HyREX

Manual

Norbert Gövert
<goevert@ls6.cs.uni-dortmund.de>

Kai Großjohann
<grossjohann@ls6.cs.uni-dortmund.de>

February 26, 2002
## Contents

1. Introduction 4
   1.1. The XML Document Model 4
      1.1.1. Data Types and Vague Predicates 5
   1.2. Multi-level Hypertext 6
   1.3. HyREX Architecture 6
   1.4. Download und Installation 7

2. Administration 8
   2.1. Index Structure Overview 8
   2.2. Document Definition Language 8
      2.2.1. The `<hyrex>` Element 9
      2.2.2. The `<access>` Element 10
      2.2.3. The `<convert>` Element 13
      2.2.4. The `<summary>` Element 14
      2.2.5. The `<datatype>` Element 16
      2.2.6. The `<inodes>` Element 17
      2.2.7. The `<structure>` Element 18
   2.3. The `hyrex_index` Indexer 18
   2.4. Command-line Search Tool 19
      2.4.1. Invoking `hyrex_search` 19
      2.4.2. Using `hyrex_search` 20
   2.5. HyREX Server 21
      2.5.1. Invoking and Stopping the HyREX Server 21
      2.5.2. The HyREX Server Protocol 21
   2.6. HyGate Server 24
      2.6.1. The Config File 25
      2.6.2. The Query Form 26
      2.6.3. The Summary Stylesheet 29
      2.6.4. The Document Stylesheet 30
      2.6.5. Invoking HyGate 30

3. Users 31
   A. Paths and Path Expressions 32
      A.1. Paths 32
<table>
<thead>
<tr>
<th>Contents</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.2. Path expressions</td>
<td>32</td>
</tr>
<tr>
<td>B. DTD for HyREX DDL</td>
<td>34</td>
</tr>
<tr>
<td>List of Tables</td>
<td>40</td>
</tr>
<tr>
<td>List of Figures</td>
<td>41</td>
</tr>
<tr>
<td>Bibliography</td>
<td>42</td>
</tr>
</tbody>
</table>
1. Introduction

XML is the emerging standard for representing knowledge in almost arbitrary applications. At least almost every kind of knowledge can be represented in XML. For exploring such knowledge, one needs a search engine which is able to let users benefit from all of the concepts with which XML blesses the world.

HyREX is the Hyper-media Retrieval Engine for XML. Hyper because it offers explicit and implicit hyperlinks to the user. Media because it offers search facilities for text but also for other media than text, at least conceptually. Retrieval engine because it allows users to explore all kinds of information structures available through XML, not only plain document retrieval. XML because it allows retrieval under consideration of content and structure inherent in XML documents.

In order to discuss HyREX’s capabilities we will first briefly describe the concepts where HyREX is based upon. We start with the description of the XML document model in Section 1.1, continue with a few words on the multi-level hypertext concept in Section 1.2. Finally we give a brief sketch on the architecture of HyREX, which enables us to outline what a HyREX application administrator needs to do in order to set up an application.

1.1. The XML Document Model

Figure 1.1 gives an overview on the document model induced by XML.

An XML document base consists of a number of document classes. Each document class has its own DTD, where each document instance of that class must conform to it. Besides the content XML documents display their logical structure by means of hierarchically organized markup.

Parts of documents can be linked to other parts of arbitrary documents, be the referenced part in the same document, in a document of the same class, or in a document of some other class.

HyREX enables users to query XML documents not only by their content but also by their logical and link structure. In addition, HyREX makes use of structure in relevance-oriented searches: It aims at retrieving those parts of the documents which are most relevant w. r. t. the users information need, i. e. the granularity of searches is refined.

\[\text{http://www.w3c.org/XML/}\]

\[\text{HyREX demands documents to be valid rather than XML conform only.}\]
1.1. THE XML DOCUMENT MODEL

1.1.1. Data Types and Vague Predicates

Markup in XML documents does not only display the logical structure of the documents. Often it provides additional semantic information. Most information in the documents can be assigned data types, i.e. the data originates from a certain domain. This information can be exploited at retrieval time: for a given data type special search predicates can be provided by HyREX. Assignment of data types to specific document parts can be done by means of the DTD, as illustrated in figure 1.2. The current set of data types available in HyREX is displayed in figure 1.3. A detailed description of data types and their respective search predicates can be found in Section ??.

Having the concept of data types and their respective vague predicates implemented allows us to extend HyREX in a flexible way. Additional data types for special applications are easily integrated into HyREX’ object-oriented design. Even data types for other media than text can be integrated. {add examples}.

Figure 1.1.: The XML Document Model

Figure 1.2.: Mapping of data types and documents
1.2. Multi-level Hypertext

Information in documents can be viewed in structures on different levels. On the bottom level there is the full text of a document. If we consider reference databases, most often the full text is not part of the document base, but a description of a document, a so-called metadata record bears the knowledge about a document, which is presented to the user. If we consider domains of certain attributes of this description, we view knowledge on the attribute level. Considering the structure of a given document base, we are arrived at the schema level.

There are various relations between the different levels and also between elements at the same level. These relations can serve the user for navigating within the information space of a given document base.

In addition to search facilities on any of these levels HyREX will provide these links in order to enhance the user’s ability to get information out of a document base.

1.3. HyREX Architecture

Figure 1.4 displays HyREX’s architecture.

On the top-most level the user contacts HyREX by means of an arbitrary Web browser. Information needs issued through the Web browser are accepted by HyGate. It converts the user’s request into a XIRQL query and delegates the processing to the lower levels of HyREX; the results are properly presented to the user.

On the conceptual level, XIRQL queries are accepted and processing. Whenever access paths are needed in order to further process a query, this request is handed to the physical level, which is named HyPath. On the physical level, there are a number of access paths for each datatype and predicate given in the XML documents.

The task of the document base administrator can be described by means of HyREX’s
different levels:

**HyGate** Describe the layout for search results and documents. This is done by specifying XSL stylesheets (see also Section 2.6).

**XIRQL** Specify data types of the various parts of documents by means of the DTD. This is done within a so-called *document definition language* (DDL) which is to be prepared for each document class. Section 2.2 describes how to do that.

**HyPath** Specify access structures for predicates and the structure of documents. This is also done within a DDL instance. See Section 2.2.

### 1.4. Download und Installation

The latest version of this software can be fetched (along with other software which might be of interest for you) from our [FTP server](ftp://ls6-ftp.cs.uni-dortmund.de/pub/projects/carmen/). A detailed installation manual [is available as well](ftp://ls6-ftp.cs.uni-dortmund.de/pub/projects/carmen/INSTALL), describing the installation process from scratch (i.e. operating system and C compiler available).
2. Administration

This chapter describes the configuration of the HyREX system. It assumes that you have already installed the required software, so HyREX is ready to run. It also assumes that you’ve already got some XML documents which you would like to index.

From there, the first thing to do is to tell the system how to index the documents. For this, we have the so-called ‘data definition language’ (DDL). So we need to describe what a DDL file looks like. Next, we describe how to start the indexer.

When you have indexed your document collection, you might want to issue a few queries. For this, there is a simple interactive command-line tool, which is described in Section 2.4.

There is also a HyREX Server which allows for (almost) arbitrary frontends. We describe how to set that up and how to run it in Section 2.5. There we also describe the protocol used by the server.

Finally, there is a simple Web frontend for HyREX, called HyGate, which is described in the last section of this chapter.

2.1. Index Structure Overview

A document collection is called a “base”. Inside it, there may be several “classes”; a class corresponds to a set of documents all conforming to the same DTD. In a class, there are several “datatypes”, each datatype provides several search “predicates”.

If you use relative file names, please note that they will be relative to the current working directory that’s in effect when you start the indexer hyrex_index.

2.2. Document Definition Language

For indexing a given set of documents, you need the documents themselves (of course), their respective DTDs (one for each of your document classes) and a DDL (data definition language) file for each of your documents classes. This DDL file tells the HyREX indexer how to index the documents. HyREX comes with a DTD in the file doc/hyrex.dtd which describes the format of the DDL files (which are XML files). You will also find a copy of the DTD in Appendix B. In this section, we explain that format in more detail. We also explain how to run the indexer.

We describe the DDL format in a top-down manner. We begin with the top-level element.
2.2. DOCUMENT DEFINITION LANGUAGE  CHAPTER 2. ADMINISTRATION

2.2.1. The <hyrex> Element

A DDL file looks like this:

```xml
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE hyrex SYSTEM ".../doc/hyrex.dtd">
<hyrex attributes>
  <access attributes> ... </access>
  <convert attributes> ... </convert>
  <summary attributes> ... </summary>
  <datatype attributes> ... </datatype>
  <inodes> ... </inodes>
  <structure> ... </structure>
</hyrex>
```

In the first line, you might also wish to use a different encoding, for example `utf-8`.

In the second line, the ellipsis indicates that you need to put in the right path name on your local system for the given file (you need that if you wish to validate your DDL file).

The other ellipses indicate where further text or XML elements were left out; the text ‘`attributes`’ means that some XML attributes were left out. We describe the attributes of the `<hyrex>` element here, the other attributes are described with their elements.

The `<datatype>` and `<structure>` elements may occur multiple times, the `<convert>` and the `<inodes>` element are optional. The other elements must be there exactly once.

The attributes of the `<hyrex>` element are:

- **directory** This gives the directory where the index lives. In this directory, HyREX creates a a directory named after the document base.

- **base** This string gives the name of the document base. It is also used as a name for the directory where the index files for your various document classes live.

- **class** The name of the document class to create within your document base.

- **dtd** All documents of a given class to be indexed must comply with a DTD. Its file name is given here.

All attributes are required. Example:

```xml
<hyrex directory="/tmp/hyrex"
  base="example"
  class="books"
  dtd="/tmp/books.dtd">
  ...
</hyrex>
```

In this case, HyREX will create a file `/tmp/hyrex/example/meta` listing all classes, and the directory `/tmp/hyrex/example/books` contains a subdirectory for each data type.
2.2. The <access> Element

This element must be present exactly once in a DDL file. It tells HyREX where to find the documents of a document class. This element has one attribute, classname, which refers to a HyREX document access class implementing a method to access documents in a certain way. The various classes available in HyREX are described below. The <access> element content consists of parameters used to configure the referenced access method. The allowed parameters and their meanings depend on the value of the attribute classname and are described below. Parameters are given as (name, value) pairs, syntactically as name and value attributes of element <parameter>. Parameters may be set-valued. Set-valued parameters are specified in several <parameter> elements, each having the same name value. For each parameter it is denoted whether it is mandatory or optional.

Here is a list of currently available classes and the corresponding element content:

HyREX::HyPath::Document::Access::XMLstream This document access class extracts subtrees of XML files. Each such subtree is considered to be a document in its own right. The class is configured with two parameters. Usage:

<access classname="HyREX::HyPath::Document::Access::XMLstream">
  <parameter name="element" value="article"/>
  <parameter name="files" value="/tmp/a/*.xml"/>
  <parameter name="files" value="/tmp/b.xml"/>
</access>

The element parameter (mandatory) gives an element name. Each element with that name is considered to be the root element of a document. You can provide more than one root element specifications.

The files parameter (mandatory, set-valued) gives shell glob patterns which determine the files that will be part of this document class. The question mark ? and the asterisk * can be used as wildcards, with the usual Unix shell glob semantics. Relative directory names are expanded to the respective absolute paths.

HyREX::HyPath::Document::Access::Tar This document access class extracts files from tarballs (*.tar and *.tar.gz files). The constructor requires two or more parameters. Usage:

<access classname="HyREX::HyPath::Document::Access::Tar">
  <parameter name="expression" value="$_[0] =~ m/^a/"/>
  <parameter name="files" value="/tmp/one.tar.gz"/>
  <parameter name="files" value="/tmp/two.tar"/>
</access>
In the expression parameter (optional) a Perl expression may be specified. The files parameter (mandatory, set-valued) contains tarball file names (shell globs are allowed again, relative directory names are expanded to the respective absolute paths).

For each tarball, the access class goes through all the files stored inside and evaluates the Perl expression with each file name thus obtained (the Perl variable \$_[0] is set to the filename under consideration). The file is skipped unless the Perl expression returns true. If no expression is given, no file is skipped.

**HyREX::HyPath::Document::Access::Find**  This access class recursively finds files in directories. The class is configured via two parameters. Usage:

```xml
<access classname="HyREX::HyPath::Document::Access::Find">
  <parameter name="expression" value="$_[0] =~ m/^a/"/>
  <parameter name="directories" value="/tmp/one"/>
  <parameter name="directories" value="/tmp/two/*/"/>
</access>
```

In the expression parameter (optional) again a Perl expression with the same meaning as described in the previous description item may be specified. Directories parameters (mandatory, set-valued) contain directory names with shell glob patterns. Relative directory names are expanded to the respective absolute paths.

The access class expands the shell glob patterns and recursively walks each directory thus obtained. For each file that is found this way, the Perl expression is evaluated (the Perl variable \$_[0] is set to the full directory/file name under consideration, so you can use this variable in the expression). The file is skipped unless the Perl expression returns true. If no expression is given, no file is skipped.

**HyREX::HyPath::Document::Access::Split**  This access class splits files according to regular expressions. Each part is considered a document. The class is configured with three parameters. Usage:

```xml
<access classname="HyREX::HyPath::Document::Access::Split">
  <parameter name="regexp" value="^From ">
  <parameter name="mode" value="start"/>
  <parameter name="files" value="/tmp/one"/>
  <parameter name="files" value="/tmp/two/*/"/>
</access>
```

The three different parameters are as follows: The files parameters (required) are interpreted as file names with shell glob patterns. Relative directory names are expanded to the respective absolute paths. The resulting files are read and split.
according to the Perl regular expression given in the \texttt{regexp} parameter (required). The \texttt{mode} parameter (optional) says how to handle the text that was matched by the regexp. If the mode is \texttt{'end'}, the matched text is part of the document before it. If the mode is \texttt{'start'}, the matched text is part of the following document. If the mode is \texttt{'skip'}, the matched text is not considered part of any document. This is also the default behaviour.

\textbf{HyREX::HyPath::Document::Access::Nnfolder} This access class can be used to index \texttt{nnfolder} groups of Emacs Gnus. The class is configured with one parameter. Usage:

\begin{verbatim}
<access classname="HyREX::HyPath::Document::Access::Nnfolder">
  <parameter name="files" value="/Mail/archive/archive"/>
  <parameter name="files" value="/Mail/archive/old/*"/>
</access>
\end{verbatim}

\begin{verbatim}
<convert classname="HyREX::HyPath::Document::Convert::Mail">
  <parameter name="encoding" value="iso-8859-1"/>
</convert>
\end{verbatim}

The \texttt{files} parameters (required, set-valued) are interpreted as file names with shell glob patterns. The resulting files are read and split. Because HyREX needs XML as data you must supply the desired converter, normally \texttt{HyREX::HyPath::Document::Convert::Mail}. See in the Section[2.2.3] for details about how to configure the converter.

\textbf{HyREX::HyPath::Document::Access::IMAP} This access class can be used to index mails on an IMAP server. The class is configured with atleast two parameters. Usage:

\begin{verbatim}
<access classname="HyREX::HyPath::Document::Access::IMAP">
  <parameter name="server" value="imap-server"/>
  <parameter name="folders" value="*"/>
  <parameter name="user" value="hyrex"/>
  <parameter name="passwd" value="hyrex"/>
</access>
\end{verbatim}

\begin{verbatim}
<convert classname="HyREX::HyPath::Document::Convert::Mail"/>
\end{verbatim}

The \texttt{server} parameter (required) contains the name of your IMAP server. The \texttt{folders} parameter (required, set-valued) is interpreted as a single folder name or a reference to a list of folder names taken as input. Each name could contain the wildcard symbols \texttt{'*'} and \texttt{%} which will be interpreted by the IMAP server. The \texttt{%} will match only one subfolder, whereas the \texttt{'*'} will match anything. Each already read mail found in the (sub)folders specified will be indexed.
The user parameter (optional) contains the username to use for authentication against the IMAP server. If this parameter is missing it will be tried to guess it by first trying to get it from `.authinfo` or `./.authinfo`. When this fails it will use the username or anonymous if possible. You can force anonymous login by specifying anonymous or anyone here.

The passwd parameter (optional) contains the password to use for authentication against the IMAP server in plaintext. If you omit the parameter it will be tried to get it from the `.authinfo` or `./.authinfo` file. When this fails it will use anonymous login if possible.

For details about the format of the `.authinfo` file see the man page of `HyREX::HyPath::Document::Convert::Mail` or the Emacs Gnus info page.

Because HyREX needs XML as data you must supply the desired converter, normally `HyREX::HyPath::Document::Convert::Mail`. See in the Section 2.2.3 for details about how to configure the converter.

Further access classes can be provided by subclassing the respective abstract class `HyREX::HyPath::Document::Access`.

### 2.2.3. The `<convert>` Element

This optional element is allowed to be present exactly once in a DDL file. It tells HyREX that your documents must be converted to XML. This element has one attribute, classname, which refers to a HyREX document convert class implementing a method to convert documents in a certain way. The various classes available in HyREX are described below. The `<convert>` element content consists of parameters used to configure the reference convert method. The allowed parameters and their meanings depend on the value of the attribute classname and are described below. Parameters are given as (name, value) pairs, syntactically as name and value attributes of element `<parameter>`. Parameters may be set-valued. Set-valued parameters are specified in several `<parameter>` elements, each having the same name value. For each parameter it is denoted whether it is mandatory or optional.

Here is a list of currently available classes and the corresponding element content:

**HyREX::HyPath::Document::Convert::Mail** This document convert class accepts RFC822 messages as input. The class is configured with two parameters. Usage:

```xml
<convert classname="HyREX::HyPath::Document::Convert::Mail">
  <parameter name="xmlversion" value="1.0"/>
  <parameter name="encoding" value="iso-8859-1"/>
</convert>
```

The xmlversion parameter (optional) gives the XML version, which will be written into the XML header line. The default value is ‘1.0’.
2.2. DOCUMENT DEFINITION LANGUAGE  CHAPTER 2. ADMINISTRATION

The encoding parameter (optional) gives the XML encoding, which will also be written into the XML header line. The default value is 'UTF-8'.

Attachments which are not of MIME type text/plain or message/rfc822 will be ingored for the XML output.

The resulting XML document is described by the DTD found in your HyREX source tree under app/mail/mail.dtd.

2.2.4. The <summary> Element

Internally in HyREX, a query result is a weighted list of paths, where each path describes a node (XML element or XML attribute, usually) in an XML document. Paths look like /book[3]/chapter[1] (first chapter in third book document). Further details on paths are given in Appendix A. Clearly, such a path is not useful for the user. Therefore, HyREX defines a so-called ‘summary’ for each document. A summary is supposed to contain information that helps the user to identify the document. Summaries are automatically extracted from the XML documents according to the rules given in the <summary> element in the DDL. For example, for book summaries the elements title, author, year, and perhaps publisher might be useful.

Document summaries can be specified via custom extraction rules or via XSL(T) stylesheets.

Summary generation via custom extraction rules

A corresponding <summary> element looks like this:

```xml
<summary>
  <element name="author">
    <element name="last">
      <query query="/book/au/ln"/>
    </element>
    <element name="first">
      <query query="/book/au/fn"/>
    </element>
  </element>
</summary>
```

An element <query query="pathexp"/> processes the given path expression query and inserts the result into the summary. See Appendix A on the definition of path expressions. The attributes of the <query> element are:

query This specifies the path expression to process. This attribute is mandatory.

structure This attribute may have only two values, yes or no. This attribute is optional and defaults to no if omitted.
If the value is no, the result of the path expression is flattened into a string, and this string is inserted into the summary.

If the value is yes, the subtree selected by the path expression is taken as is and inserted into the summary including all start and end tags.

An element `<element name="foo">children</element>` inserts a `<foo>` start tag, then processes the given children, then a `</foo>` end tag.

Note that this method is quite limited. For instance, in the above example, consider what happens if the document has two authors, Mark Smith and John Doe. Then the summary will look like this:

```xml
<author>
  <last>Smith Doe</last>
  <first>Mark John</first>
</author>
```

This is probably not what you intended.

**Summary generation with XSL(T)**

Here, you specify an XSLT stylesheet which is applied to the document. The output of the stylesheet will be used as the summary. You can either supply the stylesheet in the DDL file, or you can put a filename in the DDL file which points to the stylesheet to use.

If you want to supply the stylesheet in the DDL file, the `<summary>` element looks like this:

```xml
<summary>
  <xsl>
    <![CDATA[
      <!DOCTYPE xsl:stylesheet [<![CDATA[
        <?xml version="1.0"?>
        <xsl:stylesheet ...>
        ...
        </xsl:stylesheet>
      ]]>]]>
    </xsl>
  </summary>
```

The stylesheet has been enclosed in a CDATA section to avoid having to quote special characters.

If you want to supply the stylesheet in a separate file, the `<summary>` element looks like this:

```xml
<summary>
  <xslfile name="/tmp/foo.xsl"/>
</summary>
```

Here, the `name` attribute gives the name of the file which contains the stylesheet.
2.2.5. The `<datatype>` Element

A data type in HyREX specifies which search predicates can be used in a query. This has an impact on the kinds of queries that users can formulate.

The `<datatype>` element looks like this:

```xml
<datatype classname="Class::Name::Goes::Here">
  <parameter name="foo" value="42">
    <query query="/some/path/expression"/>
  </parameter>
  <parameter name="bar" value="4711">
    <query query="/other/path/expression"/>
  </parameter>
</datatype>
```

The `<parameter>` subelement is optional. The `<query>` subelements are mandatory. The `<query>` may occur multiple times.

The `classname` attribute of the `<datatype>` element gives a Perl class name which implements this attribute. The `<parameter>` subelement specifies parameters to be used for configuring the datatype.

The `<query>` subelements specify path expressions. See Appendix A on the definition of path expressions. All document elements matching one of the queries will be part of the regions covered by the data type and therefore, after indexing, will be searchable by predicates available for the data type under consideration.

{The following section is still not complete and will be revised soon (NG.)}  !!!

The current HyREX implementation provides the following data types to be used in the `classname` attribute. We also specify which parameters are needed (if any) for that class.

All classes know {and require?} the parameter `filter`. {What does it do?}  !!!

HyREX::HyPath::Datatype::Name This class can be used for indexing person names. Only parameter `indexfilter` is a list of filters to apply for indexing. Example:

```xml
<datatype classname="HyREX::HyPath::Datatype::Name">
  <parameter name="indexfilter" value="latin1_tr"/>
  <parameter name="indexfilter" value="latin1_lc"/>
  <parameter name="indexfilter" value="split1"/>
  <parameter name="filter" value="latin1_tr"/>
  <parameter name="filter" value="latin1_lc"/>
  <parameter name="filter" value="split1"/>
...</n</datatype>
```

HyREX::HyPath::Datatype::Text This class can be used for indexing text. It presupposes that text is a sequence of words, and words are separated from each other by whitespace and/or punctuation characters. (This means that this class is not appropriate for Chinese, say.) It knows the usual `filter` and `indexfilter` parameters. Example:
2.2. DOCUMENT DEFINITION LANGUAGE  CHAPTER 2. ADMINISTRATION

HyREX::HyPath::Datatype::Text::English This class can be used for English text. It knows the usual filter and indexfilter parameters. Example:

```xml
<datatype classname="HyREX::HyPath::Datatype::Text::English">
  <parameter name="indexfilter" value="latin1_tr"/>
  <parameter name="indexfilter" value="latin1_lc"/>
  <parameter name="indexfilter" value="split2"/>
  <parameter name="filter" value="latin1_tr"/>
  <parameter name="filter" value="latin1_lc"/>
  <parameter name="filter" value="split2"/>
  <parameter name="filter" value="stop"/>
  ...
</datatype>
```

The filter and indexfilter Parameters

Type `perldoc HyREX::HyPath::Filter` for a list of available filters.

When text is read from the XML document, it first exists as a string. It needs to be converted into items that a user can search for. For example, for text this normally means words. So when HyREX reads a string from an XML document, it invokes all the functions specified in the indexfilter {filter?} parameter and out comes a list of items to insert into the index.

{More explanation about the other parameter.}

2.2.6. The <inodes> Element

Within the optional <inode> element one can specify so called index nodes. Index nodes are such nodes are the roots of subtrees in XML documents which serve as valid answer w.r.t. relevance oriented retrieval requests.

Index nodes (of course a document may have more than one index node, the root of a given document always is an index node) are specified by means of path expressions. See Appendix A on the definition of path expressions. In the following example all 'section'
nodes in the documents are treated as index nodes (in addition to the root node of the
document):

\[
<\text{inodes}>
\begin{align*}
&\text{<query query="/\text{section}"/>}
\end{align*}
</\text{inodes}>
\]

### 2.2.7. The `<structure>` Element

The class specified in the `<datatype>` elements say how to index the *values* stored in
certain regions of the XML documents. The classes specified in the `<structure>` element,
however, say how to index the *structural information* in the XML documents.

A `<structure>` element looks like this:

\[
<\text{structure} \text{classname}="\text{Class::Name::Goes::Here}">
\begin{align*}
&\text{<parameter name="foo" value="42"/>}\n&\text{<parameter name="bar" value="4711"/>}\n\end{align*}
</\text{structure}>
\]

The `classname` attribute is the name of the class that implements this structure. This
attribute is mandatory.

Currently, HyREX supports only one class for indexing the structural information:

**HyREX::HyPath::Structure::Tree** Example:

\[
<\text{structure} \text{classname}="\text{HyREX::HyPath::Structure::Tree}"
\begin{align*}
&\text{<parameter name="compress" value="10"/>}\n\end{align*}
</\text{structure}>
\]

This class builds an external access path where the structural information for each
document is stored separately. This class knows one parameter `compress` which
specifies the effort used to determine an optimal compression for the structural
information. Unfortunately the value depends on the number of documents, the
range of values is from 1 to the number of documents you want to index. (HyREX
does not know this figure before being finished with indexing, therefore it is not
possible to provide a relative value or a percentage.) As a rule of thumb one can say
that the value might be quite low if your documents all share a similar structure.

### 2.3. The hyrex_index Indexer

After you have composed a DDL file, you need to run the `hyrex_index` program to
actually index the XML documents. This is rather simple, and can be done like this:

```bash
hyrex_index -ddl /tmp/books.xml
```

This command assumes that your DDL file is stored on disk in the file `/tmp/books.xml`. 

The `hyrex_index` command can also be used to dispose a base or a class specified within a given DDL file:

```
hyrex_index -ddl /tmp/books.xml -dropclass
hyrex_index -ddl /tmp/books.xml -dropbase
```

Additional options can be taken from the `hyrex_index` manual page.

### 2.4. Command-line Search Tool

For testing purposes it might be useful to be able to just issue XIRQL queries and see what’s the result. Therefore, we have the simple command-line tool `hyrex_search` which provides this.

#### 2.4.1. Invoking `hyrex_search`

In addition to `-help` and `-version`, the program understands the following options:

- `-directory dir` The database is stored in the given directory. This should be the same as the `directory` attribute of the `<hyrex>` element of the DDL file.
  
  When using relative directories, beware of current working directory!

- `-base base` Searches in the given document base. Should be the same as the `base` attribute of the `<hyrex>` element of the DDL file.

- `-class class` Searches in the given document class. Should be the same as the `class` attribute of the `<hyrex>` element of the DDL file.

Suppose you have a DDL file `/ddl/filename.xml` which looks like this:

```xml
<?xml version="1.0" encoding="iso-8859-1" ?>
<!DOCTYPE hyrex SYSTEM ".../doc/hyrex.dtd">
<hyrex directory="/tmp/hyrex"
  base="example"
  class="articles">
  ...
</hyrex>
```

With this DDL file, the following invocation would be right:

```
hyrex_search -dir /tmp/hyrex -base example -class articles
```

You should see a file `/tmp/hyrex/example/meta` and a directory `/tmp/hyrex/example/articles` with subdirectories for each data type.

Alternatively the following call would be right:

```
hyrex_search -ddl /ddl/filename.xml
```
2.4.2. Using hyrex_search

This is a simple command-line tool, similar to a shell. It displays a prompt and knows a number of commands. The most important commands are: find, for issuing a XIRQL query; document, for viewing a result document; part, for viewing the part of a document that was found; and of course, quit, for exiting the program.

There is some support for command-line editing, similar to bash(1).

Command list:

**help** Displays a summary of available commands.

? What might it do?

**quit** Exits the program.

**exit** Quits the program.

**find** *xirql* Execute the given XIRQL query and show the resulting ranking list. (Sorted by decreasing score.)

**find** Show again the result of the last query.

The output of find is a sequence of lines, each containing three fields:

```
1: 0.987 /project[14]
```

In general, it’s an index (line number in result list), followed by a colon and a space, then the score, then a space, then the path. The first line has index number 0.

**document number** The number specifies an item in the ranking list. Shows the corresponding document.

**part number** Shows only the part (subtree) of the document which was found relevant to the query. Note that different subtrees of the same document might be in the ranking list on various positions.

**summary number** Show the summary for the given document.

**debug** Toggle debugging. Debugging means that you can see how HyREX is processing the query.

**debug 0** Turn debugging off.

**debug 1** Turn debugging on.

**beautify** Toggle beautification. This means that the result ranking list shows the processing path in addition to the result path for each item. It also means that some highlighting is used for the document and part commands.

**beautify 0** Turn beautification off.

**beautify 1** Turn beautification on.
2.5. **HyREX Server**

This module can be used for constructing arbitrary user interfaces. We have used it for simple web interface; see section 2.6 below. The protocol used by the HyREX Server is similar to SMTP and NNTP, but of course the actual verbs and error codes used are different.

Below, we describe how to start and stop the server, and then we describe the protocol that’s used.

### 2.5.1. Invoking and Stopping the HyREX Server

Running the server is fairly simple: you only need to know the directory where the database index is stored, and the port that the server should listen on. For example:

```bash
perl -MHyREX::Server -e 'server("/tmp/hyrex", 4711)'
```

In this case, the databases are stored under `/tmp/hyrex` and the port number is 4711. If you are running this command on the host `marvin`, you can connect to this server with the following command:

```
telnet marvin 4711
```

It is equally easy to stop the server: just kill the process.

{Do we need a script which does this? Maybe like apachectl?}

### 2.5.2. The HyREX Server Protocol

{We need to say something about the EOL format!}

The protocol is similar to SMTP and NNTP. The server listens for new connections. When a client connects to the server, the server prints a greeting message. Client requests consist of a single line each. The server responds with a status line, followed by some data (possibly of length zero), followed by the end-of-data indicator.

The status line consists of a status code, followed by whitespace, followed by a textual message.

The client request consists of a verb, followed by a space, followed by an argument.

(Some requests consist of only the verb.)

In the following subsection, we list all request verbs, along with a description of the argument (if applicable). In addition, a list of possible response status codes is provided.

The subsequent subsection contains a list of status codes, together with a description of their meaning and the data.

#### Request Verbs

A client request consists of a single line of text. The line begins with a request verb, optionally followed by whitespace and an argument.

**help**  Prints a short usage message. Status code is 100.
quit  Terminates the connection. Status code is 205.

open  base  class  Opens the given database base and class class. The two are separated
by whitespace. Status codes 201, 202, 401.

datatypes  Prints a list of datatypes available in the current base/class. Status
codes 209, 406.

predicates  datatype  Prints a list of search predicates available for the specified datatyp.
Status  codes 210, 408, 501.

hits  number  The number can be omitted. If the number is given, sets the number of
hits to be displayed to the given number. If the number is 0, set to unlimited. If
the number is omitted, print the value currently in effect. Status codes 202, 204,
501.

find  query  The query can be omitted. If the query is given, processes the given query
and print some results. The number of results to print is specified by the hits
value, see above. If the query is omitted, print the results from the most recent
query again. (Only from the current connection.) Status codes 203, 404.

datatypevalues  datatype  Prints a list of values for the given datatype. Currently not
implemented. Status codes 212, 410.

document  docspec  Prints the specified document. Docspec can be a number, which is
an index into the result list (starting at 1), or a path (like /book[1]). Status codes
206, 412.

summary  docspec  Prints a summary for the specified document. Docspec same as for
document. Status code 207. {This looks suspicious. Is error checking missing in the
implementation?}  
!!!

docid  docspec  Prints the external document id for the specified document. Docspec
same as for document. The external document id can be used for locating the
real document. {What exactly is that? File name? URL? Can it be used outside of
HyREX?} Status code 208. {Is error checking missing in the implementation?}  
!!!  
!!!

Status Codes and Result Data

The server response consists of a status line, followed by some data, followed by an end-
of-data indicator. A status line consists of a number, followed by whitespace, followed
by a short text.

The end-of-data indicator is a line consisting only of a dot (.). To prevent ambiguity
in case the data contains such a line, dot stuffing is used. This means that any line in
the data which starts with a dot gets another dot prepended.

Thus, clients must remove one dot from lines which start with a dot.

The following description does not mention dot stuffing explicitly.
2.5. HYREX SERVER

100 Generated by help. Result data is some human-readable text.

201 Generated by open base class. Result data is empty. Server has selected the given database and class.

202 Generated by hits 0. Result data is empty. Server will print all hits from now on.

203 Generated by find. Result data is a sequence of lines. Each line contains three tab-separated fields. The first field is an index (which can be used for the document, summary and part commands). The second field is the score. The third field is the path for the XML subtree found.

204 Generated by hits number. Result data is empty. Server will print the given number of hits from now on.

205 Generated by quit. Result data is empty. Server closes connection after sending end-of-data indicator to client.

206 Generated by document docspec. Server prints the document specified as argument. Result data is an XML document.

207 Generated by summary docspec. Server prints the summary for the document specified as argument. Result data is an XML document.

208 Generated by docid docspec. Server prints the external document id. {Is this implemented?}

209 Generated by datatypes. Result data is one line with list of datatypes, separated by whitespace.

210 Generated by predicates datatype. Result data is one line with list of predicates available for the given data type, separated by whitespace. Note that the predicate name must be surrounded by dollar signs when used in a XIRQL query.

212 Generated by datatypevalues. Server prints list of values of this datatype. Result data is an XML fragment. The top-level element is <datatypevalues> which has one attribute name which contains the name of the datatype. For each value, the top-level element has one value child which has the value as its content. Example:

```
<datatypevalues name="Bogus::Type::Number">
  <value>one</value>
  <value>two</value>
</datatypevalues>
```

{Build a real XML document? What about escaping funny characters?}

401 Generated by open. {Fishy!}
2.6. HyGate Server

HyGate is a simple Web frontend for HyREX. It is implemented as a small Web server in Perl. The server can show an HTML query form and process the given queries. For query processing, it talks to the HyREX Server. The XML data returned by that server is processed with XSL(T) stylesheets to produce a ranking list, as well as to show the document.

HyGate uses the following configuration files:

**Config file** The config file specifies the location of the other files, as well as a few other parameters.

**Query form** This is an HTML file and is displayed when the user visits the home page of the server.

**Summary stylesheet** This is an XSL(T) file which is invoked after processing a query to display the result list.

**Document stylesheet** Also an XSL(T) file. This one is invoked when the user clicks on a document in the result list.

The distribution contains a file `t/data/config/common.xsl` which provides a few named templates which are useful in the summary and document stylesheets.

In the following, we will discuss each configuration file. We finish this section with a description how to invoke and stop the HyGate Server.
2.6.1. The Config File

Here is an example of a config file:

```xml
<!DOCTYPE hygate SYSTEM "../dtd/config.dtd">
<hygate>
  <query_form>t/data/config/projects.html</query_form>
  <gateport>8080</gateport>
  <host>localhost</host>
  <port>4055</port>
  <database>BASE</database>
  <class>projects</class>
  <maxhits>100</maxhits>
  <prefix>/foo</prefix>
  <xslt_summary>t/data/config/projects_summary.xsl</xslt_summary>
  <xslt_document>t/data/config/projects_doc.xsl</xslt_document>
  <cache_root>tmp</cache_root>
  <cache_expire>300</cache_expire>
</hygate>
```

To create your own config file, just copy the above and replace the values. Here is the meaning of the values:

- **query_form**: This element contains the file name of the query form.
- **gateport**: HyGate is a Web server and listens on this port.
- **host**: HyGate expects the HyREX Server to be running on this host.
- **port**: HyGate expects the HyREX Server to be running on this port.
- **database**: Queries are processed w.r.t. this document collection.
- **class**: Queries are processed w.r.t. this document class.
- **maxhits**: HyGate never retrieves more than this number of query results from the server. This means that there is no way for the user to see more than this number of results!
- **prefix**: Used for constructing the XIRQL query passed to HyREX. See below for an explanation of the queries generated.
- **xslt_summary**: The file name of the summary stylesheet.
- **xslt_document**: The file name of the document stylesheet.
- **cache_root**: The root directory for the query result cache. HyGate will create a subdirectory `hygate-cache` under this directory and will put the data there.
cache_expire  The number of seconds before objects in the cache expire.

Note that relative file names will be interpreted relative to the current working directory that's in effect when they HyGate Server is started!

2.6.2. The Query Form

In an HTML query form, there can be a number of input fields. The question is, how to map this fairly flat structure onto the complex structure of XIRQL queries. This is done in a simple manner; only a fairly narrow class of XIRQL queries can be issued with HyGate. Here is an example of a XIRQL query that's possible with HyGate:

/book[title $stemen$ "retrieval" $or$ author $soundex$ "fuhr"]

In general, such a query will consist of a prefix (here /book) followed by square brackets. Inside the square brackets there is a list of clauses separated by $or$. Here, there are two clauses, title $stemen$ "retrieval" and author $soundex$ "fuhr".

Each clause is a triple consisting of a path condition, a search predicate, and a comparison value (the last one is enclosed in double quotes). For example, the clause title $stemen$ "retrieval" has title as the path condition, $stemen$ as the search predicate and retrieval as the comparison value.

The above explanation is a bit simplified. Actually, it is possible for the user to enter several words into each search field. A word may begin with the + character, which indicates a mandatory condition, whereas the other conditions are optional. There are several methods for generating a query from the user input.

wsum  The “wsum” method constructs a weighted sum from the user input, for example:

/book[ wsum(1.0, title $stemen$ "retrieval", 5.0, author $soundex$ "fuhr") ]

Here, query conditions marked as mandatory by the user (via +) are given the weight 5.0 whereas the normal query conditions are given the weight 1.0. (HyREX will then normalize the weights internally such that they sum up to one.)

This method has the disadvantage that it might return documents for which none of the mandatory query conditions are fulfilled. However, if any mandatory query condition is fulfilled, then the corresponding document will appear near the top of the ranking list.

strict_bool  The “strict_bool” method constructs a nested Boolean expression from the user input, for example:

/book[ ( title $stemen$ "retrieval" $and$ title $stemen$ "information" ) $and$ ( author $soundex$ "fuhr" $or$ author $soundex$ "smith" ) ]
Here, mandatory query conditions are combined with $\text{and}$ and optional query conditions are combined with $\text{or}$, and the mandatory and optional parts of the query are combined with $\text{and}$.

This method has the disadvantage that at least one of the optional query conditions must be fulfilled. In the extreme, if the user just types in $\text{+retrieval}$ and $\text{fuhr}$, the two query conditions will be connected with $\text{and}$ which is clearly the wrong thing to do. (However, connecting the mandatory part and the optional part with $\text{or}$ has its own problems!)

Thus, this method may return fewer documents than intended by the user.

The prefix is set in the HyGate config file, but the list of clauses and the path condition, search predicate, and comparison value, of each clause can be specified via the query form. The HyGate server administrator can assign arbitrary strings here, so care must be taken that the result is a valid XIRQL query.

{Todo: have HyGate do some sanity checks, especially on the comparison value.}  

It is suggested that a path condition be a list of element/attribute specifiers, separated by / or //. Meaningful path conditions include: \texttt{./*, foo/bar, ./section/@heading, author|editor, a/b/c//d/e/@f}.

Most administrators will probably hard-wire the path condition into the query form.

It is suggested that the search predicate be the name of a HyREX search predicate that’s useful for that document region. That is, if the DDL file for a specific document class specifies the data type $\text{HyREX::HyPath::Datatype::Name}$ for all $\text{author}$ elements, then $\text{soundex}$ and $\text{plainname}$ would be useful search predicates.

Most administrators will probably either hard-wire the search predicates into the query form, or provide a drop-down list of a few of them.

It is suggested that the comparison value be a string which is a single word. {Todo: have HyGate do something useful when the comparison value entered by the user looks like $\text{foo bar}$} Most administrators will probably provide a text entry field for the user to enter such values.

Now that we have talked about the content of a query form from a rather abstract point of view, we need to explain how the contents are encoded in HTML. In HyGate, each clause is identified by a name. In the HTML query form, each input field has a name. For the clause named $\text{foo}$, the HTML query form field which specifies the path condition should be named $a\_foo$, the field for the predicate should be named $p\_foo$, and the field for the comparison value should be named $v\_foo$.

That’s all there is to it, mostly. There are only two more parameters which need to be set in the HTML form, also through input fields: the $\text{hits}$ parameter specifies a ‘page size’ for the result list. That is, it is a number and HyGate will display portions of size this number, with ‘next’ and ‘previous’ buttons as appropriate.

The $\text{prefix}$ can be specified in the HTML form and overrides the value specified in the config file.  \{Is this really true? Please check. Do we want to remove the config file

\footnote{Of course, what is meaningful, also depends on the DTD and content of the documents. The list here is just to give you a feeling for it.}
Here is an example query form:

```html
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
    <title>Test form for querying</title>
</head>
<body>
<form action="/query" method="get">
    <table>
        <tbody>
        <tr>
            <td>Title:</td>
            <td><input type=hidden name="a_title" value="./projecttitle">
                <input type=text name="v_title" size="20">
                <select name="p_title">
                    <option value="$stemen$">stemming</option>
                    <option value="$plaintexten$">exact word</option>
                    <option value="$prefixen$">prefix</option>
                </select></td>
        </tr>
        <tr>
            <td>Description:</td>
            <td><input type=hidden name="a_descr" value="./(shortdesc | description)">
                <input type=text name="v_descr" size="20">
                <select name="p_descr">
                    <option value="$stemen$">stemming</option>
                    <option value="$plaintexten$">exact word</option>
                    <option value="$prefixen$">prefix</option>
                </select></td>
        </tr>
        <tr>
            <td>Contact person:</td>
            <td><input type=hidden name="a_contact" value="./contactpersons">
                <input type=text name="v_contact" size="20">
                <select name="p_contact">
                    <option value="$soundex$">phonetic similarity</option>
                    <option value="$plainname$">equality</option>
                </select></td>
        </tr>
        <tr>
            <td>Involved person:</td>
            <td><input type=hidden name="a_involv" value="./involvedpersons">
                <input type=text name="v_involv" size="20">
                <select name="p_involv">
                    <option value="$soundex$">phonetic similarity</option>
                    <option value="$plainname$">equality</option>
                </select></td>
        </tr>
        <tr>
            <td>Person:</td>
            <td><input type=hidden name="a_pers" value=""></td>
        </tr>
    </tbody>
</form>
</body>
</html>
```
2.6.3. The Summary Stylesheet

This stylesheet is invoked for displaying a result list. The structure of the stylesheet is pretty much determined by the contents of the XML document that represents the summary. Here is a description of that XML structure:

The top-level element is <summary>. It has a number of attributes and a number of child elements. The child elements are specified in the DDL, under the ‘summary’ section.

Attributes of the top-level <summary> element:

next_url If this attribute is present, the result list has several pages, and the value of this attribute is a link to the next result page. On the last result page, this attribute is not present.

prev_url Pointer to the previous result list page, if applicable. If the result only has one page, or HyGate is displaying the first page, this attribute is not present.

offset This number gives the offset from the start of the result list that’s displayed in this page. So the first page will always have offset 0, the second page will have an offset equal to the page size, the third page will have an offset equal to twice the page size, and so on.
pagesize This number gives the size of the currently displayed page. Note that on the last result page, this number can be smaller than on the other pages.

hits The total number of results returned by HyREX. Note that this can never be larger than the maxhits value specified in the HyGate config file.

The <summary> element has one child element for each result document. Each such child element also has attributes, in addition to whatever is specified in the DDL:

docurl This is a URL that will display the current document, when invoked by the user.

count This gives an index into the result list. The first result document will have a value of 1. Note that this number increases even across pages.

HyGate comes with a file common.xsl {Specify location!} which has named templates which can conveniently be used for the summary stylesheet. Currently, there are two named templates:

- The template page expects to be invoked from within the template for the top-level <summary> element and produces output for its attributes listed above, for example, ‘next’ and ‘previous’ pointers.

- The template number can be used for numbering items in the result list. The output includes the number, followed by a colon. The number can be clicked and will display the corresponding document.

2.6.4. The Document Stylesheet

Nothing much needs to be said about this. The input is a document, the output should be some HTML.

2.6.5. Invoking HyGate

It is very simple to invoke HyGate, as it only understands one option, -file. The option is followed by the name of the config file. Example invocation:

hygate -file /tmp/foo.cf

To terminate HyGate, just kill it.
3. Users
A. Paths and Path Expressions

Here we describe syntax and semantics of paths and path expressions. The natural result of XIRQL queries are paths which each identify a subtree of an XML document. Path expressions are used in HyREX within DDL documents (see Section 2.2). They are used to select one or more subtrees from an XML document for indexing purposes.

A.1. Paths

The context-free grammar for paths is depicted in Table A.1.

- {semantics of paths}
- {indexing of nodes: count children}

(1) \[\text{path} ::= / \text{element path} \]
\[\quad | / \text{element pathend} \]
(2) \[\text{pathend} ::= / \text{attribute} \]
\[\quad | / \text{pcdata} \]
(3) \[\text{element} ::= \text{elementname index} \]
(4) \[\text{attribute} ::= @ \text{attributename} \]
(5) \[\text{pcdata} ::= \#\text{PCDATA index} \]
(6) \[\text{index} ::= [ \text{integer} ] \]

Table A.1.: Context-free grammar in EBNF notation for paths

A.2. Path expressions

The context-free grammar for path expressions is depicted in Table A.2. Syntax and semantics of path expressions is borrowsed from the abbreviated syntax from XPath [Clark & DeRose 99]. From XQL [Robie et al. 98] we borrowed the alternative construct (rule 4 in the grammar). Note that only a subset of the XPath / XQL languages can be used for path expressions.
A.2. PATH EXPRESSIONS

APPENDIX A. PATHS AND PATH EXPRESSIONS

Table A.2.: Context-free grammar in EBNF notation for path expressions

<table>
<thead>
<tr>
<th>Rule</th>
<th>Grammar</th>
</tr>
</thead>
<tbody>
<tr>
<td>pathexp</td>
<td>::= separator step pathexp</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>separator</td>
<td>::= /</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>step</td>
<td>::= elementname</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>alternatives</td>
<td>::= / ( elementname</td>
</tr>
<tr>
<td>alterelements</td>
<td>::= elementname</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>pathexpend</td>
<td>::= step</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(1) pathexp ::= separator step pathexp
              |             separator finalstep
(2) separator ::= /
              | //
(3) step ::= elementname
              | alternatives
              | *
(4) alternatives ::= / ( elementname | alterelements )
(5) alterelements ::= elementname | alterelements
              | elementname
(6) pathexpend ::= step
              | @ attributename
              | @ *
              | #PCDATA
B. DTD for HyREX DDL

Syntax of the Document Definition Language (DDL) for HyREX is defined by the following Document Type Definition. For validating your own DDL files you can access the this DTD form file doc/hyrex.dtd within the HyREX distribution.

<?xml version="1.0" encoding="ISO-8859-1" ?>

<!-- ************************************************************* -->
<!-- *** hyrex.dtd - A DTD to describe HyREX document classes **** -->
<!-- *************** and their schemas. ************************** -->
<!-- ************************************************************* -->

<!-- $RCSfile: hyrex.dtd,v $ -->
<!-- $Id: hyrex.dtd,v 1.16 2002/01/02 17:28:17 goevert Exp $ -->
<!-- $Name: $ -->

<!-- Root element 'hyrex' introduces a schema for a HyREX document class. Its attributes describe into which document base (attribute 'base') in which directory (attribute 'directory') to the document base lives, how to name the class (attribute 'class') and the file which contains the dtd for the documents to be filled into the class. -->

<!ELEMENT hyrex ( access, convert?, summary, datatype+, inodes?, structure, transfer? )>
<!ATTLIST hyrex
  base CDATA #REQUIRED
  class CDATA #REQUIRED
  directory CDATA #REQUIRED
  dtd CDATA #REQUIRED>

<!-- ************************************************************* -->
<!-- *** access to documents ************************************* -->
<!-- ************************************************************* -->

<!-- Element 'access' describes the method how to access documents for
the given document class. Attribute 'classname' gives the access
class name, while the content of element 'access' might give
arguments for the class constructor. See
HyREX::HyPath::Document::Access(3) for details. Parameters are
given as (name, value) pairs (attributes of element 'paramater').
-->

<!ELEMENT access (parameter*)>
<!ATTLIST access
  classname CDATA #REQUIRED>

<!ELEMENT parameter EMPTY>
<!ATTLIST parameter
  name CDATA #IMPLIED
  value CDATA #IMPLIED>

<!-- ************************************************************* -->
<!-- *** document conversion ************************************* -->
<!-- ************************************************************* -->

<!-- Element 'convert' describes on-the-fly document converter. Just
after reading a document from the access method specified above
the result of reading is taken is input to the converter
specified here. The conversion result is handed out to HyREX for
indexing. The converter is applied to the document at retrieval

time, too.

<!ELEMENT convert (parameter*)>
<!ATTLIST convert
classname CDATA #REQUIRED>

<!-- ************************************************************* -->
<!-- *** document summaries ************************************** -->
<!-- ************************************************************* -->

<!-- Element ‘summary’ describes how to construct a document summary. The summary will be generated in XML, the structure of the XML is described here. -->

<!ELEMENT summary (element | xsl | xslfile)>

<!-- A summary description exists of nesting elements ‘element’ and ‘query’. Elements ‘element’ only have a ‘name’ attribute (name of element generated in headline) and the facility to nest further headline queries and elements. From the ‘query’ elements real document content for the summary is derived; attribute ‘query’ gives a query which is processed against the document in order to determine the element content. If attribute ‘structure’ is given and set to ‘yes’, xml tags in summaries are retained. -->

<!ELEMENT element (element | query)*>  
<!ATTLIST element
name CDATA #REQUIRED>

<!ELEMENT query EMPTY>
<!ATTLIST query
query CDATA #REQUIRED
weight CDATA #IMPLIED
structure (no | yes) "no">

<!-- Alternative to specify the summaries structure this way an XSL
can be specified which then is processed against the document under consideration. The result of this process is taken then as the summary. XSL stylesheets can be specified either by directly including the stylesheet into the ddl (element ‘xsl’) or by referencing a file containing the stylesheet (attribute ‘name’ of element ‘xslfile’).

<!ELEMENT xsl (#PCDATA)>
<!ELEMENT xslfile EMPTY>
<!ATTLIST xslfile
    name CDATA ""
>
</!--
<!--  *******************************************************
<!--  *** datatype specifications ********************
<!--  *******************************************************

<!-- Element ‘datatype’ specifies a data type and which content from the documents is to be represented/indexed under that datatype. Attribute ‘classname’ gives the HyREX class to use for the attribute. The content of element ‘datatype’ specifies the arguments needed for the constructor of the data type class used, the name of an accesspath for the documents’ structure, and queries specifying the content to be indexed within the named data type.

<!ELEMENT datatype (parameter*, query*)>
<!ATTLIST datatype
classname CDATA #REQUIRED
>
</!--
<!--  *******************************************************
<!--  *** inode specifications ***********************
<!--  *******************************************************

<!-- Element ‘inodes’ describes the borders of index nodes in the documents, i. e. the parts in the documents’ structures which should be treated as candidates for a retrieval result. Each index node is described by a query. Documents by themselves are
treated as index nodes by default.

<!ELEMENT inodes (query*)>

<!-- ************************************************************* -->
<!-- *** structural accesspath specifications ********************* -->
<!-- ************************************************************* -->

<!-- Element ‘structure’ describes properties of the structure access path to be used to store the structural data. The content of element ‘structure’ might give arguments for the class constructor.

<!ELEMENT structure (parameter*)>
<!ATTLIST structure
    classname CDATA #REQUIRED
>
<!-- ************************************************************* -->
<!-- *** transfer specifications ********************************* -->
<!-- ************************************************************* -->

<!-- Element ‘transfer’ describes the query transfer. The attribute ‘server_url’ specifies the URL where the transfer server is running. Subelements describe how to transfer each operator.

<!ELEMENT transfer (transfer_op*)>
<!ATTLIST transfer
    server_url CDATA #REQUIRED
>
<!-- A subelement ‘transfer_op’ of ‘transfer’ specifies how to transfer a specific operator. The content is empty, but it has the following attributes:

    source_op The operator in the original query, eg $te$. 
    target_op The operator in the transferred query, eg $stemen$. 
    source_element target_element source_docset

38
source_doclang
target_docset

All these attributes are required.

source_docset and source_doclang are integers. Here is a list of possible values for *_docset, together with their meaning:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MathNet</td>
</tr>
<tr>
<td>2</td>
<td>PhysNet</td>
</tr>
<tr>
<td>3</td>
<td>SOLIS</td>
</tr>
<tr>
<td>4</td>
<td>FORIS</td>
</tr>
<tr>
<td>6</td>
<td>Die Deutsche Bibliothek</td>
</tr>
<tr>
<td>7</td>
<td>SoWi Internetquellen</td>
</tr>
<tr>
<td>8</td>
<td>Elib</td>
</tr>
<tr>
<td>9</td>
<td>GIRT</td>
</tr>
</tbody>
</table>

source_doclang is also an integer. Here is a list of possible values, together with their meaning:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Free Terms</td>
</tr>
<tr>
<td>1</td>
<td>MSC</td>
</tr>
<tr>
<td>2</td>
<td>PACS</td>
</tr>
<tr>
<td>3</td>
<td>IZ classification</td>
</tr>
<tr>
<td>10</td>
<td>IZ thesaurus</td>
</tr>
<tr>
<td>11</td>
<td>SWD</td>
</tr>
</tbody>
</table>

<!ELEMENT transfer_op (target_spec+)>
<!ATTLIST transfer_op
    statistical_threshold CDATA #IMPLIED
    min_intellectual_relevance CDATA #IMPLIED
    max_indirect_transfers CDATA #IMPLIED
    skip_terms CDATA #IMPLIED
    max_new_terms CDATA #IMPLIED
    preferred_target_doc_lang CDATA #IMPLIED
    source_doc_set CDATA #REQUIRED
    source_doc_lang CDATA #REQUIRED
    target_doc_set CDATA #REQUIRED
    source_op CDATA #REQUIRED
    source_elem CDATA #REQUIRED
    replace (true|false) "true"
<!-- Transfer Target Specification -->
<!ELEMENT target_spec EMPTY >
<!ATTLIST target_spec
target_doc_lang CDATA #REQUIRED
target_elem CDATA #REQUIRED
target_op CDATA #REQUIRED
max_new_terms CDATA #IMPLIED
>
List of Tables

A.1. Context-free grammar in EBNF notation for paths .......... 32
A.2. Context-free grammar in EBNF notation for path expressions .......... 33
List of Figures

1.1. The XML Document Model ........................................ 5
1.2. Mapping of data types and documents .......................... 5
1.3. Data types available in HyREX .................................. 6
1.4. HyREX Architecture .................................................. 7
Bibliography


