TRIPxml

User's Guide

Version 3.1

End User License Agreement

All rights to this software, its documentation and logotypes of the TRIP product family and software (altogether "Software") supplied by infinIT Services GmbH (infinIT) are exclusively owned by infinIT.

The transfer of this Software, solutions or parts thereof requires the prior written agreement of infinIT. Furthermore, the customer has the right to use licensed Software and / or process solutions supplied by infinIT to the extent specified in his contract with infinIT.

The free-to-use non-commercial version doesn't require a prior written agreement with infinIT but such customers, organizations and/or third parties agree by using the software and / or solution of infinIT to be strongly obliged to keep all rights to this software, documentation and logotypes of the TRIP product family absolutely uninfringed and protected.

Table of Contents

Introduction

About this document

This manual describes the version 3.1 of TRIPxml. Descriptions of APIs and examples for their use is found with respective SDK product (TRIPjxp, TRIPnxp, etc).

Defining XML

XML, or Extensible Markup Language, is not a fixed format like HTML. While HTML is limited to a fixed set of tags that the author can use, XML users can create their own markup (or use markup created by others) that actually relate to their content. As such it is a meta language – a language that describes languages.

XML is a subset of a language called SGML – Structured Generalized Markup Language – that has been modified for use on the internet. All the difficult and complex parts of SGML has been excluded while the flexibility remains.

Instead of going further into technicalities about XML, let's regard a few areas where introduction of XML will be of help.

Why Use XML?

Usage: Information Exchange

Exchange of information between different computers and systems is not always a simple problem. In some cases, this is near impossible, and manual recreation of the information is required. In other cases, conversion programs can do the job. XML helps us define interchange formats between different systems and applications.

Usage: Long Time Storage

Interchange of information between systems and applications are difficult without use of standards like XML, but even to move data between different versions of the same application can sometimes be difficult. Many companies therefore try to find solutions in which they can store information and guarantee the readability and usage for long time, say thirty years. They need a highly flexible standard to represent information that will not be obsolete in a few years' time. The standard they have been looking for is XML.

Usage: Reuse

Technical development is moving along at a high speed in all areas - not just in information technology. An ever-greater complexity put high demands on the technical documentation, which is critical for product understanding and usage.

SGML, the predecessor of XML, has primarily been used in areas where it is very costly to produce advanced technical documentation.

Reuse does not just mean reuse of existing text. It also means that we should be able to publish information in different media with a minimum of work. A good example here is an organization who wants to distribute information on paper, optical media, intranet and internet simultaneously. In order to do this, separation of fact from presentation and layout is required.

Reuse is also strongly associated with information retrieval, because you have to find the information in order to reuse it. In order for information to be reused, it has to be marked up so we know what it informs of. Decomposition into smaller components so that parts can be used in different context is also a great idea.

Usage: Structured Methods
Having the authors of expensive information concentrate on the fact and not the layout is more cost effective.

XML provides templates for structure and layout. In order to validate documents, i.e. certain parts must exist and be structured in a certain fashion, rules for structure can be built into the templates.

Usage: Smart Information
Information can be presented in a more or less intelligent manner. If we choose to label a paper document as a ”stupid” information carrier, we can label an advanced multimedia presentation as an ”intelligent” information carrier.

A requirement for more advanced applications is that the information to be processed is described in a way that a computer program can use. XML coded information is highly suitable for use in multimedia presentations or electronic manuals.

Conclusion
It appears that XML is a standard for describing and representing information that promises not to go obsolete before we know it. XML allows us to write concentrating on the fact, totally separated from the layout, in such a way that a computer program can use it. That, in turn, requires efficient storage of the information. Efficient and easy-to-use search and retrieval routines are also a necessity.

Installation

Requirements

In order to install this product, you need an existing installation of TRIPsystem version 7.1 or later. This release supports the following platforms:

- 32-bit versions of Windows 7, Server 2008
- 64-bit versions of Windows 7, 10, Server 2012 and Server 2016
- 32 and 64-bit versions of Linux for Intel x86 with minimum GLIBC 2.12 and GLIBCXX 3.4.13.
- Solaris SPARC

You also need something with which to create applications. The most complete API for TRIPxml is found with TRIPjxp and TRIPnxp version 2.1 and later, but you can also use the server-side TRIP toolkit, TRIPjxp and TRIPnxp. The older SDK products TRIPclient and TRIPjtk can also be used, although they are not recommended for new development projects.

Content of the distribution package

There is one distribution kit per supported platform. The following directory structure is created for Linux, Solaris, AIX and HP-UX.

v310	Top directory for this installation.
v310/bin	Server-side tools and upgrade scripts.
v310/lib	Libraries for TRIPxml core functionality
v310/include	Headers for the TRIPxml C API (to be used with the TRIPsystem toolkit).
v310/samples/xmlget	Sample TRIP toolkit source for a CGI based XML extraction utility.
v310/samples/xmlput	Sample TRIP toolkit source for an XML storage utility
v310/samples/xalice	A simple TRIPxml-version of the book "Alice in Wonderland".

The following directory structure applies to Windows installations:

v310	Top directory for this installation.
v310/bin	Libraries and tools for TRIPxml functionality.
v310/lib	Lib-files for the client-side TRIPxml C API
v310/include	Headers for the client-side TRIPxml C API
v310/samples/xmlget	Sample TRIP toolkit source for a XML extraction utility
v310/samples/xmlput	Sample TRIP toolkit source for an XML storage utility
v310/samples/xalice	A simple TRIPxml-version of the book "Alice in Wonderland".

Note that the TRIPxml documentation can be found as a separate download from infinIT Services GmbH. Also note that components of TRIPxml that are specific to the Windows platform are only available through the Windows installer.

TRIPxml no longer ships components dependent on TRIPclient. If you have been using the TCXML component, you will find its classes integrated into TRIPcom in TRIPclient from version 2.5-0.

Windows Installation Procedure

You are strongly recommended to uninstall previous versions of TRIPxml and TRIPxml Client Tools before installing the new version of TRIPxml. Not doing this may cause conflicts later if TRIPclient is installed or upgraded on the same machine.

The following steps are performed by the installer:

1. Copies the files to a location you specify.
2. Adds the bin directory to the system PATH.
3. Modifies the TRIPsystem configuration file (see steps 1, 2 and 3 in the Unix installation procedure script description for details).

Users of the TCXML component will have to upgrade TRIPclient to at least version 2.5-0, since this component has been integrated into TRIPcom and is no longer delivered as part of TRIPxml.

IMPORTANT: When upgrading to TRIPclient 2.5-0 or later on a machine that also has TRIPxml installed, make sure that the TRIPxml version is 3.0 or later before upgrading TRIPclient.

Unix Installation Procedure

Unpack

Unpack the archive containing the TRIPxml distribution for your platform to the location where you want the files to be located.

Run Installation Script

Go to the v310/bin directory and execute the script install.sh. The script does the following:

1. Modifies your tdbs.conf file to have the TDBS_XMLSHR setting under the NonPrivileged section point to the fully qualified path of the xmlfilters shared library provided in the lib directory. It will look something like this: TDBS_XMLSHR=/opt/trip/xml/v310/lib/xmlfilters.so.
2. Adds TDBS_XMLSHR to the TDBS_ASELIBS setting under the the NonPrivileged section in the tdbs.conf file. The TDBS_ASELIBS setting specifies all logical symbols that point to libraries containing ASE routines and filters. If the TDBS_USRSHR setting (the standard ASE logical symbol) is used, the value may after editing look like this: TDBS_ASELIBS=TDBS_XMLSHR, TDBS_USRSHR
3. Adds the following new settings to the tdbs.conf file: XML_LOG, XML_LINK_DB, XML_LINKS, XML_STRICT_VALIDATION, XML_SCHEMA and XML_STRICT_SCHEMA.
4. Changes the user and group ownership of the installed files according to your input to the promptings of the script.
5. Creates links to ICU and Xerces support libraries under /usr/local/trip/xml/lib

Library Path
The TRIPxml shared library depends on quite a few other libraries. If one of them is missing or otherwise cannot be found or loaded, TRIPxml will not work. If your TRIPxml installation does not work, you may want to run the 'ldd' command on the xmlfilters shared library file (named xmlfilters.so) and see if there are any broken library dependencies. If there are, you may have to adjust the system shared library path to include the appropriate directories.

Directories that may cause trouble are /usr/local/lib in which the Standard C++ library usually is located on Solaris and Linux and the support library directory for ICU and Xerces (/usr/local/trip/xml/lib).

SDK Usage
If you want to use TRIPxml from a client application, you will need to make sure that all libraries that the xmlfilters shared library depend upon (as discussed in the Library Path section above) are available to tripnetd (alternatively xinetd or inetd). Although the TRIPxml binaries are linked with an RPATH setting that should make sure that all required libraries can be found, security considerations on operating system level may prohibit the use of such settings.

TRIPxml Features

Requirements

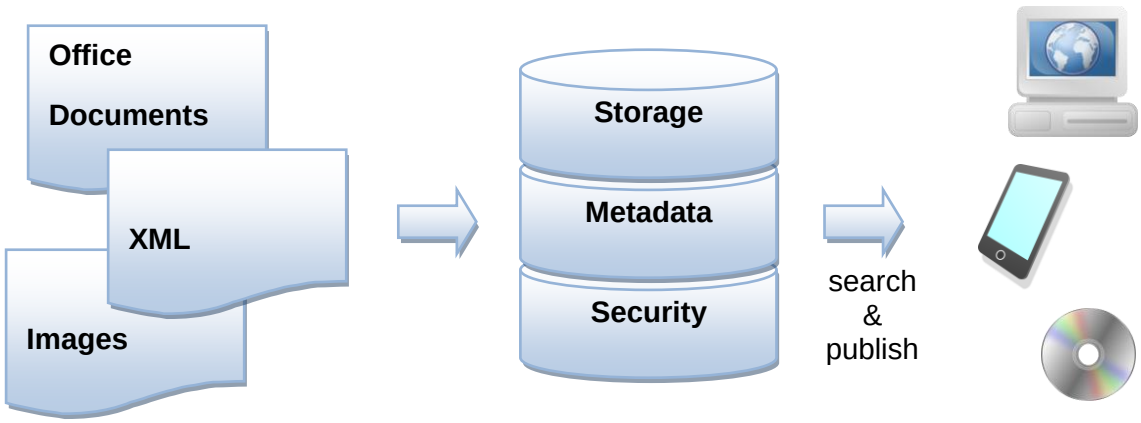
What kind of features do we want from a TRIP-based system that can store XML data as well as offer an easy-to-use interface for search and retrieval? Assuming that we deal with XML documents and not relational or object-oriented data in XML-format, the following points comes to mind:

- Ability to represent an XML-structure, complete with elements and attributes.
- It should be easy to create a TRIP query to find XML documents and sections within the documents.
- The TRIP query should be easy to compose, using only structure of the stored XML data.
- Ability to present the information in different ways depending on the user's interest.
- Optional storage of related DTD and stylesheet files.

Design

Overview

Although TRIPxml have been through some fairly major internal changes since its first version, the basic design remains. Built around the Xerces XML parser from Apache Software Foundation, it allows using TRIP as an XML database with XML-based storage, search and retrieval.



Among the possibilities offered by TRIPxml are:

- Programmatic access from both server (TRIP toolkit) and client (TRIPjxp, TRIPnxp, TRIPclient and TRIPjtk).
- Support for the XLink and XInclude protocols.
- Storage of any file, in XML format or otherwise.
- Seamless integration with TRIPview freetext-search enables non-XML documents.
- Use XPath as query language across the entire document set

Storage Structure
An XML document is basically structured like a tree, which is why the XML data will be represented as a tree. In order to make retrieval of documents efficient, the document will be stored as it is as well as in parts. Storing the parts enables more intelligent searching than otherwise would be possible.

The table below shows the minimal TRIP database design for any database that will store XML data. Each record in the database stores one XML document, both as a "blob" and in a tree-structure.

Field Name	Field Number	Part Field	Field Type	Description
D_XMLDOC	1	No	STRING	The entire XML document as a BLOB
D_META	2	No	TEXT	Document metatags
D_DOCTYPE	3	No	TEXT	Document doctype tag
D_DOCSIZE	4	No	INTEGER	Size (in bytes) of XML document stored in D_XMLDOC
D_DOCNAME	5	No	PHRASE	Filename of XML document
D_URLBASE	6	No	PHRASE	URL to document on the web, excluding the document name itself.
D_URLALIAS	7	No	PHRASE	Alias for the URL
N_ID	8	No	INTEGER	Tree node identity
N_PARENT	9	No	INTEGER	Parent node identity
N_SEQPATH	10	No	PHRASE	Path to node using node sequence numbers.
N_PATH	11	No	PHRASE	Complete path to node
N_SEQNO	12	No	INTEGER	Sequence number of the node
N_NAME	13	No	PHRASE	Node name
N_TYPE	14	No	PHRASE	Node type (element, attribute, text, ...)
N_NSPACE	15	No	PHRASE	Node namespace
N_MIME	16	Yes	PHRASE	What data is stored, e.g. image/jpeg
N_ENCODING	17	Yes	PHRASE	Type or representation of the data, e.g. base64
N_CVALUE	18	Yes	STRING	Node value (non-text)
N_TVALUE	19	Yes	1*TEXT	Node value (text only)
N_NVALUE	20	Yes	NUMBER	Node value (numeric)
N_DVALUE	21	Yes	DATE	Node value (date)
D_PROPNAME	22	No	PHRASE	Property name field
D_PROPVAL	23	No	PHRASE	Property value field
DAV_NONXMLBLOB	24	No	STRING	Field for storage of non-XML files.
D_DOCTEXT	25	No	TEXT	Field for storage of text extract of non-XML files.
D_ID	26	No	PHRASE	Document/record identity field
N_RESERVED	50	No	PHRASE	Max field number reserved for XML

XLink and XInclude support

The XLink protocol is an XML-based technique to create links between information on the web. XInclude is a small XML protocol that allows one XML file to include another. For more information on these protocols, see the World Wide Web Consortium's web site www.w3.org.

TRIPxml supports these two protocols by maintaining a link database. Information about outgoing links for each XML document is kept. For more info, see the section "Create a Link Database" below.

Querying

Querying an XML document is like querying a tree. The following points describe the possibilities of queries against a TRIPxml database.

- Search for specific XML elements
- Search for information in specific XML elements
- Perform freetext search on entire document content
- Boolean logic in search conditions (and, or, not)
- Searching for words NEAR each other, ie the search words should be in the same sentence or paragraph.
- Use CCL or XPath

TRIPxml always returns the entire XML document as an answer, and not just the element sets that matched the query. There will be ways to search for and retrieve parts of XML documents. An XPath-oriented querying style will be introduced in a future release of TRIPxml.

The following points summarise retrieval:

- Retrieval of entire documents
- Retrieval of fragments of documents found by a search. These fragments do not necessarily have to be the same parts that matched the query.
- Possibility to, no matter how the result was presented, retrieve the entire documents or specified parts thereof.
- Possibility to get the words in the text that matched the query condition marked for highlighting.

For more information about search and retrieval of XML documents, please refer to the section ”Query the database” in the ”Howto” chapter.

Application Programming

TRIPxml comes with API routines for programmatic access to the XML functionality in TRIP. Features:

- Search using XPath expressions or a simplified, abbreviated form of CCL that enables the use of element names as field names.
- Import routines to store an XML document into TRIP, with optional validation according to DTD or XML schema.

- Retrieval routines for both entire XML documents as well as parts of the documents and related DTDs and stylesheets stored in the database.
- Access from TRIPsystem, TRIPnxp, TRIPjxp, TRIPclient, TRIPjtk and TRIPhighway.

How-To

Install the XALICE sample

The xalice sample database is a simple XML version of the book "Alice In Wonderland" by Lewis Carroll. The data is divided into several xml files, with 1-2 chapters each. The purpose of the sample is to provide a data set so that you can try out the features of TRIPxml; importing, exporting, and searching. Samples for access from TRIPclient are provided with the distribution of TRIPclient version 2.0-1 or later.

The samples/xalice directory contains the xalice sample database for TRIPxml. Make that directory current and run the TRIP classic program. Issue the following call:

```
IMPO BAS=XALICE.* FILE=XALICE.DEF
```

Then exit TRIP classic. You have now created your first TRIPxml database! Note that this is not the standard way to create an XML database (see section "Create an XML database" below for details). To fill it with the provided sample data, a few extracts from the book Alice in Wonderland by Lewis Carroll. You can use the TRIPxml tool txput (see below) from the command line:

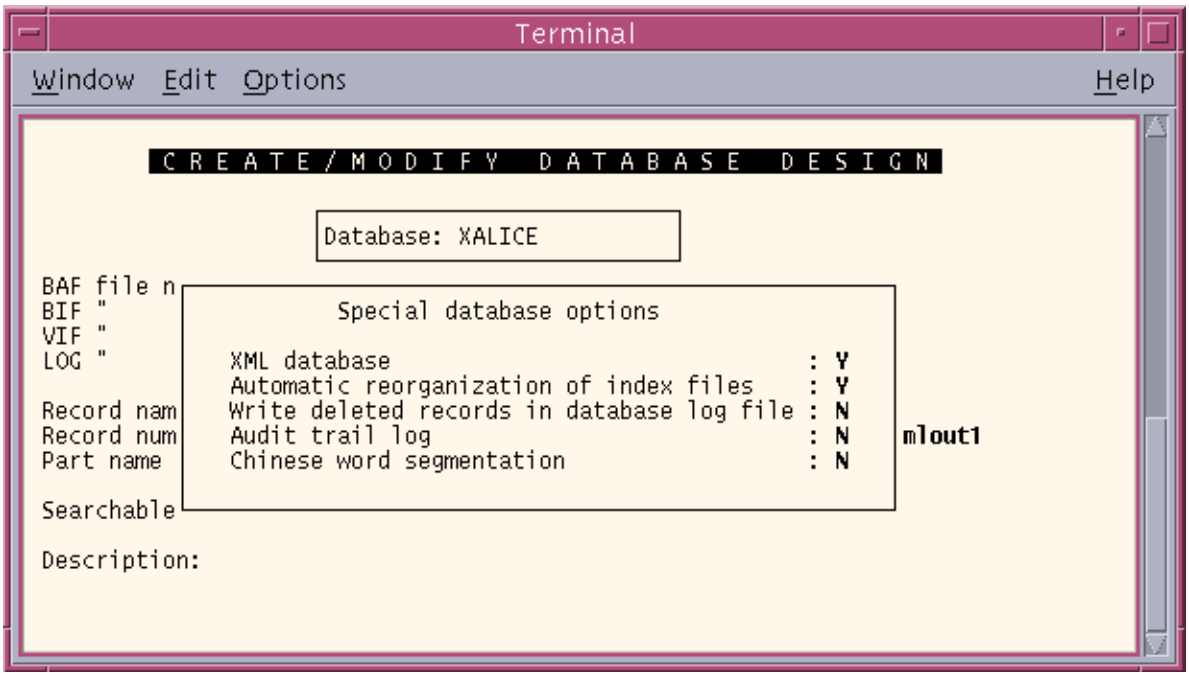
```
txput –i –d xalice –u myuser –p mypassword  wonderland*.xml
```

Note the wildcard! This will import the six XML documents that together make up the book. The "-i" option tells the program to index the database as well. If you do not specify the "-i" option, you will have to index it manually (by issuing the command "index xalice" from the command prompt). You have now imported your first XML documents into trip!

Create an XML database

Via TRIPclassic

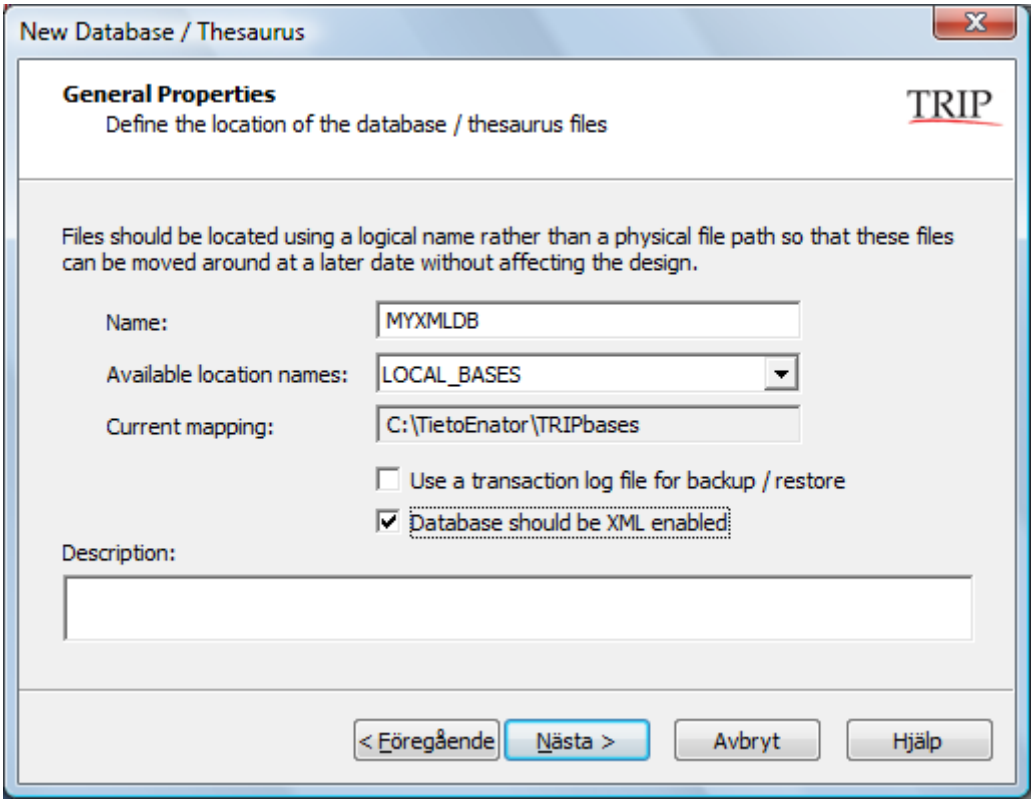
XML databases can be created using TRIPclassic.



Select Databases from the Administration menu. In the database menu, select create/modify database from the DB Design menu. Enter the name of the database to be created. Press '6' on the numeric keypad (or 'Ctrl-K' followed by '6') to bring up the special database options dialog. Set the XML database to 'Y'. Press ENTER (or 'Ctrl-E') to confirm, then finally press ENTER (or 'Ctrl-E') again to save the new database.

Via TRIPmanager
XML databases can also be created using TRIPmanager. Right-click the "Databases" node to bring up the context menu. Select the "New Database…" option and click next. On the "general properties" page of the wizard, check the option "Database should be XML enabled".

When you get the question "Do you want to specify the field collection for this new database / thesaurus?" you should only answer "yes" if you wish to customize the default XML database design. If you do customize it, please do not change the definition for any of the pre-defined fields since this may render the database unusable by TRIPxml.



Creating a Unicode XML database

TRIPxml databases created with TRIPsystem 6.2-5 or later are Unicode-enabled by default. It is not recommended to change this setting.

Migrating an XML database to Unicode

Older versions of TRIPxml used whatever character set TRIPsystem was set up to use, even if the imported XML data happended to be in Latin-1 or gb2312. Unfortunately, this means that the only reliable way to get old XML data converted to a Unicode database without risk for corruption, is to re-import the XML documents into a new Unicode TRIPxml database.

Create a Link Database

TRIPxml comes with link support disabled by default. To enable link support, add "XML_LINKS=1" to your TRIPrcs file under the nonprivileged section. A default link database XML_LINK_DB is provided with TRIPsystem. If you wish use another link database, you will have to create it. Specify the name of the link database to use in the TRIPrcs file; add "XML_LINK_DB=*mylinkdb*" under the nonprivileged section. To create the link database itself, use the *mklinkdb* utility.

Remember to give all TRIPxml users proper access rights to the link database. What is proper for a particular user depends on what he/she is supposed to be doing. For import, the link database must be readavble and writable. Otherwise it is sufficient to only have it readable. Remember that you must also set the access rights for the default link database.

If you have enabled link support, all XML documents that are imported gets an associated link information record. This record contains information about all links that emanate from the document, even if the actual link specification is in another XML document. The actual data for the link information record is gathered by a separate utility called *txlinker*, which preferably is set to be run by a timed job like *cron* on Unix platforms.

If you have an XML database in which there are documents that does not have link information records, you can use the *updlinks* utility to add link information to the database. Also, if there are records that have been removed, this utility can remove the orphaned link information records from the link database.

See the sections on *txlinker*, *updlinks*, and *mklinkdb* for more information about how to maintain a link database.

Query the Database

Using CCL

A TRIPxml database can be queried using CCL like any other TRIP database. There is one big difference, however. The difference is that with a TRIPxml database, you can pose a query using the element names of the stored XML documents! Note that if there is an element and a field with the same name in a TRIPxml database, the element will always be used.

When querying XALICE, you can use the following from TRIPclassic:

```
FIND CHAPTER=height
```

which will search for XML documents that have an element "CHAPTER", that contains – directly or in a sub element – the text "height".

The XALICE sample database has a special default output format named "XMLOUT1". This output format uses a special ASE named "xmldata" that comes with TRIPxml. This ASE is used to extract specific values from an XML document in a TRIPxml database, and can also be used from TRIPhighway. No corresponding API call exist at the moment, however.

In addition to using tag names in search conditions, as described above, you can use part of the path, like this:

```
FIND BOOK/CHAPTER=height
```

When you pose queries like this using TRIP classic, you'll notice that TRIP expands the XML-oriented query to an ordinary CCL query.

Using XPath
From TRIPxml version 3.0, it is possible to use XPath to query a TRIPxml database. This is possible via the classes TdbSearch, TdbCclCommand and TdbRecordSet in TRIPjxp and TRIPnxp version 2.1 and later. Please refer to the "TRIPjxp & TRIPnxp Programmer's Guide" document for detailed information.

The XPath syntax supported is a subset of XPath 1.0 as specified by the W3C. See the appendix for a description of supported XPath syntax.

Use the TRIPxml C API Calls

There are eight special APIs for import into and export from TRIPxml databases, and two for XPath-based functionality. Please refer to the HTML help file TXCAPI.CHM for API reference and usage examples.

Use the Toolkit API Calls

The two API calls in the TRIP toolkit that pertain to TRIPxml are TdbImport and TdbExport. These uses so-called filters. A filter is - for TRIPxml – a routine that does the actual job of exporting or importing data. Filters are functions compiled into shared libraries (extension .dll on Windows and .so, .o, or .sl on Unix) that are callable from inside TRIP.

Both TdbImport and TdbExport use a structure called filter_data. This structure is used to pass information to and from the filter routine, including options and return data. For a detailed description of this structure, see "Appendix A - API Reference".

Although there is nothing wrong with using TdbImport and TdbExport, the regular TRIPxml C API calls (see above) should be used whenever possible when creating server-side TRIP Toolkit applications. The reason for this is that TdbImport and TdbExport are low-level functions and require a lot more care in usage than their ordinary counterparts.

TdbImport
The TdbImport function is used to call TRIPxml to import XML documents.

Prototype

```
int TdbImport ( filter_data* data );
```

Applicable values for the tdb_options member:

NAME	VALUE	DESCRIPTION
IEOPT_FILENAME	1	The buffer member contains file name
IEOPT_FILEPTR	2	The buffer member contains file pointer (FILE*)
IEOPT_MEMORY	4	The buffer member points to memory area

Applicable mask values for the filter_options member:

NAME	VALUE	DESCRIPTION
FOXML_NEWREC	1	Create a new record.
FOXML_REPLACE	2	The opposite of FOXML_NEWREC. Is implicitly set if FOXML_NEWREC is not set. The record_control and cursor fields are mandatory

NAME	VALUE	DESCRIPTION
		in combination with this option.
FOXML_VALIDATE	4	Specifies that the caller wishes to validate the XML document before it is imported.
FOXML_NOBLOB	64	Specifies that the caller does not wish to store a copy of the document in the D_XMLDOC field.
FOXML_STREAM	256	Specifies that the caller wishes to perform stream-oriented I/O.

For TRIPxml, URL information may be stored by using the filter_arguments parameter in the filter_data structure. Since this structure is generic, i.e. not specific to TRIPxml, each particular TRIP module using these functions may define different syntactical and semantical rules for the filter_arguments parameter.

Anyhow, for TRIPxml, several filter arguments may be specified. The terminating character for an argument line is a newline character. The line thus follows this syntax:

```
name = value '\n'
```

So, defining the URL http://www.myweb.com/mysite/index.html, the line will look like this:

```
URL=http://www.myweb.com/mysite/index.html
```

Please note that in the database, the URL is actually stored in two parts. The filename itself (e.g. index.html) is stored in the field D_DOCNAME, and the other part of the URL (e.g. http://www.myweb.com/mysite) is stored in the field D_URLBASE.

Blob-oriented import
Blob-oriented import is actually a special variant of block-oriented import. The caller provides all the data required in memory pointed to by buffer in one go. The buffer_length specify the size of the entire blob (XML document), and blockno must be set to 0. The option for this is IEOPT_MEMORY.

File-oriented import
File-oriented import comes in two flavors. One in which a file name is specified in a character string pointed to by buffer, and one in which a file pointer is provided in buffer. In case a file pointer is provided, neither the TdbImport function, nor the filter function must close the file. The options are IEOPT_FILENAME and IEOPT_FILEPTR, respectively.

Stream-oriented import
Stream-oriented import is preferrably used when importing large files into a TRIPxml database from a web site or a client application (based on TRIPjtk or TRIPclient). This is also the only way validation of system-local DTDs or schemas can take place when the DTD is not available on the server machine.

To use streamed import, add FOXML_STREAM to the filter_options member of filter_data. It is important also to assign either IEOPT_FILENAME or IEOPT_MEMORY to the tdb_options member of filter_data. You must also add a FILEURL parameter to the filter_arguments string, specifying the URL from which you want TRIPxml to load the XML file, e.g. "FILEURL=http://mybox:1234/data.xml".

Remember that the best way to program stream oriented import is to use the functions in the regular TRIPxml C API for a much easier programming interface. Please refer to the compressed HTML online help file TXCAPI.CHM for more details.

Validation
If you want to validate an XML document when you are importing it, add FOXML_VALIDATE to the filter_options member of filter_data. If XML_SCHEMA is present and set to 1 in the server-side TRIPrcs file, then XML schemas can be validated. Otherwise, only DTDs are validated.

Note that if you are performing a blob-oriented import of an XML document referrring to a DTD using a relative path and the DTD file has not been imported into the TRIPxml database, you will get a validation error.

TdbExport
The TdbExport function is used to call TRIPxml to export XML documents, with or without hit markup.

```
Prototype

int TdbExport ( filter_data* data );
```

Applicable values for the tdb_options member:

NAME	VALUE	DESCRIPTION
IEOPT_FILENAME	1	The buffer member contains file name
IEOPT_FILEPTR	2	The buffer member contains file pointer (FILE*)
IEOPT_MEMORY	4	The buffer member points to memory area
EXPORT_ALLOC	32	Declares that export filter should allocate memory for data (the buffer member is changed to point to newly allocated memory). Used in combination with IEOPT_MEMORY.
EXPORT_FILEAPP	256	Declares that export filter should append XML document to a file. Used in combination with either IEOPT_FILENAME or IEOPT_FILEPTR.

Applicable mask values for the filter_options member:

NAME	VALUE	DESCRIPTION
FOXML_GETBYID	8	Retrieve document by URL or record name (ID). Specify URL by adding the URL parameter to the filter_arguments member. Specify ID by adding the ID to the filter_arguments member.
FOXML_REMAKE	16	Recreate the xml document from its parts. Do not use data stored in the D_XMLDOC field.
FOXML_HILIGHT	32	Insert hit-markup in the extracted xml document, so that the displaying application can highlight the hits. The option FOXML_REMAKE must also be set.
FOXML_STREAM	256	Specifies that the caller wishes to perform stream-oriented I/O.

Blob-oriented export
Blob-oriented export means that the caller wants all the required data returned in buffer in one go. The buffer_length specifies the size of the entire blob, and blockno must be set to 0. The option for this is IEOPT_MEMORY.

Blob-oriented export also comes in another flavor, and that is that the filter routine allocates a large-enough buffer to contain the requested data. If this is the case, the filter routine will set buffer to allocated memory containing the data and buffer_length to the size of the buffer. The option for this is IEOPT_MEMORY | EXPORT_ALLOC.

File-oriented export
File-oriented export comes in two basic flavours. One in which a file name is specified in a character string pointed to by buffer, and one in which a file pointer is provided in buffer. The options are IEOPT_FILENAME and IEOPT_FILEPTR, respectively. If combined with the EXPORT_MKFILE option, the file name (or pointer) is created by the filter routine and returned in buffer. It will in this case not be neccessary (or required) to provide anything in buffer by the caller.

For both types of file-oriented export applies that either EXPORT_FILEAPP or EXPORT_FILETRUNC must be used. No default action exists.

Stream-oriented export
Stream-oriented export is preferrably used when exporting large files from a TRIPxml database to a client application (based on TRIPjtk or TRIPclient). To use streamed import, add FOXML_STREAM to the filter_options member of filter_data. It is important also to assign either IEOPT_FILENAME or IEOPT_MEMORY to the tdb_options member of filter_data. You must also add a FILEURL parameter to the filter_arguments string, specifying the URL to which you want TRIPxml to upload the XML file with a HTTP POST message, e.g. "FILEURL=http://mybox:1234/data.xml".

Remember that the best way to program stream oriented export is to use the functions in the regular TRIPxml C API for a much easier programming interface. Please refer to the compressed HTML online help file TXCAPI.CHM for more details.

Server-Side API calls

TRIPxml C API
For C or C++ based server applications, the recommended method to access TRIPxml is to use the TRIPxml C API. Please refer to the HTML help file TXCAPI.CHM for API reference and usage examples.

Client-Side API calls

XML and TRIPcom
From version 2.0-1 of TRIPclient, the interface to the Record object in includes the XML-specific methods *CopyFromXMLFile* and *CopyToXMLFile*.

The TCXML add-on component for TRIPclient is from version 3.0 of TRIPxml no longer distributed. Its classes has been added to TRIPcom in TRIPclient version 2.5-0. These classes provide additional support for XML import and export to TRIPclient-based applications.

Please refer to TRIPclient documentation for details.

TRIP Java Toolkit
TRIPjtk supports programmatic access of TRIPxml through its classes XMLRecord and XMLLink. Version 1.0 of TRIPjtk supports TRIPxml up to version 2.0, and version 1.1 of TRIPjtk supports TRIPxml up to version 2.1.

TRIPjxp and TRIPnxp
TRIPjxp and TRIPnxp from version 2.1 provide the most complete APIs for development of TRIPxml applications. Please refer to the TRIPjxp and TRIPnxp documentation for details.

Using the XML Tools

TXPUTS
This is a server-side command line tool that is used to import an XML document into a TRIPxml database.

```
txputs command line options

Options:

   -d <database>    Name of database (mandatory)

   -u <username>    Name of user to access database as.

   -p <password>    Password for user.

   -r <url>         Store URL information.

   -e <record id>   Replace (update) named record.

   -s <portno>      Stream load via specified TCP port.
                    Pass 0 for dynamic.

   -m <importmode>  1=File, 2=Url (default: 1)

   -b               Do not store original document as blob.

   -v               Validate XML document.

   -x               Store non-xml files
                    (using TRIPview if installed).

   -l <loglevel>    Log level. 0=off, 1=errors, 2=warnings,
                    3=debug

   -i               Index the the database after successful
                    modification.
```

The txputs tool will always use Unicode sessions with TRIPsystem 6.0 and later.

To import an XML document into TRIP with subsequent indexing, issue this call:

```
txputs –i –l 2 –d mydatabase –u myuser –p mypassword myfile.xml
```

You can also use wildcard characters in the filename, so that you can import several XML documents at once:

```
txputs –i –l 2 –d mydatabase –u myuser –p mypassword *.xml
```

If you want to store your DTD (or any other file) in the database as well, you type the following:

```
txputs –i –l 2 –x –d mydatabase –u myuser –p mypassword myfile.dtd
```

Note that you cannot mix non-XML files and XML files when importing. If the "-x" option is specified, all files will be treated as non-XML, even if they actually happen to be XML. Another important thing to remember is that if TRIPview is installed on the server, it will automatically be used to extract the text from non-XML files, making them searchable. So, if TRIPview is not installed, all non-XML files stored in a TRIPxml database will not be searchable by content.

TXGETS
This is a command line tool that extracts stored XML data from a TRIP database, thus reconstructing the original XML file.

TXGETS has a CGI-style command line interface that is adapted for use from a web server. It is also possible to be run it manually from the command prompt – either by defining the QUERY_STRING environment variable with the query data, or putting the query data as input argument to the program in a quoted string.

Highlighting of hits in search results are possible if you tell TXGET to work from an existing search in an existing SIF file. The highlighting works by inserting tags belonging to a separate namespace "*http://www.tieto.com/trip/xml*". The tags do not have content, so they will not disturb the displaying of the XML information using style sheets that does not take the highlighting tags into account. The TRIPHLBEGIN tag is used to mark the start of a highlighted section, and the TRIPHLEND tag is used to mark the end of such a section.

A highlight-section may look like this:

```
We <TRIPHLBEGIN
xmlns="http://www.tieto.com/trip/xml"/>promise<TRIPHLEND
xmlns=" http://www.tieto.com/trip/xml"/> that
```

This could be formatted, using a stylesheet adapted to the highlighting tags as:

```
We promise that
```

in which the <TRIPHLBEGIN> element is replaced by <B> and <TRIPHLEND> by </B> (assuming that output is to be HTML).

Parameter Name	Full Name of Parameter	Explanation
DB	Database	The name of the TRIPxml database
SEARCH	Search number	The number of the search to use as data set. The use of this parameter will cause txget to ignore the DB parameter and ask for reopening of an existing SIF file - either specified by name or belonging to a database user (see below). The results produced using this parameter will have the hit locations automatically marked up.
SIF	SIF file name	Use this parameter to specify the unqualified name of the SIF file to use if it is not conforming to the "username.SIF" naming. Note that the SIF file is assumed to be located in the directory identified by the TDBS_SIF parameter, or in the temp directory if TDBS_SIF is unspecified.
RID	Record id	The RID of the requested record or the (ordinal) number of the record in a search set.
MODE	Reconstruction mode	Set this to 1 for reconstruction of the XML document from its parts. Set to 0 or do not use if the XML document blob (from the D_XMLDOC field) is to be returned.
TYP	Content type	Content type is highly recommended when sending the document via a web server. The TYP parameter is a numeric code. These types are defined so far: 1. text/html 2. application/msword 3. application/pdf 4. text/plain
FNO	Field number	Field number for the blob field. The XML blob is stored in field 1 (preset number for the D_XMLDOC field). If blobs are stored in other formats, other

		numbers may be used. This parameter is ignored when MODE is 1.
USR	User name	The name of the database user to connect to the local TRIP server as. Defaults to "system".
PAS	Password	The password of the named database user. Defaults to "z".
HOST	Host name	This parameter only applies to the version of txget which is compiled for TRIPclient. Name of computer where TRIP server is running. Defaults to "localhost".
LOG	Log level	Enable logging to stderr. Log levels are: 0 - fatal errors 1 - errors (default) 2 - errors and warnings 3 - debug
PORT	Port number	Enables streaming I/O. Pass the number of any free TCP port, or 0 (zero) for dynamic port allocation.
UTF	Unicode mode	Toggles use of Unicode databases. Pass non-zero to enable, and 0 (zero) to disable. Default is 0.
HIGHLIGHT	Hit Markup	If a search number has been specified and this parameter has been assigned a non-zero numeric value, then hit markup will be applied to the returned document.

TXGET Usage Examples

Value of QUERY_STRING or first argument	Explanation
"DB=XALICE&RID=1&FNO=1"	Extract the XML blob stored in the first field of the first record in the XALICE database.
"HOST=frodo&DB=XALICE&RID=1&FNO=1"	Same as above, but used from a pc with TRIPclient installed, sending the request to the host frodo.
"SEARCH=2&RID1&MODE=1"	Extract the reconstructed XML document from the first record i the second search set using existing default SIF file.
"SEARCH=2&SIF=xml123.sif&RID=1&MODE=1"	Extract the reconstructed XML document from the first record in the second search set using SIF file xml123.sif.
"DB=XALICE&RID=1&MODE=1&PORT=0&LOG=3"	Extract the reconstructed XML document from the first record in the database with streaming I/O and dynamic port allocation. Logging is set to the highest level.

When running txgets as a CGI program, the URL might then be:

```
http://www.myserver.com/txget?DB=XALICE&RID=1&FNO=1
```

for producing an XML document from the database XALICE, extracting the stored XML blob from field number 1.

UPDLINKS

The updlinks utility is used to bring an out-of-synch link database up to date. You can use it to enable link support on an existing link database, or to remove references to removed XML documents, or to create link records for XML documents that have been imported while the link support has been temporarily disabled.

```
updlinks command line options

USAGE: updlinks <command> [options]

Commands are:

        m <dbname>      Refresh link database entries for
                        named XML database.

        c               Clean up link database.

Options are:

        -h              Show usage help (this text).

        -i              Ignore XML_NOLINKS/XML_LINKS setting.

        -u <usr>        User name.

        -p <pwd>        Password.

        -b <dbname>     Use this database as link database.
```

To create link database entires for a database which previously has none, ignoring XML_LINKS setting, and using MYLINKDB as link database.

```
updlinks m –i –b mylinkdb
```

Please note that you must run txlinker (see below) to actually get any data in your link database entries!

TXLINKER

The txlinker utility processes link information in new or modified TRIPxml records. When links are found, the corresponding link database record for the TRIPxml record from which the link emanates is updated with the link information. If the target document does not exist as imported into any one XML database the link information is still created, referring to the URL on which the file supposedly is to be found.

The txlinker matches the URLs in the link specification with the value in the field D_URLALIAS. So if link support is enabled, documents should be imported with a specified URL.

Execution of txlinker is required to get link data into the link database. The import filter (which is called by txput) does not do that due to performance considerations. If link support is always enabled, you really want to configure execution of txlinker as a timed job using cron or something equivalent.

```
txlinker command line options
USAGE: txlinker -u username -p password [options]
Options are:
        -h              Show usage help (this text).
        -v              Verbose.
        -i              Ignore XML_NOLINKS/XML_LINKS setting.
        -b <dbname>     Use this database as link database.
```

To update link information for all XML databases into the default link database, simply execute txlinker without any arguments.

MKLINKDB

The mklinkdb utility is used to create a link database.

```
USAGE: mklinkdb username password databasename
```

Applications

Server-side applications

Server-side applications can be written using the TRIPxml C API with the TRIP toolkit (a C application programming interface to TRIP) that is delivered with TRIPsystem.

See the *xmlput* sample for an idea of how to create a server side application using the TRIP toolkit API function TdbImport.

In the compressed HTML online help file TXCAPI.CHM, usage examples for the TRIPxml C API can be found.

XPI Applications

The products TRIPjxp and TRIPnxp have the most complete API for the development of TRIPxml applications. Please refer to the "TRIPnxp and TRIPjxp Programmer's Guide" document (available with TRIPnxp and TRIPjxp) for detailed information on how to create TRIPxml applications with these SDK products.

Networked applications using TRIPclient

The TRIPclient specific components that were available in previous versions of TRIPxml have been discontinued.

However, the objects from the TCXML COM component have been integrated into TRIPcom in TRIPclient from version 2.5-0. Note that these objects (TripTCXml, TripXmlRecord, TripXmlLink and TripXmlProperty) now have different program ids and different class ids. Existing applications that want to continue to use these objects will therefore have to be adjusted to use the correct references to these objects.

Networked applications using TRIPjtk

In TRIPjtk 1.1-0 and later, there are two samples for TRIPxml applications; jtxput and jtxget. These samples demonstrate how to implement TRIPxml import and export with Java.

CGI-style applications

A CGI application is a web application that can be used with most web servers. See the xmlget sample for an idea of how to create a (server-side) CGI program.

Appendix A - API Reference

This appendix describes the TRIP toolkit APIs to use for low-level access of the TRIPxml functionality. The recommended way is to use the TRIPxml functionality integrated into TRIPnxp and TRIPjxp version 2.1 and later. The TRIPxml C API can also be used for server-side TRIP Toolkit applications. The TRIPxml C API, TRIPnxp and TRIPjxp are documented separately.

The filter_data structure

This is the structure used by the API functions TdbImport and TdbExport:

Data Type	Member Name	In/Out	Description/Usage
TdbHandle	record_control	In/Out	The record control of the record into which an XML document is to be imported, or the record that contains an XML document that is to be exported. If the filter created a record control, it is returned in this member.
TdbHandle	cursor	In/Out	For imports; a record cursor. For exports; a cursor to the D_XMLDOC field. Is optional for exports. If the filter created a cursor, it is returned in this member.
TdbHandle	filter_address	In/Out	Contains a handle to the called API routine on return. This handle may be used on subsequent calls to boost performance. Not fully implemented in this version!
char	filter_name[32]	In	The name of the filter to call. For TRIPxml, the name of the import filter is "tripxmlput", and the export filter name is "tripxmlget".
char	filter_lib_env[32]	In	The name of the logical symbol in the TRIPrcs file that specifies the fully qualified path to the TRIPxml shared library.
void*	buffer	In/Out	Field with many uses, dependent on what the tdb_options member say is contained herein. May be allocated memory, placeholder for allocated memory, name of file, or file pointer.
int	buffer_length	In/Out	Length of buffer.
char*	filter_arguments	In	Filter-specific string of arguments. Set to NULL or empty string if no arguments are passed.
int	arg_length	In	Length in bytes of the content of the filter_arguments member, including the terminating NULL-character.
int	filter_options	In	Filter-specific options.
int	tdb_options	In	Import/export specific options.
int	blockno	In	Block number (only for block-oriented i/o – see below). Set to 0 for final block, 1 for first in a sequence of several, etc.
int	errorcode	Out	Filter-specific error code (may also be informational or warning).

Data Type	Member Name	In/Out	Description/Usage
char	errortext[256]	Out	Textual, filter-specific, error message.

Valid values for the filter_options member

Filter Option Name	Value	Description
FOXML_NEWREC	1	TdbImport: Create a new record.
FOXML_REPLACE	2	TdbImport: The opposite of FOXML_NEWREC. Is implicitly set if FOXML_NEWREC is not set. The record_control and cursor fields are mandatory in combination with this option.
FOXML_VALIDATE	4	TdbImport: Tells the XML parser to validate the XML record if any internal/external DTD subset have been seen.
FOXML_GETBYID	8	TdbExport: Retrieve document by URL or record name (ID). Specify URL by adding the URL parameter to the filter_arguments member. Specify ID by adding the ID to the filter_arguments member.
FOXML_REMAKE	16	TdbExport: Recreate the xml document from its parts. Do not use data stored in the D_XMLDOC field.
FOXML_HILIGHT	32	TdbExport: Insert hit-markup in the extracted xml document, so that the displaying application can highlight the hits. The option FOXML_REMAKE must also be set.
FOXML_NOBLOB	64	TdbImport: Do not store the original document.
FOXML_STREAM	256	Specifies that the caller wishes to perform stream-oriented I/O.

Valid values for the tdb_options member

Tdb Option Name	Value	Description
IEOPT_FILENAME	1	The buffer member contains file name.
IEOPT_FILEPTR	2	The buffer member contains file pointer (FILE*).
IEOPT_MEMORY	4	The buffer member points to memory area.
EXPORT_ALLOC	32	TdbExport: Declares that export filter should allocate memory for data (the buffer member is changed to point to newly allocated memory). Used in combination with IEOPT_MEMORY.
EXPORT_FILEAPP	256	TdbExport: Declares that export filter should append XML document to a file. Used in combination with either IEOPT_FILENAME or IEOPT_FILEPTR.

Appendix B - Encodings

Supported Encodings

TRIPxml supports, via version 4.2 of the ICU library (International Components for Unicode), all major encodings including:

- ASCII
- UTF-8
- UTF-16 (Big/Small Endian)
- UCS4 (Big/Small Endian)
- EBCDIC code pages
- GB2312 and BIG5
- IBM037 and IBM1140 encodings
- ISO-8859-1 (aka Latin1)
- Windows-1252

For a more complete list of encodings, see the ICU homepage http://icu-project.org or IANA's list of character set names at http://www.iana.org/assignments/character-sets.

Best Practices

The best choice in most cases is either utf-8 or utf-16. Advantages of these encodings include:

- *The best portability*. These encodings are more widely supported by XML processors than any others, meaning that your documents will have the best possible chance of being read correctly, no matter where they end up.
- *Full international character support*. Both utf-8 and utf-16 cover the full Unicode character set, which includes all of the characters from all major national, international and industry character sets.
- *Efficient*. Utf-8 has the smaller storage requirements for documents that are primarily composed of of characters from the Latin alphabet. Utf-16 is more efficient for encoding Asian languages. But both encodings cover all languages without loss.

The only drawback of utf-8 or utf-16 is that they are not necessarily the native text file format for operating systems, meaning that common text file editors and viewers may not be possible to use directly.

A second choice of encoding would be any of the others listed in the table above. This works best when the xml encoding is the same as the default system encoding on the machine where the XML document is being prepared, because the document will then display correctly as a plain text file. For systems in countries speaking Western European languages, the encoding will usually be iso-8859-1.

A word of caution for Windows users: The default character set on Windows systems is windows-1252, not iso-8859-1. While Xerces C++ does recognize this Windows encoding, it is a poor choice for portable XML data because it is not widely recognized by other XML processing tools. If you are using a Windows-based editing tool to generate XML, check which character set it generates, and make sure that the resulting XML specifies the correct name in the encoding="..." declaration.

Appendix C - TRIPrcs Settings

The following TRIPrcs settings under the nonprivileged section are specific to TRIPxml. All settings that have a valid value set of 0 and 1 are boolean – i.e. 0=off, 1=on.

Setting	Valid Values	Default Value	Description
TDBS_XMLSHR	special	special	The value of this setting is the fully qualified path to the xmlfilters library.
TDBS_ASELIBS	special	special	The value of this setting is a comma-delimited list of the names of all settings referring to ASE or filter libraries. In order for TRIPxml to work, this list must include TDBS_XMLSHR.
XML_LINKS	0, 1	0	Link support
XML_LINK_DB	special	special	Link database name if other than XML_LINK_DB
XML_NAMESPACES	0, 1	0	Namespace support.
XML_SCHEMA	0, 1	0	Schema support
XML_STRICT_SCHEMA	0, 1	0	Strict schema validation
XML_STRICT_VALIDATION	0, 1	1	If documents are imported with validation, the strict validation means that the parser will load any external references required to validate the document such as DTD files, etc.
XML_NOENTITYRESOLVER	0, 1	0	The entity resolver is used to redirect requests for DTDs and schemas to the proper location. E.g. if a DTD has been imported into a TRIPxml database, the entity resolver will export the DTD from there. Turning on this setting means that an entity resolver will not be created. This will break any import where validation is enabled and a DTD or XML schema file needs to be loaded!

Appendix D - Supported XPath Syntax

This appendix is an overview of the part of the XPath 1.0 syntax supported by TRIPxml. Refer to the W3C (http://www.w3.org/TR/xpath/) for a complete description of XPath 1.0.

Definitions

Node
A node is a node in the tree structure that makes up an XML document. There are several types of nodes, e.g. element, attribute and text;

```
<element attribute="">text</element>
```

In a TRIPxml database, every node is stored in a tuple (associated subfields across more than one field), with an optional associated text value in a part record with the same number as the tuple (subfield) number.

Node Set
A node set is what an XPath expression evaluates to. A node is usually an element, an attribute or text. Every node in a node set "knows" from which position in the document it comes, and can therefore be used as context for further XPath expressions.

In TRIP terminology, a node in a node set is a reference to a specific tuple, or subfield, in a record in a TRIPxml database.

Location Step
Every location step is an XPath expression in itself, and as such it has a context. A context is the XML node(s) relative to which the expression is to be evaluated. A bit like "../../somedir" in a file system is relative to the current directory.

A location step consists of the three parts *axis*, *node test* and *predicate* in the form `axis::nodetest[predicate]`.

1. An *axis* determines the direction of the selection that the expression is to go, relative to context.The most common axis type is "child". The child axis says that the expression matches nodes that has the context node (or nodes) as direct parents.

2. A *node test* makes a selection of nodes from the specified axis. A node test can be the name of an element, a node type, a wildcard, etc.

3. A *predicate* is a filter of sorts. It further limits the set of nodes that matches the location path expression. This filtering can be made in many different ways. A common one is to specify specific attributes and values of specific attributes. Only the nodes that matches the condition of the predicate will be included in the final node set for the current location step.

For example, in the expression:

```
child::para[position()=1]
```

child is the axis, *para* is the node test and *[position()=1]* is the predicate. In plain English, this means "*select the first para element that is child to one of the nodes in the context node set*".

Axis Types

Axis	Description
child	Selects of nodes that are direct children to nodes in the context node set.
attribute	Selection of nodes that are attributes to (element) nodes in the context node set.
descendant	Selects nodes that have any of the nodes in the context node set as parent or ancestor. In other words, this selects the sub trees where the nodes in the context node set are roots.
ancestor	Selects nodes that are parents or ancestors to the nodes in the context node set.
ancestor-or-self	Selects nodes that are part of the context node set, or are parents or ancestors to nodes in the context node set.
descendant-or-self	Selects nodes that are part of the context node set, or have a node in the context node set as parent or ancestor.
self	Selects all nodes from the context node set.
following-sibling	Selects nodes that have the same parent as any of the nodes in the context node set, and are located after the context node in question.
preceding-sibling	Selects nodes that have the same parent as any of the nodes in the context node set, and are located before the context node in question.
parent	Selects nodes that are parents to nodes in the context node set.
following	Selects all nodes that follows the nodes in the context node set.
preceding	Selects all nodes that precedes the nodes in the context node set.

Functions

The following functions are supported for use in predicates:

Function	Description

position()	Evaluates to the position of a context node. Must be used as an lvalue in a comparison. For example:
	`//child::para[position()=1]`
	which also can be written as:
	`//child[1]`

last()	Evaluates to the last node in the context node set. Must be used as an rvalue in a comparison with position(). For example: `//child::para[position() = last() - 1]` Which selects the second-to-last node from the context node set.

Function	Description
contains(nodeset,value)	Selects nodes that have TEXT contents (as descendant nodes) that contains the specified value. For example: `//sect1[contains(para,'welcome')]` This expression selects sect1 elements that have para children that contains TEXT nodes in which the word "welcome" can be found.

Predicate Usage

The existence of an attribute
TRIPxml supports the following predicate syntax for checking if an attribute exists. These expressions are equivalent:

```
[@ID]

[attribute::ID]
```

Applied to a location step, this limits the selected nodes to those that have the specified attribute ("ID" in this example).

Attribute value check
The most common predicate is probably the one that checks the value of an attribute. These expressions are equivalent:

```
[@lang="EN"]

[attribute::lang="EN"]
```

Applied to a location step, this limits the selected nodes to those that have the specified attribute with the specified value.

Exact text node contents
Whether or not to use an attribute or a text node under an element to represent a particular value is in many cases not a clear-cut choice. The equivalent of doing an attribute value check for the text content of an element is this:

```
[. = "Enterprise"]

[text() = "Enterprise"]
```

Applied to a location step, this limits the selected nodes to those that have exactly the specified value as their text contents. No truncation is performed.

Truncated text node contents
The only XPath function supported by TRIPxml that can perform a truncated search is the contains() function. If there is a text node that contains "they welcomed us to their party", then we can use this predicate expression:

```
[contains(.,"welcome")]

[contains(text(),"welcome")]
```

Applied to a location step, this limits the selected nodes to those that have exactly the specified value as their text contents. No truncation is performed.

Truncated attribute text node contents
A variant on the truncated text node contents, we replace the first argument of the contains function with an expression that evaluates to an attribute.

```
[contains(@name,"TRIP")]
```

```
[contains(attribute::name,"TRIP")]
```

Applied to a location step, this limits the selected nodes to those that have an attribute "name" whose value contains "TRIP". So we would for instance be able to find "TRIPxml" this way.

Comparison Operators
All the six normal comparison operators are supported; =, !=, >, <, >=, and <=.

Note that comparison is always text based. This means that a non-equi comparison with numerical data may not yield the expected results.

Node position
Perhaps more useful in fragment retrieval than in querying, limiting the node set by node position is also possible with TRIPxml. The functions position() and last() can be used here, as well as the abbreviated form of just using a number

Selecting a node at a specific position from the context node set, the following expressions are equivalent:

```
[position()=1]
```

```
[1]
```

Selecting a range of nodes:

```
[position() < 10]
```

Selecting the last node in the set:

```
[position() = last()]
```

```
[last()]
```

Verifying that the context node set has a certain size (here exactly 2 nodes):

```
[last()=2]
```

Selecting nodes at the end of the set, e.g. the last two ones:

```
[position() >= last() - 1]
```

Multiple predicates in a single location step
The keywords "and" and "or" can be used within a predicate. Multiple predicates can be listed one after the other within the same location step, in which case there is an implicit "and" operation between each predicate.

This combination exemplifies both variants:

```
[@type="S" or contains(.,"TRIP")][position()<10]
```

This would select the first nine elements that either have an attribute "type" having the value "S" or contains the text "TRIP".

Additional Examples

Expression over Axis "child"
The following expressions are equivalent:

```
/child::doc/child::sect1
```

```
/doc/sect1
```

Expression with axis "attribute" in predicate
The following expressions are equivalent:

```
/child::doc/child::sect1/child::title[attribute::ID="chhist"]
```

```
/doc/sect1/title[@ID="chhist"]
```

Expression with axis "parent"
The following expressions are equivalent:

```
/descendant-or-self::node()/attribute::ID/parent::*
```

```
//@ID/..
```

Expression with multiple axes in a predicate
This selects the "para" elements that are located somewhere under a "sect1" element whose direct child element "title" has an attribute "ID" with the value "chhist".

```
//para[ancestor::sect1/title/@ID="chhist"]
```

Expression with axis "descendant" and multi-axis predicate
Here we want the "para" elements located somewhere under a "sect1" element whose direct child element "title" has an attribute "ID" with the value "chhist".

```
//sect1[title/@ID="chhist"]/descendant::para
```

Expression with axes "ancestor-or-self", "parent" and "descendant"
This expression selects the "sect1" elements under which there is an element "title" whose attribute "ID" contains the text "hist". Furthermore, we want to make sure that there are "para" elements under the selected "sect1" elements, and that these "para" elements or one of their ancestors have an attribute "lang" with the value "EN".

```
//para/ancestor-or-self::node()[@lang="EN"]/
descendant::title[contains(@ID,"hist")]/parent::sect1
```

Note: this is all a single, long line. The line break is for clarity only.