the query. Boost values higher than 1.0 increase the relevance of a clause, while boost values between 0.0 and 1.0 decrease the relevance of a clause. Boost values less than zero will generate an error. Note that boost values multiply. That is, if tags within an `<or>` tag have boost attributes, their individual boost values will be multiplied by the boost set for the containing `<or>` tag.

If a single field is specified (using the `field` attribute), documents will match if *any* search term occurs in that field. Documents with more matching terms will score and rank higher than those with fewer matching terms.

If multiple fields are specified (using the `fields` attribute), documents will match if *any* search term occurs in *any* of the listed fields. Documents with more matching terms (in any field) will score and rank higher than those with fewer matching terms. This will internally decompose to a conjoined set of `<orNear>` queries; hence the requirement for the `slop` attribute in this case.

Within the or tag, Term is a term tag, and Clause is a phrase, exact, and, or, orNear, near, range, resultData, or not tag. If the field is "text", a Section Type sub-query may optionally appear, restricting this phrase query to particular sections of a document based on section types added by the **Pre-Filter** stylesheet at index time.

- OrNear Tag: <orNear>

This tag defines a search where at least one of the sub-terms or clauses must be in a document. If multiple terms do appear, hits where the terms are close together will score higher than those where the terms are far from each other. Essentially, this provides the functionality of an <or> search, with improved scoring. This tag has the form:

```
<orNear slop         = "MaxMatchDistance"
        {field       = "FieldName"}
        {maxSnippets = "SnippetsToOutput"}
        {boost       = "BoostValue"}>

    Term | Clause
    Term | Clause
    …

    {OptionalSectionTypeQuery}

</orNear>
```

where

`slop="`MaxMatchDistance"	is a measure of the "nearness" of the terms or clauses specified. See the discussion of slop calculation below.
`field="`FieldName"	is an optional attribute that identifies which field in the index to search. Often this attribute is set to text to indicate that the main text of the document should be searched. It can also be set to the name of a meta field such as author or subject. It should be mentioned that the field name

	specified by a `<orNear>` tag must match the field name set by any tags that contain it. Otherwise, an error will be generated. If no parent tags specify a field name, any field name can be used by the `<orNear>` tag. Similarly, any tags within a `<orNear>` tag must either specify the same field name as the orNear clause, or specify no field name at all. Otherwise, an error will be generated. (Note: Allowing nested copies of the field attribute doesn't serve a purpose other than to make it easier to write **docReqParser.xsl** stylesheets in a uniform way. Effectively, the outermost field value is always used.)
`maxSnippets="SnippetsToOutput"`	is an optional attribute that identifies the number of snippets to pass on to the **Result Formatter** stylesheet for display. A snippet is defined as the matching text found in a document for a particular query, along with some additional text around it for context. The amount of context displayed for each match is defined by the maxContext attribute. If not specified this attribute defaults to 3, meaning snippets for the top three matches for a document are returned. Also, this attribute can be set to -1, meaning all the snippets for a document are returned. As with the field attribute, if the maxSnippets attribute appears in any containing tags, the value set for it in the `<orNear>` tag must match the value set in the containing tags. Otherwise, an error will be generated. Similarly, any occurrences of the maxSnippets attribute in tags within the `<orNear>` tag must have the same value as their containing tags. (Note: Allowing nested copies of the maxSnippets attribute doesn't serve a purpose other than to make it easier to write **docReqParser.xsl** stylesheets in a uniform way. Effectively, the outermost maxSnippets value is always used.)
`boost="BoostValue"`	is an optional attribute that specifies a relevance boost multiplier for this clause in the query. Boost values higher than 1.0 increase the relevance of a clause, while boost values between 0.0 and 1.0 decrease the relevance of a clause. Boost values less than zero will generate an error. Note that boost values multiply. That is, if tags within an `<orNear>` tag have boost attributes, their individual boost values will be multiplied by the boost set for the containing `<orNear>` tag.

Within the OrNear tag, Term is a term tag, and Clause is a phrase, exact, and, or, orNear, near, range, resultData, or not tag.

A slop value for a potential match is accumulated as follows:

1. A potential match is found when all the items specified in the orNear tag are found in a document. Initially, the accumulated slop for this potential match is set to zero.
2. Next the accumulated slop is incremented by one for each word in the potential match that doesn't appear in orNear tag.

3. Finally, the accumulated slop is incremented by one each time a word specified in the orNear tag appears out of order in the document. If the final accumulated slop exceeds the value set by the slop attribute of the orNear tag, the terms are considered to be separate (and thus lower scoring).

It should also be noted that, when applied to the "text" field, if the value specified for the slop attribute exceeds the maximum proximity for the index being used, the maximum proximity value will be used instead. There is no limit on the slop attribute when applied to meta-data fields.
If the field is "text", a Section Type sub-query may optionally appear, restricting this query to particular sections of a document based on section types added by the **Pre-Filter** stylesheet at index time.

- Not Tag: <not>

This tag defines a search where none of the sub-terms or clauses must exist in a document for a match to be made. This tag has the form:

```
<not {field       = "FieldName"}
     {maxSnippets = "SnippetsToOutput"}
     {boost       = "BoostValue"}>

    Term | Clause
    Term | Clause
    …
</not>
```

where

`field="FieldName"`	is an optional attribute that identifies which field in the index to search. Normally, this attribute is set to text to indicate that the main text of the document should be searched. It can also be set to the name of a meta field such as author or subject. It should be mentioned that if the field attribute appears in any tags outside it, the field name in the `<not>` tag must match the value set in the outer tags. Otherwise, an error will be generated. However, if the `<not>` tag does not specify a field name, and no tags outside it specify a field name, then the tags directly below the `<not>` tag may have any combination of field names desired. This is how mixed queries of document text and meta data are formed.
`maxSnippets="SnippetsToOutput"`	is an optional attribute that identifies the number of snippets to pass on to the **Result Formatter** stylesheet for display. A snippet is defined as the matching text found in a document for a particular query, along with some additional text around it for context. The amount of context displayed for each match is defined by the maxContext attribute. If not specified this attribute defaults to 3, meaning snippets for the top three matches for a document are returned. Also, this attribute can be set to -1, meaning all the snippets for a document are returned. As with the field attribute, if the maxSnippets attribute appears in any containing tags,

	the value set for it in the <not> tag must match the value set in the containing tags. Otherwise, an error will be generated. Similarly, any occurrences of the maxSnippets attribute in tags within the <not> tag must have the same value as their containing tags. (Note: Allowing nested copies of the maxSnippets attribute doesn't serve a purpose other than to make it easier to write **docReqParser.xsl** stylesheets in a uniform way. Effectively, the outermost maxSnippets value is always used.)
boost="BoostValue"	is an optional attribute that specifies a relevance boost multiplier for this clause in the query. Boost values higher than 1.0 increase the relevance of a clause, while boost values between 0.0 and 1.0 decrease the relevance of a clause. Boost values less than zero will generate an error. Note that boost values multiply. That is, if tags within an <not> tag have boost attributes, their individual boost values will be multiplied by the boost set for the containing <not> tag.

Within the not tag, Term is a term tag, and Clause is a phrase, exact, and, or, orNear, near, resultData, or range tag.

The not tag is usually used to restrict the results of a larger query (such as an **and**, **or**, or **near** query.) To use it this way, embed the not tag within another query tag. However, not can also be used alone at the top level (just inside the <query> element), and the effect is to gather *all* documents that don't match the <not> specification.

It is important to note that when applied to the "text" field, the not tag in reality operates as a **"not near."** That is, the items to not be found are localized to other terms in the search query. Why? Consider a query for **"bat" not "cave"**. Imagine a document where **"bat man"** appears in Chapter 1. Would you really want to ignore that match if the word "cave" appears in a wholly unrelated passage somewhere in Chapter 12? Probably not. Consequently, search elements specified within a <not> clause are considered relevant if they are within the same chunk as other elements in the query. For more about document chunks sizing, see **textIndexer Configuration File** section in the **XTF Deployment Guide**. By contrast, when applied to meta-data fields, the <not> clause operates as a "**not anywhere**", and ignores chunk sizing.

- Near Tag: <near>

This tag defines a search where all of the sub-terms or clauses must be in a document, **and** within a specified distance of each other for a match to be made. This tag has the form:

```
<near slop         = "MaxMatchDistance"
      {field       = "FieldName"}
      {maxSnippets = "SnippetsToOutput"}
      {boost       = "BoostValue"}>

    Term | Clause
    Term | Clause
    …

    {OptionalSectionTypeQuery}

</near>
```

where

`slop`="MaxMatchDistance"	is a measure of the "nearness" of the terms or clauses specified. See the discussion of slop values below.
`field`="FieldName"	is an optional attribute that identifies which field in the index to search. Often this attribute is set to text to indicate that the main text of the document should be searched. It can also be set to the name of a meta field such as author or subject. It should be mentioned that the field name specified by a `<near>` tag must match the field name set by any tags that contain it. Otherwise, an error will be generated. If no parent tags specify a field name, any field name can be used by the `<near>` tag. Similarly, any tags within a `<near>` tag must either specify the same field name as the near clause, or specify no field name at all. Otherwise, an error will be generated. (Note: Allowing nested copies of the field attribute doesn't serve a purpose other than to make it easier to write **docReqParser.xsl** stylesheets in a uniform way. Effectively, the outermost field value is always used.)
`maxSnippets`="SnippetsToOutput"	is an optional attribute that identifies the number of snippets to pass on to the **Result Formatter** stylesheet for display. A snippet is defined as the matching text found in a document for a particular query, along with some additional text around it for context. The amount of context displayed for each match is defined by the maxContext attribute. If not specified this attribute defaults to 3, meaning snippets for the top three matches for a document are returned. Also, this attribute can be set to -1, meaning all the snippets for a document are returned. As with the field attribute, if the maxSnippets attribute appears in any containing tags, the value set for it in the `<near>` tag must match the value set in the containing tags. Otherwise, an error will be generated. Similarly, any occurrences of the maxSnippets attribute in tags within the `<near>` tag must have the same value as their containing tags. (Note: Allowing nested copies of the maxSnippets attribute doesn't serve a purpose other than to make it easier to write **docReqParser.xsl** stylesheets in a uniform way. Effectively, the outermost maxSnippets value is always used.)
`boost`="BoostValue"	is an optional attribute that specifies a relevance boost multiplier for this clause in the query. Boost values higher than 1.0 increase the relevance of a clause, while boost values between 0.0 and 1.0 decrease the relevance of a clause. Boost values less than zero will generate an error. Note that boost values multiply. That is, if tags within an `<near>` tag have boost attributes, their individual boost values will be multiplied by the boost set for the containing `<near>` tag.

A slop value for a potential match is accumulated as follows:

1. A potential match is found when all the items specified in the near tag are found in a document. Initially, the accumulated slop for this potential match is set to zero.
2. Next the accumulated slop is incremented by one for each word in the potential match that doesn't appear in near tag.
3. Finally, the accumulated slop is incremented by one each time a word specified in the near tag appears out of order in the document. If the final accumulated slop exceeds the value set by the slop attribute of the near tag, the potential match is ignored. Note that setting the slop value for the near tag to zero effectively produces an exact phrase search, and is in fact how the phrase tag is implemented internally.

Within the near tag, Term is a term tag, and Clause is a phrase, exact, and, or, orNear, near, range, resultData, or not tag.

It should also be noted that, when applied to the "text" field, if the value specified for the slop attribute exceeds the maximum proximity for the index being used, the maximum proximity value will be used instead. There is no limit on the slop attribute when applied to meta-data fields.
If the field is "text", a Section Type sub-query may optionally appear, restricting this query to particular sections of a document based on section types added by the **Pre-Filter** stylesheet at index time.

- Range Tag: <range>

This tag matches any words that fall within the range specified by the specified first and last term. It has the form:

```
<range {inclusive   = "YesOrNo"}
       {numeric     = "YesOrNo"}
       {field       = "FieldName"}
       {maxSnippets = "SnippetsToOutput"}
       {boost       = "BoostValue"}>

    <lower>FirstTermToFind</lower>
    <upper>LastTermToFind</upper>

    {OptionalSectionTypeQuery}

</range>
```

where

`inclusive="YesOrNo"`	is an optional attribute that specifies whether the range should include the first and last term when matching. If not specified, this attribute defaults to **yes**.
`numeric="YesOrNo"`	is an optional attribute that specifies whether the data in the field is numeric and in a rigid consistent format. If set to yes, upon the first range query on the field, XTF will read into memory a table of all the data values, converting them to 64-bit integers; subsequent queries can then be processed extremely efficiently. If this attribute is set to no, the query is much more tolerant of variable formatting in the data; XTF will expand the range query into a multi-term

	OR (just like a wildcard query.) However, for some types of data, wildcard expansion can result in too many terms for the engine to handle. If not specified, this attribute defaults to no.
`field`="FieldName"	is an optional attribute that identifies which field in the index to search. Often this attribute is set to text to indicate that the main text of the document should be searched. It can also be set to the name of a meta field such as author or subject. It should be mentioned that if the field attribute appears in any tags outside it, the field name in the `<range>` tag must match the value set in the outer tags. Otherwise, an error will be generated. (Note: Allowing nested copies of the maxSnippets attribute doesn't serve a purpose other than to make it easier to write **docReqParser.xsl** stylesheets in a uniform way. Effectively, the outermost maxSnippets value is always used.)
`maxSnippets`="SnippetsToOutput"	is an optional attribute that identifies the number of snippets to pass on to the **Result Formatter** stylesheet for display. A snippet is defined as the matching text found in a document for a particular query, along with some additional text around it for context. The amount of context displayed for each match is defined by the maxContext attribute. If not specified this attribute defaults to 3, meaning snippets for the top three matches for a document are returned. Also, this attribute can be set to -1, meaning all the snippets for a document are returned. As with the field attribute, if the maxSnippets attribute appears in any containing tags, the value set for it in the `<range>` tag must match the value set in the containing tags. Otherwise, an error will be generated. (Note: Allowing nested copies of the maxSnippets attribute doesn't serve a purpose other than to make it easier to write **docReqParser.xsl** stylesheets in a uniform way. Effectively, the outermost maxSnippets value is always used.)
`boost`="BoostValue"	is an optional attribute that specifies a relevance boost multiplier for this clause in the query. Boost values higher than 1.0 increase the relevance of a clause, while boost values between 0.0 and 1.0 decrease the relevance of a clause. Boost values less than zero will generate an error.
FirstTermToFind	is the first term in the range to find. Usually, this is a starting number, date, or year.
LastTermToFind	is the last term in the range to find. Usually, this is an ending number, date, or year.

This tag returns a match for any word that lexicographically falls in the range specified by the upper and lower tags. This tag is primarily used to search for a range of revision numbers, dates, or years.

If the field is “text”, a Section Type sub-query may optionally appear, restricting this query to particular sections of a document based on section types added by the **Pre-Filter** stylesheet at index time.

- Section Type Tag: <sectionType>

This tag modifies other queries, causing them to apply to only certain portions of the full text of a document. It has the form:

```
<sectionType>

    Term | Clause

</sectionType>
```

This tag may only be used within another query, and has the effect of limiting that query to search only those parts of the full document text whose sectionType attributes match the specified term or clause. The sectionType tag is only allowed within queries on the “text” field. XTF evaluates the specified term or clause against the section type attributes recorded at index time by the **Pre-Filter** stylesheet using the xtf:sectionType attribute.

For example, if one wanted the option to only search the chapter headings of all books in a repository, then the pre-filter would be modified to mark all the headings with xtf:sectionType=”heading”, and then a sectionType tag would be added within the main text query, containing term tag on the word “heading”.

Within the sectionType tag, Term is a term tag, and Clause is a phrase, exact, and, or, orNear, near, range, resultData, or not tag.

- Facet Tag: <facet>

This tag specifies a facet for which to count hits and form groups, and optionally to gather document hits. This tag has the form:

```
<facet field               = "FieldName"
       {select             = "GroupsToSelect"}
       {sortGroupsBy       = "SortKind"}
       {sortDocsBy         = "ListOfMetaFields|score|totalHits"}
       {includeEmptyGroups = "YesOrNo"} />
```

where

`field=“FieldName”`	is a required attribute that identifies which meta-data field in the index for which to count and build groups. (Note: Meta tags to be used for faceted queries should also have an xtf:tokenize=”no” attribute set, or sorting will produce unpredictable results.)
`select=“GroupsToSelect”`	is an optional attribute specifying a subset of groups to select and return in the query result. For maximum flexibility, this specification is made using a special language that resembles XPath to some extent. It allows selecting groups by name or position in the list, and supports various operations on hierarchical meta-data. See examples below. If this attribute is not specified, it

	defaults to: *
`sortGroupsBy="TotalDocsOrValue"`	is an optional attribute telling XTF the order in which to sort groups. ▪ If set to "totalDocs" (or not set), groups will be sorted in decreasing order of the total number of documents per group. ▪ If set to "value", groups will be sorted in increasing order by the value (i.e. name) of the facet group. "reverseValue" is also supported. ▪ If set to "maxDocScore", groups will be sorted in decreasing order by their relevance score (relevance judged in relation to the main query.)
`sortDocsBy=` "ListOfMetaFields|score|totalHits"	is an optional attribute specifying a list of meta fields by which to sort the results. The list should consist of a quoted string containing one or more meta-field names, separated by commas. If multiple meta-fields are specified, the results are sorted first by the leftmost meta-field, then sub-sorted by subsequent fields to produce the final output. Optionally, each meta-field name can be preceded by a plus sign (+) or a minus sign (-) to indicate whether the results for that field should be sorted in ascending or descending order. If no plus or minus sign is specified for a meta-field, then the results are sorted in ascending order by default. An additional option is available: setting this attribute to "totalHits" will order the results by descending number of hits within each document. That is, a document with more hits will appear before one with fewer hits, regardless of the *quality* of those hits. If this attribute is not specified, documents are by default sorted in order of decreasing score (so the most "relevant" documents are first.) This default behavior can also be explicitly set by providing a value of "score" (or synonym "relevance"). (Note: Meta tags to be used for sorting should also have an xtf:tokenize="no" attribute set, or sorting will produce unpredictable results.)
`includeEmptyGroups="YesOrNo"`	is an optional attribute that specifies whether to include empty groups in the results. If set to "yes", empty groups will be included. If set to "no" they will be excluded. If this attribute is not specified, it defaults to "no."

The `<facet>` tag enables counting and grouping for a single meta-data field. First, XTF scans the index and forms a table of all the possible values of that field. Then the query is performed as normal, as each document hit is encountered, XTF looks up that document's value in the table and increments the count for it. If enabled in the selection specification, a list of the document hits for each value is also accumulated.

After the counting is completed, XTF sorts them, removes empty groups if enabled, and then applies the group selection specification (from the select attribute) to decide which groups to send to the **Result Formatter** stylesheet. For more information on the selection process, see the Group Selection section of the **XTF Programmer's Guide**. The final selected groups will appear within a Facet Result tag in the results.
Note that only one facet query is allowed per meta-data field. Trying to specify more than one will result in an error message.

Some examples of the `select` attribute:

```
 *[1-5]
 Politics#all
 **[topChoices]
 US::Berkeley#all|US::*
 History#all|**[selected][page(size=5)]
```

For more information, refer to the Group Selection section of the **XTF Programmer's Guide**.

- Spellcheck Tag: <spellcheck>

This tag specifies that the Text Engine should evaluate whether terms of the query were likely misspelled, and to suggest likely correction(s) for each term. This tag should appear directly within a Query Tag.

```
<spellcheck {fields          = "FieldNames" }
            {docScoreCutoff  = "MaxDocScore"}
            {totalDocsCutoff = "MaxDocCount"}/>
```

where

`fields`="FieldNames"	is an optional attribute that restricts spelling correction to the specified set of fields. The field names can be separated by commas, semicolons, pipe symbols (|), or spaces. If not specified, or if set to the special value #all, then all tokenized fields in the index will be checked for spelling (including the special field text which contains all words not marked as meta-data.) Specifying a subset of fields can speed up query processing if the **Query Parser** stylesheet introduces extra fields that the user didn't explicitly type, and thus needn't be checked for spelling.
`docScoreCutoff`="MaxDocScore"	is an optional attribute that controls whether XTF performs spelling correction. If any document resulting from the query scores higher than this number, no correction will be performed. If not specified, this attribute defaults to 0.0, which disables the score cutoff.
`totalDocsCutoff`="MaxDocCount"	is an optional attribute that controls whether XTF performs spelling correction. If the number of documents resulting from the query exceeds this number, spelling correction will not be performed. If set to zero, the document count cutoff is disabled (correction will always be considered.) If not specified, this attribute defaults to 10, meaning that if less than 10 documents are found by a query, spelling correction is applied.

If spelling corrections are found, the **Result Formatter** stylesheet will receive a Spelling Result tag.

- Result Data Tag: <resultData>

This tag allows the **Query Parser** stylesheet to pass arbitrary data on to the **Result Formatter** stylesheet. XTF will not attempt to interpret the contents of the tag, but simply passes the tag and its contents through unchanged into the Query Result Tag. This tag may appear anywhere within a query.

```
<resultData>

    YourDataHere

</resultData>
```

XTF will pass the tag and its contents unchanged to the result formatter.

- More Like Tag: <moreLike>

This tag specifies that XTF should locate documents "similar" to a given document, where many of the similarity parameters can be controlled. This tag has the form:

```
<moreLike  fields        = "FieldList"
          {boosts        = "BoostFactorList"}
          {minWordLen    = "MinWordLength"}
          {maxWordLen    = "MaxWordLength"}
          {minDocFreq    = "MinDocFrequency"}
          {maxDocFreq    = "MaxDocFrequency"}
          {minTermFreq   = "MinTermFrequency"}
          {termBoost     = "ShouldBoostTerms"}
          {maxQueryTerms = "MaxQueryTerms"}>

    DocumentQuery

</moreLike>
```

where

`fields="`*FieldList*`"`	is a required attribute naming all of the fields that XTF should search for "interesting" terms. The field names may be separated by spaces, commas, semicolons, or pipe symbols "|". For best performance, this list should be kept relatively small, and concentrate on fields of most interest to users, such as title, author, subject, etc. Note that XTF currently does not support using the special field name text to search the full document text for interesting terms, and behavior is undefined if you specify this as a field name.
`boosts="`*BoostFactorList*`"`	is an optional attribute specified exactly one boost factor for each field listed in the fields attribute. Each boost factor should be a non-negative decimal number, and is multiplied into the scoring for all terms from the given field. For example, a boost factor of 0.5 will reduce the score for terms by half, while a factor of 2.0 will double

	the score. In general, the boost factor is very useful in adjusting the weight that various fields have on selecting similar documents. For instance, if one decided that the title should be twice as important as author and subject, the fields attribute might be “title,author,subject” and the boosts attribute would be “2.0,1.0,1.0″. If not specified, the boost factor for all fields in the fields list is set to 1.0.
`minWordLen="`*MinWordLength*`"`	is an optional attribute that limits the length of terms from the source fields that will be considered for similarity matching. Terms shorter than the specified number of characters will be disregarded. This can speed up processing and improve results by getting rid of useless words. If not specified, this attribute defaults to 4.
`maxWordLen="`*MaxWordLength*`"`	is an optional attribute that limits the length of terms from the source fields will be considered for similarity matching. Terms longer than the specified number of characters will be disregarded. This can speed up processing and improve results by getting rid of useless words. If not specified, this attribute defaults to 12.
`minDocFreq="`*MinDocFrequency*`"`	is an optional attribute that helps select which terms from source fields will be considered for similarity matching. In particular, terms that appear in fewer than the specified number of documents will be discarded. This can speed processing and improve results by discarding highly unusual terms. If not specified, this attribute defaults to 2.
`maxDocFreq="`*MaxDocFrequency*`"`	is an optional attribute that helps select which terms from source fields will be considered for similarity matching. In particular, terms that appear in more than the specified number of documents will be discarded. This can speed processing and improve results by discarding very common terms. If not specified, this attribute defaults to -1, meaning that there is no limit at all.
`minTermFreq="`*MinTermFrequency*`"`	is an optional attribute that helps select which terms from source fields will be considered for similarity matching. In particular, if the term occurs in the original field less than the specified number of times, it will be discarded. This can help choose more relevant terms by concentrating on those that are repeated in the field. If not specified, this attribute defaults to 1.
`termBoost="`*ShouldBoostTerms*`"`	is an optional attribute controls whether the similarity engine should calculate and attach a boost factor to each term. This factor will be equal to the score that was calculated for that term, and serves to make more important terms select documents more specifically. In general, it’s best to leave this at the default value, which is true.

`maxQueryTerms="`*MaxQueryTerms*`"`	is an optional attribute that controls how many "interesting" terms are selected from the original document's fields. Generally, this should be chosen to balance speed (more terms take longer to process) vs. quality (more terms can result in higher quality results, up to a point.) If not specified, this attribute defaults to 10.

Within the `moreLike` tag, *DocumentQuery* is a normal XTF query that results in a single document. That document's fields will be scanned, and each term will be scored for "interestingness" subject to the attributes above. Those terms that rank highest will be combined into a new `<orNear>` query, and the results will be documents that are similar to the original document selected by DocumentQuery.

- All Docs Tag: <allDocs>

  This special query tag matches all of the documents in the index. Doing this can be useful in conjunction with faceted browsing, in order to explore the entire collection rather than a subset of it. This should appear within a <query> element, and has the form:

```
<allDocs/>
```

Search Result Formatter

- Input Tags

  The following tags make up the XML input for the Search Result Formatter stylesheet. They constitute a simple XML representation of the matching text found for the most recent query. The Search Result Formatter stylesheet uses these input tags to generate the HTML for the actual search result web-page viewed by the user. The root tag is <crossQueryResult>.

  <crossQueryResult>
  <docHit>
  <meta>
  <snippet>

  <hit>
  <term>
  <explanation>
  <facet>
  <group>

  <spelling>
  <suggestion>

- Query Result Tag <crossQueryResult>

  This tag is the outermost container tag for the results produced by the crossQuery servlet. It has the form:

```
<crossQueryResult queryTime = "TimeInSeconds"
                  totalDocs = "NumberOfDocs"
                  startDoc  = "FirstDocNumber"
```

```
                  endDoc    = "LastDocNumber">

    Parameters
    Query

    Spelling      <!-- if spelling corrections requested and applicable -->

    DocumentHit
    DocumentHit
    …

    FacetResult   <!-- if facets were queried -->

    FacetResult

</crossQueryResult>
```

where

`queryTime=“TimeInSeconds”`	is the amount of time, in seconds, that the servlet spent parsing the query, processing it, and gathering the results.
`totalDocs=“NumberOfDocs”`	is the number of documents that had matches for the specified query.
`startDoc=“FirstDocNumber”`	is the sequential document number for the highest ranking document returned by the current Query. Note that this may not be the overall highest ranking document if a paged query was specified. See the query tag for more details about performing paged queries.
`endDoc=“LastDocNumber”`	is the sequential document number for the lowest ranking document match returned by the current query. Note that this may not be the overall lowest ranking document if a paged query was specified. See the query tag for more details about performing paged queries.

and

`Parameters`	is the same `<parameters>` block that was sent to the Query Parser stylesheet
`Query`	is the query block that was output by the Query Parser stylesheet, included for reference. This `<query>` block and the `<parameters>` block may be useful to the **Search Result Formatter** stylesheet in formulating its output.
`Spelling`	is a tag that will appear only if spell checking was enabled in the query, *and* a spelling correction dictionary was created at index time (i.e. enabled in textIndexer.conf), *and* the engine detected likely misspelled words and suitable suggested replacements. See the Spelling Correction Result Tag tag for more information.
`DocumentHit`	is one or more tags, one per document in which search hits were found. These tags contain specific hits for each document. See the Document Hit Tag
`FacetResult`	is a tag that will be included only if a facet query was performed. These tags will contain grouped counts of the matching documents. See the Facet Result Tag for more information.

- Document Hit Tag <docHit>

This tag identifies a document containing one or more matches for the current query. This tag has the form:

```
<docHit rank  = "DocRelevanceRank"
        path  = "DocumentLocation"
        score = "DocRelevanceScore">

    DocumentMetaData

    Snippet
    Snippet
    …

    {ScoreExplanation}

</docHit>
```

where

rank="DocRelevanceRank"	is the ordinal ranking of this matching document, with 1 being the most relevant document for a query. Note that this is an absolute ranking for the document with respect to the entire query, and not a page relative ranking. For more information about paged queries, see the query tag.
path="DocumentLocation"	is the file path and name for the matching document. This path is relative to the base XTF directory (i.e., XTF_HOME.)
score="DocRelevanceScore"	is the document relevance score ranging from 0% to 100%. The document with the highest overall relevance will receive a score of 100%, and less relevant documents will receive lower scores. Note that this score is an overall relevance score and is not affected by paging. For information about paged queries, see the query tag.
Document MetaData	is a tag which contains the meta-data for the matching document. See the Document Meta-Data Tag for more information.
Snippet	are tag(s) that contain the text of individual hits within the document, along with surrounding context, as requested in the query. See the Snippet Tag for more information.
Score Explanation	are tags only present if score explanation was requested in the query. These tags detail how the textEngine calculated the score for this document. See the Score Explanation Tag for more information.

- Document Meta-Data Tag <meta>

This tag identifies a block of meta-data for a matching document. This tag has the form:

```
<meta>
    …
</meta>
```

The actual tags within the meta-data block will depend on the implementation of the **Pre-Filter** used by the textIndexer. However, any matches found in the meta-data will be marked with Snippet or Hit tags, depending on what the query specified (snippets by default.).

The document meta-data tag is always included in the query results, regardless of whether there are any matches in it or not. This guarantees that the result formatter has access to the title of the document and other document related information in addition to the match results.

- Snippet Tag <snippet>

This tag contains a snippet of text associated with a match found in a document. The snippet consists of the text matched along with some context text around it. This tag has the form:

```
<snippet rank="MatchRelevanceRank" score="MatchRelevanceScore">

    Hit Text (and context text, if any)

</snippet>
```

where

rank="MatchRelevanceRank"	is the ordinal ranking of this match in the current document, with 1 being the most relevant match in the document. Note that this is an absolute ranking for the match with respect to the entire document, and not a page relative ranking. For more information about paged queries, see the query tag.
score="MatchRelevanceScore"	is the relevance score for this match ranging from 0% to 100%. The snippet with the highest overall relevance will receive a score of 100%, and less relevant snippets will receive lower scores. Note that this score is an overall relevance score and is not affected by paging. For information about paged queries, see the query tag.

The amount of context text displayed in a snippet is determined by the query tag in the original query. If the context length specified is less than the matched text, then only the matched text will be displayed.

Within the snippet itself, each matching word from the query will be marked with a single term tag, and the matching text around which the context text is centered on will also be marked with a single hit tag.

- Hit Tag <hit>

This tag identifies actual matched text within the context text of a snippet tag. This tag has the form:

```
<hit>
    …
</hit>
```

A hit may contain one or more matched words, which are separately marked with term tags.

If the original query used a near or and clause, the hit tag will mark the entire range of text between the first and last word found for the clause. For example, for a “man” near “war” query, the result would look like this:

```
<snippet rank="3" score="86">
```

```
    that <hit><term>man</term> had never actually been
    to <term>war</term></hit>, but he spoke as if he had

</snippet>
```

For an OR query, if multiple matched words exist in the snippet, all the matched words will be marked with Term tags, but only the word that this snippet is centered around will be marked with a hit tag.

- Term Tag <term>

  This tag identifies a word in a document that matches one of the terms in the query. This tag has the form:

```
<term>
    MatchedWord
</term>
```

  A term tag may or may not be inside a hit tag, depending on whether the occurrence of the matched word is within the primary match for a snippet, or simply another occurrence within the context text.

- Explanation Tag <explanation>

  This optional tag contains an explanation of how the score used to rank a document hit was calculated. This tag has the form:

```
<explanation value="Score" description="Description">

    <explanation...>

    <explanation...>
    ...

</explanation>
```

  where

Score	is a floating-point number calculated by XTF
Description	is a brief, technical description of how this score value was calculated.

  To further describe the score, there may be one or more nested <explanation> elements which break down the score's components, and they may have their own nested explanations in turn.

  Score explanations are enabled by setting the explainScores attribute of the Query Tag produced by the **Query Parser** stylesheet. In the default XTF stylesheets, one can simply add &explainScores=1 to the query URL.

- Facet Tag <facet>

  This tag identifies and contains the results for counting/grouping on one facet (or meta-data field). One copy of this tag will be sent into the **Result Formatter** for each Facet Query Tag specified in the query. It will appear within a Query Result Tag container. It has the form:

```
<facet field       = "FieldName"
       totalGroups = "NumberOfGroups"
```

```
       totalDocs   = "NumberOfDocs">

    GroupResult
    GroupResult
    …

</facet>
```

where

`field=`“FieldName”	is the name of the meta-data field for which faceted data is being reported.
`totalGroups=`“NumberOfGroups”	is the number of groups groups this facet contains (which may be more than are selected and returned as GroupResults.) In the case of a hierarchical facet, this is actually a count of the top-level groups only.
`totalDocs=`“NumberOfDocs”	is the number of documents that had matches for the specified query *and* had a value for this facet.

If any groups matched the group selection in the Facet Query Tag, then one or more Group Result Tags will appear as children of the Facet Result Tag.

- Group Tag <group>

#crossQuery_ResultFormatter_Group?This tag identifies and contains the results for the count (and possibly document hits) for a single group within a facet. One copy of this tag will be sent into the **Result Formatter** for group selected by the Facet Query Tag specified in the query. The Group Result Tag will appear within a Facet Result Tag container, or in the case of a hierarchical facet, may appear inside another Group Result Tag. It has the form:

```
<group value          = "GroupValue"
       rank           = "GroupSortedRank"
       totalSubGroups = "NumOfSubGroups"
       totalDocs      = "NumberOfDocs"
       startDoc       = "FirstDocNumber"
       endDoc         = "EndDocNumber">

    GroupResult   <!-- if facet is hierarchical -->
    GroupResult
    …

    DocumentHit   <!-- if document hits were requested -->
    DocumentHit
    …

</group>
```

where

`value=`“GroupValue”	is the specific facet value of the group being reported. One might also think of this as the “name” of the group.
`rank=`“GroupSortedRank”	is the ordinal ranking of this group within the set of groups at this level, with 1 being the first in sort order. Note that this is an absolute ranking for the group with

	respect to the entire set, and not a page relative ranking. For more information about paging groups, see the Group Selection section of the **XTF Programmer's Guide**.
`totalSubGroups="NumOfSubGroups"`	is, for a hierarchical facet, the number of sub-groups this group contains, which may be more than were actually selected and returned. For a non-hierarchical facet, this will always be zero.
`totalDocs="NumberOfDocs"`	is the number of documents that had matches for the specified main query ***and*** had GroupValue in the facet field.
`startDoc="FirstDocNumber"`	is the sequential document number for the highest ranking document match reported for the current group. Note that this may not be the overall highest ranking document if a paged query was specified. See the Group Selection section of the **XTF Programmer's Guide** for more details about paging documents.
`endDoc="LastDocNumber"`	is the sequential document number for the lowest ranking document match reported for the current group. Note that this may not be the overall lowest ranking document if a paged query was specified. See the Group Selection section of the **XTF Programmer's Guide** for more details about paging documents.

If any groups matched the group selection in the Facet Query Tag, then one or more Group Result Tags will appear as children of the Facet Result Tag. If the facet is also hierarchical, Group Result Tags may contain nested Group Result Tags for sub-groups. If any document hits were selected, then one or more Document Hit Tags will also be included.

- Spelling Tag <spelling>

  This tag contains the suggestions resulting from spelling correction performed on a user query. It will only appear directly within a Query Result tag.

```
<spelling>

    SpellingSuggestion
    SpellingSuggestion
    …

</spelling>
```

  Within the Spelling Tag, one ore more Spelling Suggestion tags will appear, one for each potentially misspelled term in the original query submitted to the engine.

  Note that this tag will only appear if spell checking was enabled in the query, *and* a spelling correction dictionary was created at index time (i.e. enabled in textIndexer.conf), *and* the engine detected likely misspelled words and suitable suggested replacements.

- Spelling Suggestion Tag <suggestion>

This tag contains a spelling suggestion for a single term from the original query submitted to the **Text Engine**. It will appear directly within a Spelling Correction Result tag.

```
<suggestion origTerm      = "OriginalWord"
            suggestedTerm = "ReplacementWord"/>
```

where

`origTerm`="OriginalWord"	is the (potentially) misspelled term found in the original query.
`suggestedTerm`="ReplacementWord"	is the best correction that the spelling engine could find for the original term. This may be two words if the original word should be split (e.g. "harrypotter" -> "harry potter"). This may be an empty string, indicating that the original word should be removed from the query (e.g. "usa" "bility" -> "usability" "empty"

Error Generator

The purpose of the **Error Generator** stylesheet is to generate a web-page that displays user friendly messages when crossQuery errors occur. Since this stylesheet works the same way in both dynaXML and crossQuery, it is documented in the common Error Generator Stylesheets section.

dynamXML

return to top

Document Request Parser

- Document Request Parser Input Tags and Parameters

  The XML input fragment passed to the **Document Request Parser** stylesheet forms a simple XML representation of the document request URL supplied to the dynaXML servlet from either a document catalog web-page or a crossQuery search result web-page. The XML input and XSL parameters are the same as those for the crossQuery **Query Parser**, and are documented in the Common Parser Input Tags and Common Stylesheet Parameters pages.

- Output Tags

  The following tags form the output from a dynaXML Document Request Parser stylesheet. The parser is required to output a document request in the form of at least a <style> tag and an <auth> tag, and optionally other tags as outlined below. This XML request is then passed to the dynaXML servlet's document retrieval engine for processing.

  <style>

  <source>
  <brand>
  <index>
  <query>

Public Authentication
IP List Authentication

LDAP Authentication
External Authentication

- <style>

This tag identifies the location of the **Document Formatter** Stylesheet. It has the form:

```
<style path="DocFormatterLocation"/>
```

where

path="DocFormatterLocation"	is the file path and name for the **Document Formatter** stylesheet to use for the requested document. If this path is not specified as an absolute path, it is assumed to be relative to the base XTF installation directory (i.e., **XTF_HOME**.)

- <source>

This tag identifies the location of the document to be retrieved. It has the form:

```
<source path="SrcDocLocation"/>
```

where

path="SrcDocLocation"	is the file path and name for the document to be retrieved. If this path is not specified as an absolute path, it is assumed to be relative to the base XTF installation directory (i.e., **XTF_HOME**.)

- <brand>

This tag identifies a file that contains a list of site-specific parameters to be passed on to the document formatter. It has the form:

```
<brand path="BrandStylesheetLocation"/>
```

where

path="BrandStylesheetLocation"	is the file path and name for the brand file to use. If this path is not specified as an absolute path, it is assumed to be relative to the base XTF installation directory (i.e., **XTF_HOME**.)

This contents of the branding file is a set of parameter definitions of the form:

```
<name>value</name>
<name>value</name>
...
```

These parameters are most often used to pass "branding" information to the **Document Formatter** stylesheet (e.g., background color to use, cascading stylesheet to use, font to use, etc.)

- <index>

  This tag identifies the index to use for lazy loading of text, and for marking search hits in a document. It has the form:

```
<index configPath="TextIndexerConfigLocation" name="IndexName"/>
```

  where

`configPath=`"TextIndexerConfigLocation"	is the file path and name for the textIndexer config file. If this path is not specified as an absolute path, it is assumed to be relative to the base XTF installation directory (i.e., **XTF_HOME**.)
`name=`"IndexName"	is the name of the index to use. This index name must exist in the config file specified by the configPath attribute above.

  This tag is required if the document request passed to the dynaXML servlet will include a query tag. Using the query tag in a document request without an index tag will result in an Unsupported Query Error being sent to the **Error Generator** stylesheet.

  Placing this tag in a document request will also enable lazy loading of the requested document (i.e., portions of the document are loaded only when viewed by the user.) Using lazy loading can substantially increase the responsiveness of the XTF system for libraries of large documents.

- <query>

  This tag is a container tag for a crossQuery style query. It has the form:

```
<query>

    crossQuery-Style Query Tags

</query>
```

  where crossQuery-Style Query Tags are any of the query tags outlined in the Query Parser Output Tags section. Note however that the dynaXML query tag doesn't use the attributes available for the crossQuery query tag.

  Including a query in the document request allows the dynaXML servlet to mark query hits in context in the original document. Then, the **Document Formatter** stylesheet can opt to provide quick links to the hits or to simply highlight them in context.

  Note: If a query tag is used in a document request, the index tag **must** also be present in the document request. If the index tag is not present in the document request, an Unsupported Query error will be sent to the **Error Generator** stylesheet.

- Public Authentication

  This variant of the authentication tag is used to provide or deny full public access to documents. It has the form:

```
<auth access="AllowOrDeny" type="all"/>
```

where

`access="AllowOrDeny"`	specifies whether all users should be allowed access (access="allow") or denied access (access="deny") to the requested document.

Allowing access to all users with this tag can be used when no special authorization is required to access on-line documents.

Note: One or more auth tags must exist in the **Document Request Parser** stylesheet. These tags will be processed in the order they are encountered until one of them authorizes or denies access. If none of the authentication tags explicitly authorize or deny access, the dynaXML servlet will deny access by default.

- IP List Authentication

This variant of the authentication tag is used to provide access to documents based on the user's IP address. This tag provides simple (but not particularly robust) authorization for document access. It has the form:

```
<auth access="AllowOrDeny" type="IP" list="LocationOfIPList"/>
```

where

`access="AllowOrDeny"`	specifies whether addresses in the IP list should be allowed access (access="allow") or denied access (access="deny") to the requested document.
`list="LocationOfIPList"`	specifies the path and filename of a list of IP addresses to allow or deny access. If not specified as an absolute path, this path is assumed to be relative to the XTF base install directory (i.e., XTF_HOME.) To learn about the format of the IP List file, see the XTF Deployment Guide.

Note: One or more auth tags must exist in the **Document Request Parser** stylesheet. These tags will be processed in the order they are encountered until one of them authorizes or denies access. If none of the authentication tags explicitly authorize or deny access, the dynaXML servlet will deny access by default.

- LDAP Authentication

This variant of the authentication tag is used to provide access to documents based on an LDAP database. It has the form:

```
<auth access        = "allow"
      type          = "LDAP"
      server        = "LDAPServerURL"
      realm         = "PswdRequestDescr"
     {bindName      = "LDAPConnectName"}
     {bindPassword  = "LDAPConnectPswd"}
     {queryName     = "LDAPRecordNameToFind"}
     {matchField    = "LDAPFieldToFind"}
     {matchValue    = "LDAPValueToMatch"}/>
```

where

`server="LDAPServerURL"`	identifies the location of the LDAP server to use.
`realm="PswdDialogDescr"`	is a string to display in the browser dialog box that asks for the user's name and password.
`bindName="LDAPConnectName"`	is an optional attribute specifying the name to use when connecting to the LDAP server. If this attribute is omitted, then an anonymous LDAP connection will be attempted. If anonymous connections are permitted by the LDAP database, then the bindPassword attribute should also be omitted, and the queryName attribute must be present for user authentication to proceed. For anonymous LDAP access the matchField and matchValue attributes are optional. If the name passed for this attribute is the LDAP administrator name, then the bindPassword attribute must be set to the LDAP administrator password, and the queryName must also be present for user authentication to proceed. For administrative LDAP access, the matchField and matchValue attributes are optional. It should also be noted that the user name will be substituted for any occurrence of the % symbol in this attribute. Doing so allows connections with the LDAP database to be established using the user name instead of an LDAP administrator name. Finally, if successfully connecting to the LDAP database with a user name and password is all that is required for authentication, then no other attributes need to be specified in the authentication tag. Otherwise, the queryName attribute and optionally the matchField and matchValue attributes may be specified to complete the authentication request.
`bindPassword="LDAPConnectPswd"`	is an optional attribute specifying the password to use when connecting to the LDAP server. If an anonymous LDAP connection is being performed (i.e, the bindName attribute has not been specified), this attribute should also not appear in the authentication tag. If the bindName attribute specifies the LDAP administrator name, this attribute must be set to the LDAP administrator password. Finally, the user password will be substituted for any occurrence of the % symbol in this attribute. Doing so allows connections with the LDAP database to be established using the user password instead of an LDAP administrator password.
`queryName="LDAPRecordToFind"`	is an attribute identifying the name of an LDAP record to find. If an anonymous or administrator connection to the LDAP server is being attempted, this attribute is required. For user connections, this attribute is optional. As with the bindName attribute, the user name will be substituted for any occurrence of the % symbol in this attribute. Doing so allows connections with the LDAP

	database to be established using the user name instead of an LDAP administrator name. Also, if the queryName attribute is specified without the matchField or matchValue attributes, then user authentication will succeed if the given record name simply exists in the LDAP database. If the given record is not in the LDAP database, authentication will fail.
`matchField=`“LDAPFieldToFind”	is an attribute identifying the name of a field to find in the LDAP record named by the queryName attribute. Note that the matchField attribute should not be used if the queryName attribute hasn’t been specified. Like the queryName attribute, the user name will be substituted for any occurrence of the % symbol in this attribute. Doing so allows connections with the LDAP database to be established using the user name instead of an LDAP administrator name. Finally, if the matchField attribute is specified without the matchValue attribute, then user authentication will succeed if the given field name simply exists in the LDAP record. If the given field name does not exist in the LDAP database authentication will fail.
`matchValue=`“LDAPValueToMatch”	is an attribute that specifies the value that must exist in the LDAP field named by the matchField attribute for authentication to succeed. If the specified value doesn’t match the LDAP field, user authentication will fail. As with the previous attributes, the user’s password will be substituted for any occurrences of the % symbol. Doing so allows connections with the LDAP database to be established using the user password instead of an LDAP administrator password.

Note: One or more auth tags must exist in the **Document Request Parser** stylesheet. These tags will be processed in the order they are encountered until one of them authorizes or denies access. If none of the authentication tags explicitly authorize or deny access, the dynaXML servlet will deny access by default.

- External Authentication

This variant of the authentication tag is used to provide access to documents based on an external authentication web-page or server. It has the form:

```
<auth access = "allow"
      type   = "external"
      key    = "SecretKeyStr"
      url    = "AuthenticationURL"/>
```

Note: One or more auth tags must exist in the **Document Request Parser** stylesheet. These tags will be processed in the order they are encountered until one of them authorizes or denies access. If none of the authentication tags explicitly authorize or deny access, the dynaXML servlet will deny access by default. For more details about external authentication, see the XTF Deployment Guide.

Document Formatter

The following tags and attributes are added by the dynaXML servlet to the original XML tags that make up the requested XML document, which are then passed to the **Document Formatter** stylesheet for output formatting.

Document Formatter Attributes

```
xtf:hitCount="NumberOfHitsBelowThisTag"
xtf:firstHit="FirstHitNumberBelowThisTag"
```

These attributes are added to XML documents to indicate where matched text hits are located. By providing these tags, the dynaXML servlet allows the **Document Formatter** stylesheet to quickly determine if a section of a document needs any special highlighting or not.

If the requested document has no hits, these attributes will appear once in the outermost tag for the document with both the attributes set to zero. If the document has one or more hits, these attributes will appear for any XML tag that has a hit inside it or inside its child tags.

Document Formatter Tags

<xtf:snippets>
<xtf:snippet>

<xtf:hit>
<xtf:more>
<xtf:term>

- <xtf:snippets>

  This tag is a container tag added to the start of the document when there are query results for the document. It has the form:

```
<xtf:snippets>

   Snippet
   Snippet
   …

</xtf:snippets>
```

  where each Snippet is a dynaXML snippet tag that summarizes one query match in the requested document.

  Note: The `<snippets>` tag is prefixed with the xtf: namespace to differentiate it from tags that came from the original XML document.

- <xtf:snippet>

  This tag contains a snippet of text associated with a match found in the requested document. The snippet consists of the text matched along with some context text around it. This tag has the form:

```
<xtf:snippet hitNum="HitNumber"
             rank="MatchRelevanceRank"
```

```
              score="MatchRelevanceScore">

    Hit Text (and context text, if any)

</xtf:snippet>
```

where

`xtf:`	is an XTF namespace prefix added to the tag to differentiate it from tags that came from the original XML document. The namespace URI for this prefix is: `http://cdlib.org/xtf`
`hitNum="HitNumber"`	is the ordinal ID of the current hit. This attribute will also appear in hit tags in the main text, allowing the hit number for the next or previous in-context tag to be easily determined.
`rank="MatchRelevanceRank"`	is the ranking of this match in the current document, with 1 being the most relevant match in the document.
`score="MatchRelevanceScore"`	is the relevance score for this match ranging from 0% to 100%. The snippet with the highest overall relevance will receive a score of 100%, and less relevant snippets will receive lower scores.

The amount of context text displayed in a snippet is determined by the query tag in the original query. If the context length specified is less than the matched text, then only the matched text will be displayed.

Within the snippet itself, each matching word from the query will be marked with a single Term Tag, and the matching text around which the context text is centered on will also be marked with a single Hit Tag.

- <xtf:hit>

This tag identifies actual matched text in a snippet tag or in the main text for the requested document. This tag has the form:

```
<xtf:hit hitNum    = "HitNumber"
         rank      = "MatchRelevanceRank"
         score     = "MatchRelevanceScore"
         continues = "YesOrNo">

    …

</xtf:hit>
```

where

`xtf:`	is an XTF namespace prefix added to the tag to differentiate it from tags that came from the original XML document. The prefix will not be present in hit tags that appear within the initial snippets summary tag, but only in hit tags that occur in the main text for the document. The namespace URI for this prefix is: `http://cdlib.org/xtf`
`hitNum="HitNumber"`	is the ordinal ID of the current hit. This attribute allows the hit number for the next or previous in-context tag to be easily determined.

`rank`="MatchRelevanceRank"	is the ranking of this match in the current document, with 1 being the most relevant match in the document.
`score`="MatchRelevanceScore"	is the relevance score for this match ranging from 0% to 100%. The snippet with the highest overall relevance will receive a score of 100%, and less relevant snippets will receive lower scores.
`continues`="YesOrNo"	indicates whether this hit continues into the next XML tag (continues="**yes**") or not (continues="**no**"). Note that when a hit continues into the next XML tag, a More Tag will always follow.

A hit may contain one or more matched words, which are separately marked with Term Tags.

If the query in the document request included a near or and clause, the Hit Tag will mark the entire range of text between the first and last word found for the clause. Also, within the main text, if the entire range spans multiple XML tags in the document, the hit will be broken down into an initial Hit Tag followed by one or more More Tags.

For summary snippets, if the query for the requested document includes an or clause, all the matched words in the snippet will be marked with Term Tags, but only the word that the snippet is centered around will be marked with a hit tag. Within the main text however, any term matched in the or clause will be marked with **both** a Hit Tag and a Term Tag.

- <xtf:more>

This tag identifies the continuation of a hit tag that started in a previous XML tag. This tag has the form:

```
<xtf:more hitNum    = "HitNumber"
          rank      = "MatchRelevanceRank"
          score     = "MatchRelevanceScore"
          continues = "YesOrNo">

    …

</xtf:more>
```

where

`xtf:`	is an XTF namespace prefix added to the tag to differentiate it from tags that came from the original XML document. The namespace URI for this prefix is: `http://cdlib.org/xtf`
`hitNum`="HitNumber"	is the ordinal ID of the hit to which this More Tag belongs.
`rank`="MatchRelevanceRank"	is the ranking of the hit to which this More Tag belongs.
`score`="MatchRelevanceScore"	is the relevance score of the hit to which this More Tag belongs.
`continues`="YesOrNo"	indicates whether the associated hit continues into yet another XML tag (continues="**yes**") or not (continues="**no**"). Note that when a hit continues into another XML tag, another more tag will always follow.

A More Tag may contain one or more matched words, which are separately marked with Term Tags.

If the query in the document request included a near or and clause, and the entire range of the hits

spans multiple XML tags in the document, the More Tag will mark any parts of the range that occur in subsequent XML tags.

- <xtf:term>

  This tag identifies a single word in the main text that matches a query associated with the requested document. This tag has the form:

```
<xtf:term>MatchedWord</xtf:term>
```

  where

`xtf:`	is an XTF namespace prefix added to the tag to differentiate it from tags that came from the original XML document. The prefix will not be present in term tags that appear within the initial snippets summary tag, but only in term tags that occur in the main text for the document. The namespace URI for this prefix is: `http://cdlib.org/xtf`

Error Generator

The purpose of the **Error Generator** stylesheet is to generate a web-page that displays user friendly messages when dynaXML errors occur. Since this stylesheet works the same way in both dynaXML and crossQuery, it is documented on the common Error Generator Stylesheets page.

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