Keystone Administration

Adam Dickmeiss

Sebastian Hammer

Mike Taylor

Marc Cromme

Anders Sønderberg Mortensen

Keystone Administration

by Adam Dickmeiss

by Sebastian Hammer

by Mike Taylor

by Marc Cromme

by Anders Sønderberg Mortensen

Copyright © 2002, 2003, 2004, 2005 Index Data ApS

The Keystone Library (TKL) (http://www.indexdata.dk/keystone/) is a web application framework that eases the construction of library web portals and other web sites with semi-structured content. It is used to build literature web portals which are highly user customizable, and literature search engines based on the Z39.50 or SRW protocol.

TKL consists of many modules and pluggable handlers used to add specialized functionality. Among the available modules are OAI harvesters and web harvesters. The existing handlers are specialized to integrate Z39.50 queries, SOAP queries, SQL queries and much more into the TKL framework. Custom designed pluggable components can be freely added.

The Keystone Library is based on XML and XSLT technologies. It currently uses Sablotron as processing engine, and PHP as application glue language. Helper utilities are written in Perl and Tcl.

This documentation describes the installation and programming interfaces of The Keystone Library (TKL) (http://www.indexdata.dk/keystone/).

CVS id: $Id: tkl.xml,v 1.7 2005/06/23 10:03:58 adam Exp $

Table of Contents

1. Introduction ... 1

1.1. Basic principles ... 1
1.2. The base portal Bibliotheca ... 2

2. Installation ... 4

2.1. Debian ... 6
2.2. Red Hat ... 8
2.3. Web Server ... 10
2.4. XML parser ... 11
2.5. DOM XML ... 11
2.6. Sablotron XSLT ... 11
2.7. Z39.50 Support ... 12
2.8. PHP ... 12
2.9. PHP/YAZ ... 13
2.10. Installing Keystone files ... 13
2.11. Enabling Keystone in Apache 1.3.X ... 15
2.12. Enabling Keystone in Apache 2 ... 16
2.13. Installing the Sample Portal ... 16
2.14. Starting the Zebra server ... 17
2.15. Testing it ... 17
2.16. Virtual hosting ... 18

3. Content administration ... 20

3.1. Directory structure ... 20
3.2. Documents ... 22
3.3. Schemas ... 22
3.4. User Administration ... 23
3.5. The base portal Bibliotheca ... 23
3.6. Searching ... 24

4. Managing end-user interfaces ... 25

4.1. Document context and the user shell ... 25
4.2. Parameters ... 26
4.3. Session parameters ... 26
4.4. Supporting functions ... 26
4.4.1. tkl-admin-header ... 27
4.4.2. tkl-args ... 27
4.4.3. tkl-default ... 28
4.4.4. tkl-file ... 29
4.4.5. tkl-file-exists ... 29
4.4.6. tkl-find ... 30
4.4.7. tkl-grant ... 32
4.4.8. tkl-header ... 33
4.4.9. tkl-help ... 33
4.4.10. tkl-mail ... 33
4.4.11. tkl-path ... 34
4.4.12. tkl-search ... 35
4.4.13. tkl-set-footer ... 36

4.4.14. tkl-soap 36
4.4.15. tkl-sql 38
4.4.16. tkl-time 40
4.4.17. tkl-unique 40
4.4.18. tkl-user 41
4.5. Portal configuration 42
4.6. Debugging 43
4.7. The base portal Bibliotheca 43
4.7.1. User interface 43
4.7.2. Overall page structure - interface.xsl 43
4.7.3. portal.xsd/xsl 44
4.7.4. subject.xsd/xsl 45
4.7.5. link.xsd 45
4.7.6. document.xsd/xsl 45
4.7.7. search.xsl 45
4.7.8. news.xsd - newspage.xsl 46
5. Keystone and OAI Metadata Services 47
5.1. Keystone as an OAI repository 47
5.2. Bibliotheca example OAI repository 48
5.2.1. Identify 48
5.2.2. ListMetdataFormats 48
5.2.3. ListSets 49
5.2.4. ListIdentifiers 49
5.2.5. GetRecord 49
5.2.6. ListRecords 50
5.3. OAI harvesting from within Keystone 50

Chapter 1. Introduction

The Keystone Library (TKL) (http://www.indexdata.dk/keystone/) is a programming framework for fast deployment of information-driven web portals.

By *portals*, we mean websites that make information resources available. These may be corporate or personal homepages, “subject gateways”, or any number of other types of resources. Along with Keystone, we provide a “base portal” called Bibliotheca, which supports a hierarchically organised collection of metadata about external resources, but which is also capable or presenting its own content, such as articles, news entries, etc. By building on top of the base portal, or by constructing a completely new template, practically any type of information-based website can be constructed.

The Keystone Library can be seen as an alternative to traditional, closed “content management” systems, or to custom-built, database-driven websites. Especially in the case of sites that are naturally organized as collections of documents of various types, Keystone can be a useful tool. The system is particularly well-suited to situations where the separation of content from presentation is important.

The Keystone Library was developed, and has been extensively tested, using the Apache web server version 1.3. Other servers should work provided that they support PHP and have the necessary extra components.

1.1. Basic principles

A Keystone portal is a collection of *documents* of different types. The documents are typically organized in a directory structure which reflects the logical structure of the portal. The documents are represented in standard XML. A portal generally is made up from documents of different types. Each type is represented by an XML *Schema* that describes which data elements appear in the documents, which elements are repeatable, which are mandatory, what their types are, etc. The XML schema also contains additional information specific to Keystone, for instance whether a given element is multi-lingual, corresponds to a restricted vocabulary, etc.

The Keystone Library includes a content management system which allows editors to manage the file structure and its contents (and hence the structure and contents of the user-facing portal) via a web-based interface. Access control is supported and the system supports the delegation of responsibility for the maintenance of different parts of a portal.

The documents are generally stored as ordinary XML text files on the server’s filesystem. All Keystone documents have the filename suffix `.tkl`. The web server is configured so that when the end-user attempts to access a document with the `.tkl` suffix, The Keystone Library “user shell” is invoked. The user shell is responsible for displaying the document to the user in a suitable way.

For the purpose of display, each document type is associated with a presentation format. In this way, a help file is displayed differently from a subject group or a metadata record, and so on. The presentation formats are realised as XSLT *Stylesheets* which are processed to display each type of document. So the task of the Keystone user shell is, for a given document, to determine which XSLT stylesheet should be used to display the document. Apart from the document itself, the user shell makes a range of information available to the stylesheet, such as where in the directory hierarchy the document is located, whether there are other documents nearby (eg. in subdirectories), etc. It also provides the stylesheet with functions for invoking external services such as information retrieval (searching), remote procedure calls and database access.

In summary, a Keystone portal generally consists of three basic elements:

- Documents, represented in XML
- Document types, expressed as XML schemas
- Presentation formats, expressed as XSLT stylesheets

The documents are the pivot of the system - they constitute the *content* of a portal. The document types are primarily used in portal maintainance. The presentation formats are used when a user views a document.

The maintenance and user-facing sides of Keystone are fully independent. You can replace the content management interface with another system, or use *ad-hoc* scripts or batch jobs to maintain all or parts of the contents of a site. In the same way, you can use the Keystone management interface to provide remote administration for a group of files, and then use different techniques to actually present their content to end-users (assuming you even care about presenting the content to users).

1.2. The base portal Bibliotheca

Together with Keystone we provide an example of a fairly basic type of portal. It is included as an example of how Keystone may be used, and it may be freely used and expanded as a starting point for new portals.

All of the pages of the portal are built around a shared structure, starting with a graphical bar at the top of the page to lend identity to the website. Underneath is a horizontal menu with certain fixed functions, including a search field. Below this, on the left side of the display, is the main menu, which also enables a language selection (if relevant). In the centre of the display is the content window, where the content of each page or document is presented. In the right-hand side is a column for news entries.

Clicking on a news item, the “show more news” link (displayed when not all news entries fit in the left-hand column) or the “News” menu-item in the main menu brings the user to a news page where the full content of all articles in the news directory are displayed.

Clicking on “Help” in the top menu brings the user to a system of help pages.

The “Articles” item in the left-hand menu brings you to a collection of local articles.

Typing in one or more keywords in the search field allows a free-text search through the portal. The administrator controls which directories should be searchable.

Articles and help documents are both based on the same document type, described by the schema document.xsd. This is a general-purpose “article” type with the quality that it can display a “mini-menu” of any sub-documents. This makes it possible to use this simple document type to construct complex structures of documents (such as encyclopaedia articles with multiple chapters).

Chapter 2. Installation

The Keystone Library relies on a number of software components.

If you are running Debian, you can use its package management facilities to handle most of the installation: web server, PHP, XML parser, XSLT, DOM support and Z39.50 support. See Section 2.1, then skip ahead to Chapter 3.

Similarly, on Red Hat boxes, you can use its RPM package management facilities for part of the work. Unfortunately, some our tools are not packaged as RPMs yet. See Section 2.2, then skip ahead to Section 2.9.

If you are not running Debian or Red Hat, skip those section and read the instructions on how to install the components by hand, beginning with Section 2.3.

It is recommended to think twice before upgrading The Keystone Library `TKL` packages: Some functionality has changed and will not be compatible with behavior fond in the pre 1.5.2 releases. We do recommend testing an upgrade in a safe, non-production sandbox before applying to a production server.

Warning

IMPORTANT INSTALL NOTICE:

This release has Sablotron dependencies changed to Index Data's own bug fixed versions 1:1.0-1 and larger, which are conflicting with the original Gingerall versions 1.0.1/1.0.2, and Debian version 1.0.2-2. A bug fixed Sablotron version is available from our web site as tarball and as Debian packages.

The original 1.0.1 and 1.0.2 release has a bug which cripples XSLT handler arguments by applying unnecessary and unwanted mangeling of handler arguments, and another which segfaults Sablotron most unpleasently.

The full description of the bug is found on Sablotrons maillist http://archive.gingerall.cz/archives/public/sablot2004/msg00000.html and the Gingerall bug report with patch is found at http://bugzilla.gingerall.cz/show_bug.cgi?id=2811 http://bugzilla.gingerall.cz/attachment.cgi?id=713&action=view See also the following debian bug reports: http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=311609 http://bugs.debian.org/cgi-bin/bugreport.cgi?bug=311611

Unfortunately, the supplied patch which prevents unwanted mangeling of Args/URI for user-defined scheme XSLT handler calls has not yet been accepted upstream. This forces any Keystone installation to use the patched Sablotron version 1:1.0.1-2 found at our web site. ftp://ftp.indexdata.dk/pub/sablotron/sablotron-1.0-ID-patched-1.tar.gz

The pached version is available as Debian package from our Debian package site.

Furthermore, this Keystone release 1.5.3 has changed the admin XSLT handler calling code to work with our Sablotron patch. All XSLT handler args are modified from

```
document('a-handler:xyz')
```

to

```
document('a-handler:/xyz')
```

(notice the slash) to avoid breaking by non-mangeling.

Notice please that your own XSLT portal code needs to be changed the same way to be compliant with Keystone 1.5.3 and the patched Sablotron 1:1.0.1-2.

Similar changes have to be applied to XSLT include and import statements. For example

```
<xsl:include href="tkl-file:some.xsl"/>
```

has to be changed to

```
<xsl:include href="tkl-file:/some.xsl"/> .
```

2.1. Debian

Index Data's Debian packages must be in your Debian sources list (`/etc/apt/sources.list`). You must add the following two lines:

```
deb http://www.indexdata.dk/debian indexdata/sarge released
deb-src http://www.indexdata.dk/debian indexdata/sarge released
```

Then install using the usual commands:

```
apt-get update
apt-get install tkl
```

The following Debian packages will be installed when installing the Keystone binaries: apache apache-common idzebra libdigest-md5-perl libhtml-parser-perl libhtml-tagset-perl libhtml-tree-perl libmime-base64-perl libmm11 libnet-perl libsablot0 libtclrobot-tcl libtkl-perl liburi-perl libwww-perl libxml-libxml-perl libxml-libxslt-perl libxml-namespacesupport-perl libxml-parser-perl libxml-sax-perl libxslt1 libyaz php4 php4-domxml php4-xslt php4-yaz tkl tklite.

Warning

In case you have controlled the Sablotron versions in your `/etc/apt/preferences`, you need to erase those entries. The Index Data patched Sablotron packages have the epoch number 1, and will therefore be preferred over the original Debian Sablotron packages.

Please make sure that the following lines are found in the PHP4 config file `/etc/php4/apache/php.ini`

```
extension=domxml.so
extension=yaz.so
extension=xslt.so
```

and restart the Apache web server.

The Keystone binaries are found in the packages `tklite_1.4.7-1_i386.deb` (core PHP and XSLT functionality, necessary package), `libtkl-perl_1.4.7-1_i386.deb` (Perl libraries, man pages, necessary package), `tkl_1.4.7-1_i386.deb` (init scripts, man pages, cron scripts, Perl binaries, necessary package), `tkl-oai-harvester_1.4.7-1_i386.deb` (complete OAI harvester including init scripts, man pages, cron scripts, Perl binaries, additional package), `tkl-web-harvester_1.4.7-1_i386.deb` (complete WEB harvester including init scripts, man pages, cron scripts, Tcl binaries, additional package), `libtclrobot-tcl_0.2.0-1_i386.deb` (Tcl extention needed by tkl-web-harvester_1.4.7-1_i386.deb, additional package), and `tkl-doc_1.4.7-1_i386.deb` (this documentation in HTML and PDF).

The image uploading features are not part of the core Keystone packages but are contained in the example portal `bibliotheca`. They do not work properly unless you install the `ImageMagick` package as well.

After installation of one or more portals in the Apache DocumentRoot directory `/var/www` these should be registered in either the system global Keystone configuration file `/etc/tkl.conf`, or in the main Apache configuration file `/etc/apache/httpd.conf`. The files must contain valid Apache Directory stanzas - or virtual host stanzas containing the "Directory" directive. See the commented examples in the configuration file `/etc/tkl.conf`.

While all Apache stanzas can be omitted from `/etc/tkl.conf`, if just included from some other file, the "Directory" tags must be present and correctly pointing to Keystone portal root directories in order to start the indexing and harvesting servers.

Now the installed portals are registered, and the idzebra search daemon of each portal can be indexed, started, and stopped by the root UID using the `/etc/init.d/tkl` script with one of the folowing options:

```
# /etc/init.d/tkl index   [/path/to/portal [/sub/dir]]
# /etc/init.d/tkl start   [/path/to/portal]
# /etc/init.d/tkl restart [/path/to/portal]
# /etc/init.d/tkl stop    [/path/to/portal]
```

The OAI harvesting and web harvesting daemons are started and stopped by the root UID using the `/etc/init.d/tkl-oai-harvester` and/or `/etc/init.d/tkl-web-harvester` script with one of the folowing options:

```
# /etc/init.d/tkl-oai-harvester start
# /etc/init.d/tkl-oai-harvester restart
# /etc/init.d/tkl-oai-harvester stop
```

Please consult the appropriate man pages for detailed information.

```
$ man 8 tkl
$ man 5 tkl.conf
$ man 5 tkl.config
$ man 8 tkl-oai-harvester
$ man 8 tkl-web-harvester
```

Furthermore, the daemons are started after each reboot, and the portals are indexed during each install of the tkl package. That is, installing the portals before installing tkl even removes the burden to remember the initial IDZebra indexing.

Notice also that the default Apache configuration has changed. To enable Keysatone you need to uncomment these lines in the apache config files:

```
# LoadModule mime_module /usr/lib/apache/1.3/mod_mime.so
# LoadModule dir_module /usr/lib/apache/1.3/mod_dir.so
# LoadModule action_module /usr/lib/apache/1.3/mod_actions.so
# LoadModule php4_module /usr/lib/apache/1.3/libphp4.so
```

In case you use the new configuartion layout of Apache, you might prefer to make the requested changes manually using the `modules-config` utility:

```
/usr/sbin/modules-config apache enable mod_mime
/usr/sbin/modules-config apache enable mod_dir
/usr/sbin/modules-config apache enable mod_actions
/usr/sbin/modules-config apache enable mod_php4
```

Be careful not to misspell any of the module names when invoking `modules-config`. It will not report any such errors, but quietly do nothing, leaving you with a hard-to-debug problem down the line.

Finally, manually add the

```
Include /etc/tkl.conf
```

line to `/etc/apache/httpd.conf`

It is possible that the installation of the Sablotron PHP extension will not configure PHP to recognise that extension. If not, you will see messages like `Call to undefined function: xslt_create()`. In this case, manually add the line

```
extension=xslt.so
```

to the end of `/etc/php4/apache/php.ini`

Debian installations are nice and smooth, so skip ahead to Chapter 3 for the fun part of Keystone.

2.2. Red Hat

Index Data provides some Red Hat 9.0 packages, and most of external packages, which our software depends on, can conveniently be installed by the RPM system. This approach is only tested on Red Hat 9.0 systems, although it should work on running Red Hat 8, or even 7.

You need to install the following official RPM packages: `httpd`, `php`, `php-devel`, `tcl`, `tcp_wrappers`, as well as the following packages from Index Data: `libyaz`, `libyaz-devel`, `idzebra`.

For the remaining components there are no official packages. PHP/YAZ as well as Sablotron and PHP/XSLT must be compiled and installed separately.

Install Sablotron first. It exists as a RPM package, but this one will not work with Keystone due to misconfiguring of the bundled Java interface. Therefore, get the tarball from Keystone support (http://ftp.indexdata.dk/pub/tkl/support) or from the official Sablotron (http://www.gingerall.com) site. Configure, make and make install as usual.

Warning

Unfortunately, the Sablotron packages version 1.0 have a nasty bug which segfaults the Keystone applications using the tkl-file xslt macro. This forces any Keystone installation to use the patched sablotron version 1:1.0.1-2 found at our web site.
`http://ftp.indexdata.dk/pub/sablotron/sablotron-1.0-ID-patched-1.tar.gz`

The rest of this guide assumes installation of Sablotron in `/usr/local`.

For PHP/XSLT we need to make a PHP dynamic shared object (DSO) that links with the official RPM packages for PHP. Check the version number of PHP for RedHat (in Red Hat 9 the version is 4.2.2), and fetch a PHP source tarball of the same version number. You may download the 4.2.2 tarball from Keystone support (http://ftp.indexdata.dk/pub/tkl/support), or from the PHP download area. (http://www.php.net/downloads.php)

Then unpack the PHP source and do the following:

```
cd ext/xslt
vi config.m4
```

Remove the check for iconv in config.m4. iconv is installed already and is part of the running PHP! Remove or comment out the three following lines, starting at PHP_SETUP_ICONV.

```
PHP_SETUP_ICONV(XSLT_SHARED_LIBADD, [], [
```

```
  AC_MSG_ERROR([iconv not found, in order to build sablotron you
                                        need the iconv library])
])
```

Now we have a proper `config.m4`, and can generate the `configure` script, and run it.

```
phpize
./configure --enable-xslt --with-xslt-sablot=/usr/local
make
# su
# make install
```

Make sure that the file `/usr/lib/php4/xslt.so` exists.

Now, we need to enable XSLT support in PHP. On a Red Hat 9 system using Apache version 2, we inject the following lines in the PHP included INI files like this:

```
# echo 'extension=xslt.so' >/etc/php.d/xslt.ini
```

On a Red Hat 8 system, we add the following lines to the `/etc/php.ini` configuration file:

```
extension=xslt.so
```

That's it. Restart/start apache and ensure that xml, domxml, xslt, and yaz are all enabled:

```
# echo '<?phpinfo()?>' >/var/www/html/phpinfo.php
# /etc/init.d/httpd stop
# /etc/init.d/httpd start
```

Then open the url `http://localhost/phpinfo.php` and inspect the `phpinfo()` output.

Proceed to Section 2.9.

2.3. Web Server

http://www.php.net works with many different web servers. Keystone has only been tested with Apache (http://httpd.apache.org/) versions 1.3.23-1.3.27 on Linux. We have also tried Keystone on RedHat 8 and 9, which use Apache 2.0.40.

We suggest you compile Apache yourself rather than using a preinstalled one because the Expat library which is included with Apache may conflict with another version of Expat elsewhere. Configure Apache without Expat and with shared libraries enabled, like this:

```
./configure --disable-rule=EXPAT --enable-module=so ...
```

Compile and install as usual:

```
$ make
$ su
# make install
```

Note: On Solaris for DSO to work, you may have to use configure option `--enable-rule=SHARED_CORE`.

2.4. XML parser

Expat (http://expat.sourceforge.net/) is a very common XML parser. On many Linux systems it's already installed, in which case you can ignore the rest of this section. Check if `libexpat.so` or `libexpat.so.1` is available.

If not, get Expat from its download page (http://sourceforge.net/projects/expat/). Configure, compile and install.

```
$ ./configure
$ make
$ su
# make install
```

2.5. DOM XML

GNOME XML (http://www.xmlsoft.org/) offers a DOM API. Check if you already have it on your system by checking if `libxml2.a` or `libxml2.so` is present.

If you don't have libxml already, then download it from the home page (http://www.xmlsoft.org/). Configure, compile and install as always.

2.6. Sablotron XSLT

Sablotron (http://www.gingerall.com/) is a XSLT processor. You can check if it's available by looking for `libsablot.a` or `libsablot.so`.

If these are not present, download Sablotron from the home page. Configure, compile and install as usual.

Warning

Unfortunately, the Sablotron packages version 1.0 have a nasty bug which segfaults the Keystone applications using the tkl-file xslt macro. This forces any Keystone installation to use the patched sablotron version 1:1.0.1-2 found at our web site.
`http://ftp.indexdata.dk/pub/sablotron/sablotron-1.0-ID-patched-1.tar.gz`

2.7. Z39.50 Support

If Keystone uses Z39.50 to provide searching facilities, YAZ (http://www.indexdata.dk/yaz/) and Zebra (http://www.indexdata.dk/zebra/) must be installed. Get the source for those these from the Index Data software area (http://indexdata.dk/software/). Configure, compile and install as usual.

The PHP server must be able to access the `zebraidx` and `zebrasrv` binaries. You must edit the top of `search.php` and ensure that values of `$zebraidx` and `$zebrasrv` are correct.

2.8. PHP

PHP (http://www.php.net/) binds all the XML tools together and a PHP script does the processing of Keystone pages. Download the PHP source from here (http://www.php.net/downloads.php).

Configure PHP

```
./configure --with-apxs=/usr/local/apache/bin/apxs
   --enable-sockets \
   --with-dom=/usr/local \
   --with-xslt-sablot=/usr/local \
   --enable-xslt
```

Note: Do *not* enable yaz for PHP in the configure line above, since the PHP/YAZ shipped with PHP 4 versions is old and outdated. Install PHP/YAZ as a PECL (http://pecl.php.net/) module instead. See Section 2.9.

Option `--with-apxs` tells PHP where the Apache Extension tool is located - the location given here is the default location for Apache (unless changed with `--prefix`). Option `--enable-sockets` enables socket support for PHP - no external library is required. Option `--with-dom` enables DOM support and the argument specifies prefix of libxml2. Option `--with-xslt-sablot` sets the XSLT handler to be Sablotron. Last option `--enable-xslt` enables XSLT.

Tip: If you wish to reconfigure PHP you can edit/rerun the script `config.nice` which includes the most recently used options.

Compile and install PHP

```
$ make
$ su
# make install
```

2.9. PHP/YAZ

PHP/YAZ is now distributed as a PECL (http://pecl.php.net/) module. Get the source from here (http://pecl.php.net/package/yaz), then install as follows:

```
$ tar zxf yaz-<version>.tar.gz
$ cd yaz-<version>
$ phpize
$ ./configure --with-php-config=/usr/bin/php-config
$ make
$ sudo make install
```

Make sure that the file `/usr/lib/php4/yaz.so` exists after installation. If configure above does not find YAZ, specify option `--with-yaz` followed by the YAZ installation prefix (such as `/usr/local/`).

Ensure that PHP/YAZ is loaded by PHP by ensuring that `php.ini` has a line reading:

```
extension=yaz.so
```

Restart Apache after modifying `php.ini` and inspect the output of PHP's phpinfo.

2.10. Installing Keystone files

You can put Keystone support wherever the Web server reads its normal HTML content. A possible location for a default Apache installation is the document root from which HTML content is served, this might be `/var/www/tklite`, or `/var/www/html/tklite` (on Red Hat systems). Check your DocumentRoot directive in your Apache config file for details.

You can download Keystone files, packages, etc from our Keystone area (http://ftp.indexdata.dk/pub/tkl).

Get the Keystone tarball `tklite-1.2.0.tar.gz`. Unpack it, and configure, make and make install as usual.

```
$ ./configure
$ make
$ su
# make install
```

This process will create the sub directory `/usr/local/share/tklite`, where the core files are installed, and the `/usr/local/share/doc/tklite`, where the documentation is installed. Finally, add a symlink to make the files accessible from within the server DocumentRoot directory using the correct one of the following:

```
# ln -s /usr/local/share/tklite /var/www
# ln -s /usr/local/share/tklite /var/www/html
```

Warning

Make sure that the `tklite` directory is physically within the DocumentRoot directory (a symbolic link will do, an Apache `Alias` will not!) otherwise you will get mystifying errors like these:

```
Warning: Unknown scheme 'tkl-header'
Warning: Unknown scheme 'tkl-file'

Error: Error Number 3, Level 0, Fields;
msgtype => error
code => 69
module => Sablotron
URI => tkl-file://interface.xsl
line => 1
msg => unknown encoding " error
```

This is because, although Apache can find the Keystone scripts through the alias, the XSLT processor - which needs to call back into Keystone - knows nothing of Apache's `Alias`es, and so can not find the necessary scripts.

Check that the Keystone install area `/usr/local/share/tklite` is readable by the userid of the web server - often `nobody`. On Debian, the user is `www-data`, and on Red Hat systems it is `apache`. Check your Apache configuration for details, and change ownership or read-access bits recursively if necessary.

2.11. Enabling Keystone in Apache 1.3.X

The Apache configuration must be modified in order to enable Keystone files to be processed by the PHP module. Edit the file `httpd.conf`.

First, the page type `index.tkl` must be added to the list of files for the `DirectoryIndex` directive. Example:

```
<IfModule mod_dir.c>
DirectoryIndex index.html index.php ... index.tkl
</IfModule>
```

Note: Apache module `mod_dir` must be enabled

Second, the tkl-handler must be set. Add the following two lines inbetween the `mod_actions` brackets. (If your redhat configuration does not have the mod_actions brakcets, you can add those lines somewhere near the other AddHandler lines)

```
<IfModule mod_actions>
  AddHandler tkl-handler .tkl
  Action tkl-handler /tklite/shell.php
  ...
</IfModule>
```

Make sure the last line reflects the actual location of the shell script relative to the document root of the server.

Note: Apache module `mod_actions` must be enabled

2.12. Enabling Keystone in Apache 2

You can change the main web-server configuration file `httpd.conf` or add a file that is included by Apache. On some systems, there is already a include directory for extensions such as PHP etc. On Red Hat 8 and 9 that directory is `/etc/httpd/conf.d`.

The following directives has to be added:

```
AcceptPathInfo On
AddHandler tklhandler .tkl
Action tklhandler /tklite/shell.php
DirectoryIndex index.tkl
```

2.13. Installing the Sample Portal

A sample portal called Bibliotheca can currently be found at the URL bibliotheca (http://ftp.indexdata.dk/pub/tkl/bibliotheca-1.0.tar.gz).

Get the basic portal and unpack it in the HTML root of the web server, or any other convenient directory acessible by the web server. It will create sub directory `bibliotheca-1.0`, which you will probably wish to access by a symlink `bibliotheca`. Make sure that this directory and its subdirectories are writable by the user of the web server - often `nobody`. On Debian, the web server user is `www-data`. Check your Apache configuration for details.

The semi-structured XML databases bundled with the portal must be indexed before the search engine can use them. Indexing is done in the root directory of the portal (in this case, the `bibliotheca` directory) by issuing the following three commands as the user the web server runs as:

```
zebraidx -l db/server.log -c db/zebra.cfg init
zebraidx -l db/server.log -c db/zebra.cfg update articles help links news suggest
zebraidx -l db/server.log  -c db/zebra.cfg commit
```

The `init` command is only used once, at the very first time when a new portal is created. The `update` and `commit` commands are used whenever new documents have been added to the portal.

2.14. Starting the Zebra server

You must start the Zebra server for each running portal.

> **Note:** Ideally, this should be handled by PHP/Apache, but we have not yet found an good way to do this.

In your portal's `db/zebra.cfg` file, check that the `profilePath` parameter has the correct value. For a typical Zebra installation, such as one done from Debian or Red Hat packages, the value is `/usr/share/idzebra/tab`.

Check that the PHP scripts correctly refer to zebraidx/zebrasrv, by inspecting the definitions of `$zebraidx` and `$zebrasrv` at the top of `.../tklite/search.php`.

In the root directory of the portal, start Zebra as follows:

```
zebrasrv -l db/server.log -c db/zebra.cfg unix:db/socket &
```

2.15. Testing it

You can check the sample portal by visiting `http://myhost/bibliotheca/` . The administration interface is started by using the URL `http://myhost/tklite/admin.php?cwd=site`.

Check that the search box (toward the upper right-hand corner of the page) works. If you get an error message like this:

```
ERROR!!!! 'Connect failed at target
unix:/usr/local/src/z39.50/bibliotheca/db/socket'
```

That indicates that the Zebra server is not running, or that its socket is not in the correct placed (the `db` subdirectory of the portal's root.) See the previous section.

If you get an error message that says

```
ERROR!!!! 'Database unavailable'
```

you probably forgot to initialize the zebra database, or something went wrong during the initialization phase. See Installing the Portal .

To administrate the `keystone-portal` on `localhost` you would use `http://localhost/tklite/admin.php?cwd=bibliotheca`.

2.16. Virtual hosting

For Keystone to work, the `shell.php` must always be accessible using the same path. To enforce this, either use Apache's `Alias` directive or use a symbolic link.

Consider we have Web server which runs `www.domain` and you want Keystone to run on virtual host `tkl.domain`. The relevant Apache 1.3.X config will look like this:

```
# If you want to use name-based virtual hosts you need to define at
# least one IP address (and port number) for them.
#
NameVirtualHost 1.2.3.4

# Our primary domain
<virtualHost 1.2.3.4>
  ServerName www.domain
</VirtualHost>

# Our Keystone domain
<virtualHost 1.2.3.4>
  ServerAdmin tklmanager@domain
  DocumentRoot /home/tkl/html
  ServerName tkl.domain
  ServerAlias tkl
  ErrorLog /home/tkl/logs/error.log
  CustomLog /home/tkl/logs/access.log common
  Alias /tklite/ /var/www/tklite/
```

```
</VirtualHost>
```

Chapter 3. Content administration

> **Note:** Some the the examples in this section refer to document types associated with the base portal Bibliotheca. They might appear different in another portal with different document types, etc.

The normal way to edit the content of a portal is using Keystone's administration interface (admin). Since the content of the portal is represented as files in the server's file system, it is also possible to maintain the files using a common user shell on the system, or using custom scripts, etc. The admin module allows controlled maintenance of the key portal content using an ordinary web-browser. Editing is controlled to minimize the possibility of mishaps (access is controlled, documents are checked for XML validity prior to storage, updated or added documents are indexed for searching on the fly, etc.).

The admin interface is started by providing the URL path for the script admin.php (the exact URL depends on the installation, with the parameter cwd to give the location of the portal based on the root document directory of the web server.

For instance, if the test portal has been installed under the directory HTDOCS/keystone on the server at www.indexdata.dk (so that the portal can be reached using the URL http://www.indexdata.dk/keystone/), and the administrative scripts are available under the directory HTDOCS/keystone-admin, the admin interface can be invoked like this:

http://www.indexdata.dk/keystone-admin/admin.php?cwd=keystone

If all's gone well, the admin interface main window should now appear, and in the following sections, we will describe the individual elements in this interface.

> **Note:** In the following sections, we will describe the organisation of files in the the server file system in boxes such as this one. Editors or administrators who intend to use the admin interface only to maintain their portal can skip these boxes if they wish.

1: Keystone administration interface - main window

In the top of the admin interface is a horizontal menu containing special functions. Underneath is the Path from the root of the portal to the current directory. Then comes a listing of sub-directories, if any, and finally a list of documents in the current directory.

3.1. Directory structure

The directory structure of a portal will generally have at least a rough correspondence to the structure of the portal (although this is not a requirement or a neccesity). In the base portal Bibliotheca, we find the following main directories:

- Articles, holding local articles
- Help, holding one or more documents comprising a help system
- links, holding a hierarchy of subject groups and external ressources
- news, holding a collection of news items of relevance to the portal

(It's important to realise that this list is not cast in stone - the portal designer is free to add more, delete unwanted directories, or start from scratch and design his own structure).

You can always delete empty directories, rename directories or create new directories, but the display formats associated with a portal may expect to find some directories or files in certain locations (for the base portal, this includes the link directory which is expected to be called 'link'.

If you click on a directory name, the admin interface moves to that directory. The directory path in the top of the display can always be used to retrace your steps.

The admin interface shows files and directories located at the given level of the server filesystem. However, some system-specific directories are hidden to simplify the interface.

The main directory - or root directory - of a portal is defined as the directory what contains a file by the name tkl.config. All functions in the system use this directory as a reference when resolving directory paths.

When displaying a file, the administration interface shows both the filename and, if found, the contents of the *title* XML element of the document. When displaying directories, the admin interface attempts to display both the filename and the contents of the *title* element of an index.tkl file in the directory, if one is found.

Under the directory listing is a list of the documents stored in the given directory. Most documents can be edited by anyone with editing privileges for the directory, with a couple of important exceptions. The files users.tkl and directory.tkl may only be edited (or viewed) by users with administrative privileges. The file users.tkl lists the administrative and editorial users. The usage of the file directory.tkl is described hereunder.

If a document contains a file called index.tkl, then that is the document the user will see if he attempts to view the directory (ie. without referring to an explicit filename within the directory). This corresponds exactly to the file index.html in a conventional website.

All directories *may* contain a file called directory.tkl. It defines different qualities that may be associated with a directory (and its subdirectories, except those that themselves contain directory.tkl files). The directory.tkl file (according to the document type directory.xsd) may contain the following elements:

Searchable

Determines whether the subdirectory (and subdirectories) should be searchable.

Auto index type

Determines whether an index.tkl file should automatically be created when a new subdirectory is created (and if so, what type/schema it should belong to).

Allowed schema (repeateble)

Determines whic document types are allowed in a given directory. For instance, in the base portal Bibliotheca, under the "news" directory, only news items may be created, not other types of documents. You can also control how new files of a given type should be named. The 'allowed schema' construction reduces the workload of editors, and ensures that administrative work can be delegated without risk of harm to the portal structure.

3.2. Documents

All documents can (if you have the privileges) be edited using a common editing window. Different document types will look differently in the window, with different input fields, etc.

The editor has buttons to validate and store the document, or to return to the admin interface without changing the document.

Some element types may be associated with type-specific controls. For instance, an element designated to hold an URL will have a button to check if the URL is 'live' and check if there are already documents in the portal which refer to this URL. Date fields may have a button to automatically insert today's date, while text entry fields designed for larger amounts of text have a "focus" button which pops up a larger edit window.

For elements which are language-dependent, the editor is asked to fill in the element in all relevant languages.

In the documents, multi-lingual fields are simply repeated, with different xml:lang attributes associated. Prior to processing, the user-shell filters the document (and stylesheet) to remove any elements which belong to another language than the user's current language.

3.3. Schemas

Schema editor is not yet described here.

3.4. User Administration

The user list is located in the portal main directory. It can only be edited by users with administrator privileges. Here, you can create new users, or change attributes associated with existing users.

3.5. The base portal Bibliotheca

The base portal consists of four main areas, which are visible as subdirectories when the admin-interface is opened under the root directory.

- News - with the news articles shown in the news window
- Articles - with the local articles in the portal
- Help - with the documents which constitute the help system
- links - which contains the top of the subject hierarchy

The news items have the simplest structure. If you go to this directory (with the admin interface), you will find that you can only create files of one type - news.xsd. To create a new article, you simply click on the "create document" button and fill in the given fields. The file index.tkl in this directory cannot be edited, since it has no real content (and no associated schema) - it simply forces the user shell to use a specific stylesheet to view the news listing, and this stylesheet in turn accesses the individual news items to construct a listing (but more on all this below, if you’re interested in writing or modifying stylesheets).

If you go back to the root directory and click on ’articles’, you find both another index.tkl file and a collection of subdirectories. Index.tk is the document you see when you click on "Articles" from the main menu of the -end-user interface of the portal. The sub-directories correspond to subdocuments which in the end-user interface will be displayed as a bullet list at the bottom of the page (once you have selected "Articles" from the menu. Each of these subdirectories contain their own index.tkl file which provides the content of this document, and the subdirectories may contain further subdirectories, and so on. By building a hierarchy of articles in this wway, you can construct an entire encyclopaedia or article collection for your portal. It is also possible to extend the portal by creating different article directories, which you can link to from the main menu if desired. Note the file directory.tkl (its structure described above) in the article directory. It is this file which tells the admin interface that this directory may only contain documents of the type document.xsd, and not, for instance, news documents. It would, however, be possible to allow more different document types to co-exist with the document.xsd type.

The help directory is structured in the same way as the article directory.

The link directory, on the other hand, has a different stucture (again, enforced by the directory.tkl file). In this directory, you will find a collection of subdirectories, corresponding to the top level of the subject hierarchy of the portal. Under these directories you may find further subdirectories, and, in certain cases, metadata about external resources. Every subject directory contains an index.tkl file of the type subject.xsd, which provides the name of the subject group. If resources have been cataloged under any given group, their metadata will be in the corresponding directory, in files of the type link.xsd. You can add new resources at any given level simply by creating new documents of the type 'link', and you can create new sub-catagories at any level simply by creating a new subdirectory. The admin interface will automatically ask you to fill in a subject document for the new subject group/sub-directory. You can freely pick the names of the subdirectories - they are not displayed to the user.

As an aside, it would be possible to mix the metadata records with, for example, local articles following the document.xsd schema. However, to handle these correctly would require a minor modification to the XSLT stylesheet which displays the subject groups (more on these stylesheets below).

3.6. Searching

The admin interface ensures that every time a document has been added or modified, it will be made searchable by looking for a directory.tkl file in the current directory or in any parent directory. If the file is to be made searchable, the indexing program is called to immediately add the the file to the index files of the search function. You can also, from the admin interface, perform a total re-indexing of the entire portal (for instance, if you have changed the directories which are to be searchable).

Chapter 4. Managing end-user interfaces

This section describes the Keystone approach to managing end-user interfacees for web portals. In general, the administration of user interfaces - unlike the administration of portal *content*, requires certain technical skills; in particular, a basic understanding of file management and editing under Unix (although this work *can* be done over a remote filesystem from another platform, if required), as well as an understanding of HTML, XML, and XSLT.

4.1. Document context and the user shell

In the Apache configuration file, requests for filenames with the suffix ".tkl" are set up to be handled by a script called shell.php, which is part of the Keystone distribution. This is the user shell for Keystone; the program which ensures that a given document is connected with the correct presentation format (in the form of an XSLT stylesheet).

The first thing the user shell does is to locate the root directory of the portal to which the givel .tkl file belongs. This is done by examining every directory, starting at the location of the file, and moving upwards, until a file named tkl.config is located. It is an error if this file doesn't exist under the document hierarchy of the current (logical) webserver. The portal root directory is the baseline for various references to file paths that can be made from inside stylesheets. Since the portal root is associated with the location of the file config.tkl, it is easy to run multiple portals under Keystone on a single webserver - each with their own portal root.

After this step, the user shell determines which XSLT stylesheet should be used to display the document to the user. To this end, the shell looks up the name of the root, or document tag in the XML document, for instance, "<metaData>". After this, the user shell searches in the current directory and and from there upwards towards the portal root for a .xsl (XSLT stylesheet) file with a matching name (note that this makes it possible to have multiple stylesheets for the same document type in a single portal, because the user shell always chooses the one closest to the document in the directory hierarchy.

The next step is to preprocess both the XML (Keystone) document and the stylesheet. In this stage, all XML elements which have an xml:lang attribute not matching the current language selection, are removed. The user selects his language by giving a 'lang' parameter with his HTTP request - generally based on a button or link from the interface - and his selection is stored in a session-persistent variable (based on a cookie). This technique makes it a simple matter to maintain multilingual data and user interfaces. The stylesheet editor simply has to remember to maintain multi-lingual variants of relevant parts of the stylesheet. If a single language-dependent part occurs in the middle of a large block of HTML, the element can be used with an xml:lang attribute to encapsulate any language-dependent parts.

The preprocessing stage also looks for declarations of session-persistent variables within the stylesheet (see below).

Now, the stylesheet is processed with the document as input, and the result is sent to the user [1]

Because the stylesheet may need to know something about the document's place in a greater context (unlike typical stylesheets which execute a context-neutral conversion of a document), the user shell makes a collection of parameters and services (functions) available to the stylesheet. These are user parameters that are provided by HTML forms or URL parameters that are made visible as parameters to the stylesheet, and functions encapsulated in private retrieval URL schemes, that can be accessed using XSLT's standard document() function, and which return different kinds of information.

4.2. Parameters

Simply by declaring an XSLT parameter using <xsl:param>, the programmer gains access to any user-supplied parameters in the HTTP request (note that the 'lang' parameter is always available).

In addition, there are certain system-specific parameters that may be used if needed, in particular:

- root, which gives the portal root directory relative to the web-servers root-directory. This parameter can be used to construct.
- portpath provides the absolute path to the portal root in the server file system. This parameter is rarely used.

4.3. Session parameters

Sometimes it is useful to associate parameters with the user session, to avoid having to carry an extensive list of parameters around through each link within the system. In the user shell, this can be done using a special tag, for example:

```
<xsl:param name="sort"/>

<portcom:session-var name="sort" default="title"

xmlns:portcom="http://www.indexdata.dk/keystone"/>
```

The <xsl:param> element introduces a parameter in the usual way. <portcom:session-var> tells the user shell that the given parameter should be stored and associated with the user session. The Default parameter is optional. The stored value is overwritten if the user (via GET/POST parameters) provides an explicit value for the parameter.

4.4. Supporting functions

The supporting functions are absolutely central to the function of the user shell, and it important to understand the possibilities that they make available, if you are to build a serious application using Keystone.

Since there is currently no good, standardised way to define and involke external functins from XSLT, we have chosen to implement our supporting functions as private URI schemes that can be used via the document() function, but also, for example, in the <xsl:import> and <xsl:include> elements.

In the following sections, we describe the individual support functions. All of the functions return their results in the form of an XML document which can treated in the usual way in XSLT.

Please not that when these functions are involked from XSLT, the ampersand (&), which is used to spearate parameters, must be escaped as &.

Note also that Keystone allows the portal to provide its own extension functions in either PHP or Perl, without requiring changes to the user shell. This is very useful in situations where a task is simply not well-suited for implementation in XSLT alone.

4.4.1. tkl-admin-header

Function: Generates relevant HTTP headers for the admin interfaces.

Synopsis: This handler is invoked automatically by the Keystone system.

Result: This handler is special in the sense that it is invoked automatically by the shell.php in case the admin interface is about to be loaded. If not overwritten, this handler generates the relevant HTTP headers for authentication.

Warning: In general, this handler should not be overwritten except when you know exactly what you're doing!

See also *tkl-header*.

4.4.2. tkl-args

Function: Query HTTP GET'd/POST'd variables.

Synopsis: `tkl-args:`

Result: XML encoding representation of a list of key => value pairs, for instance,

```
<args>
  <var1>value1</var1>
  <var2>value2</var2>
</args>
```

If an array variable, i.e.

```
Array
(
    [key1] => value1
    [key2] => value2
)
```

is present, it will be encoded this way:

```
<args>
  <list i="key1">value1</list>
  <list i="key2">value2</list>
</args>
```

4.4.3. tkl-default

Function: Looks for the presence of a file in portal area and Keystone core area.

Synopsis: `tkl-default:?path=my_dir/my_file`

Result: Path is the name of the file relative to the portal root. The handler looks for this file in the portal area. If it exists, the absolute path is returned. Otherwise, the file is looked for in the Keystone core area. If it exists here, the absolute path is returned, otherwise an error flag is raised. Example:

```
<tkl-default>
  <abs_path>/var/www/bibliotheca/my_dir/my_file</abs_path>
</tkl-default>
```

on success. On error, the returned XML looks like this:

```
<tkl-default>
  <error>Error message</error>
```

```
</tkl-default>
```

4.4.4. tkl-file

Function: Opens a Keystone file and filters language dependent elements.

Synopsis: `tkl-file://subdir/my_file.tkl`

Result: This handler returns an XML document tree ready to process in XSLT. It is important not to confuse the "tkl-file" handler with the generic XSLT handler "file". The differences can be summarized this way:

- File name paths are resolved relative to the Keystone portal root rather than relative to the root of the filesystem. Thus, tkl-file://my_file.tkl will look in the portal root for my_file.tkl.
- Language dependent elements are subjected to filtering. A specific language is associated with an XML element with the xml:lang="xx" attribute, e.g.

  ```
  <tag xml:lang="en">Here goes some CDATA in english</tag>
  <tag xml:lang="oz">Here goes the CDATA in some other language</tag>
  ```

  If the chosen interface language is set to "oz", then elements with, for instance xml:lang="en" will be suppressed.

The tkl-file handler is well-suited for importing general XML into your stylesheet, Keystone files as well as XSLT stylesheets, or any other type of XML. The tkl-file handler works excellent in combination with the XSLT import and include instructions, i.e.

```
<xsl:import href="tkl-file://my_stylesheet.xsl"/>
<xsl:include href="tkl-file://subdir/some_other_stylesheet.xsl"/>
```

which are the preferred XSL stylesheet inclusion idioms.

4.4.5. tkl-file-exists

Function: Checks if a specified file exists.

Synopsis: `tkl-file-exists:?file=path/to/file.tkl`

Result: Returns the following XML structure:

```
<tkl-file-exists>
  <exists>1</exists>
</tkl-file-exists>
```

if the file exists, and of course otherwise,

```
<tkl-file-exists>
  <exists>0</exists>
</tkl-file-exists>
```

4.4.6. tkl-find

Function: Finds files which match a given pattern or mask.

Synopsis: `tkl-find://?path=pathmask&mask=filemask&select=fieldmask&level=number`

Result: A list oif <file> elements, one for each matching file, containing the elements of each file given by fieldmask.

Example: tkl-find:/?path=*/index.tkl&select=title

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<tkl-find>
<file path="./news/index.tkl">
<title>Nu med links!</title>
</file>
<file path="./oldcheese/index.tkl">
<title>Gammel ost på nye flasker</title>
</file>
.....
</tkl-find>
```

The tkl-find function can be used in two different ways. IN the simple version, it searches for files which match path-parameters (comparable to a simple listing of files matching a filename (glob) pattern in a Unix shell. It returns a number of file elements, containing the elements selected by the fieldmask parameter.

The fieldmask parameter has the form element|element|... - in other words, an arbitrary number of element names separated by vertical bars.

The function can also be used like the find-command of Unix, to recursively traverse one or more subtrees.This example from the base test portal finds all index.tkl files in subdirectories of links/* two levels down.

tkl-find:/?path=links/*&mask=index.tkl&select=title&level=2

Returns:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<tkl-find>
 <dir path="./links/27/" level="1" att="2">
 <file path="./links/27/index.tkl">
 <title xml:lang="en">General Subjects</title>
 </file>
 <dir path="./links/27/01/" level="2" att="2">
 <file path="./links/27/01/index.tkl">
 <title xml:lang="en">Cross-disciplinary subjects</title>
 </file>
 </dir>
 <dir path="./links/27/03/" level="2" att="2">
 <file path="./links/27/03/index.tkl">
 <title xml:lang="en">Library collections</title>
 </file>
 </dir>
 </dir>
 ................
</tkl-find>
```

(please note that the structure above represents a hierarchy, where directory elements can contain both files and onther directories).

If the path expression only matches files, there is no needs to provide a mask. If the path expression matches directories, the function will search these subdirectories, looking for files matching the mask parameter, until level levels (steps)

The <dir> and <file> elements both have a path-attribute, which provides a relative path to the file or directory (with respect to the location of the current document).Usually, this means that these paths can be used without modification in, say, a HTML <A> element, as a relative URL.

However, please note that if the original path expression is absolute (ie. starts with a '/'), it is processed relative to the root of the *portal*, and the path-attribute in the <dir> and <file> elements can be used directly as a server-absolute URL. For example:

```
tkl-find://?path=/news/news*.tkl&select=date|title
<tkl-find>
 <file path="/tkl/news/news1.tkl">
 <date>2002-07-08</date>
 <title xml:lang="en">The test has begun</title>
 </file>
 <file path="/tkl/news/news429.tkl">
 <date>2002-07-10</date>
 <title xml:lang="en">The news entries have been cleaned out</title>
 </file>
</tkl-find>
```

Please note that tkl-find, like the other functions, pre-process their results so that any elements marked with xml:lang attributes not matching the current language are filtered out. Under normal circumstances, the programmer should not have to worry about resource languages in his stylesheets.

4.4.7. tkl-grant

Function: Checks for user read/write/create authorization. This handler only makes sense in an administration interface, i.e. where `?admin=1`.

Synopsis: `tkl-grant://?type=create&file=tkl_cwd/new_file.tkl`

Result: In case, the user is authorized to perform the given action, the following XML structure is returned:

```
<tkl-grant>
  <granted>1</granted>
</tkl-grant>
```

Otherwise, false is returned, i.e.

```
<tkl-grant>
  <granted>0</granted>
</tkl-grant>
```

The parameters `file` and `type` are both mandatory. The `file` argument specifies at which object the authorization request is targeted, while `type` specifies the type of action. The `type` argument must be one of the following:

- read - Read permission is requested and the `file` argument must contain the the path to a file.
- write - Request write permission, `file` can either be a file or a directory.

- create - Ask for permission to create some kind of object, `file` must be a directory.

All paths are relative to the Keystone portal root.

4.4.8. tkl-header

Function: Send HTTP headers to the user interface.

Synopsis: This handler is called automatically by the Keystone system. Don't attempt to call it from your XSL stylesheets.

Result: This Keystone handler is intentionally born empty. When developing a Keystone portal, the programmer is encouraged to overwrite the tkl-header by something meaningful. Typically, one wants to exploit the fact that this handler is called before any output has been generated by the XSLT processor. Thus, for instance HTTP headers can be created here.

See also *tkl-admin-header*.

4.4.9. tkl-help

Function: Find the closest occurence of help.tkl.

Synopsis: `tkl-help:`

Result: This support function is in a state of development and in the future it may be out-phased in favour of a more generalized implementation. But for the time being, it looks in the current working directory for the presence of a help.tkl file. If there is no such file there, it goes one directory level back and looks there. It keeps doing this process, until it finds a help.tkl file or reaches the Keystone portal root.

If a help.tkl file was found, the following XML document is returned:

```
<tkl-help>
  <file>../../help.tkl</file>
</tkl-help>
```

where the content of the `<file>` tag is the relative path from the current working directory to the location of the help.tkl file.

Otherwise, the `<file>` tag is replaced by some error message.

4.4.10. tkl-mail

Function: Support function for sending e-mails.

Synopsis: `tkl-mail:?to=recipient@domain.org&subject=Subject text&message=Message body.`

Result: The following mail is sent with the native system's mail transmission agent:

```
To: recipient@domain.org
Subject: Subject text

Message body.
```

If the mail was sent successfully, the following XML is returned:

```
<tkl-mail>
  <status>OK</status>
</tkl-mail>
```

Otherwise, an error flag is raised:

```
<tkl-mail>
  <error>error message</error>
</tkl-mail>
```

4.4.11. tkl-path

Function: Returns a path of "breadcrumbs" to the root of the portal.

Synopsis: `tkl-path:/?select=title`

Example:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<tkl-path>
<step path='./../../../'>
<title xml:lang="en">Bibliotheca</title>
</step>
<step path='./../'>
<title xml:lang="en">Social Sciences</title>
</step>
```

```
<step path='./'>
<title xml:lang="en">Law</title>
</step>
</tkl-path>
```

Tkl-path returns an XML representation of the path from a given document to the root of the portal. In the base portal Bibliotheca, in the file interface.xsl, in the template insert-path, is an example of the usage of this function. In the base portal, the function is used to create a clickable path on each side, as a navigational aid.

The function operate by examining all directories from the current directory up to the root. For all directories which contain an index.tkl file, it returns a <step> element with a path-attribute providing a relative path. The path is generally suitable for use as a relative URL in an HTML <A> tag.

4.4.12. tkl-search

Function: Searches a Z39.50 database

Synopsis: `tkl-search://unix:/home/indexdata/html/tkl/db/socket?query=@attrset idxpath computer&start=1&syntax=xml&number=6`

Example:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<search>
<start>1</start>
<number>6</number>
<server url="unix:/home/indexdata/html/tkl/db/socket" status="1">
<hits>6</hits>
<end>6</end>
<record offset="1">
<subject xmlns:idzebra="http://www.indexdata.dk/zebra/">
<title xml:lang="en">Computer science</title>
<idzebra:size>153</idzebra:size>
<idzebra:localnumber>332</idzebra:localnumber>
<idzebra:filename>links/30/05/index.tkl</idzebra:filename>
</subject>
</record>
...............
<record offset="6">
.......
</record>
</server>
</search>
```

Tkl-search executes a search against the given Z39.50 server. The address of the server generally follows the Z39.50 URL format, even though the example aboce is not standard, but an INdex Data specific extension which allows the use of Unix filesystem sockets (Unix fomain sockets) instead of internet host addresses/port numbers. In the local search function of the base portal Bibliotheca, Unix domain sockets are used to avoid having to allocate a new TCP/IP port to every portal running on the same machine.

The parameters start, number, and syntax work as you would expect, and queries are given in the PQF format (described at http://www.indexdata.dk/yaz/doc/tools.php#AEN2265) or ISO CCL (XX add documentation for the configuration of CCL field mapping setup).

The results come back as shown above. In particular, the users start and number-parameters are repeated in the XML structure. In the <server> element (which may become repeatable if multi-target searching is introduced) is the number of hits, along with the highest record number returned. After this follows a number of tecord elements, each of which contains one retrieval record from the server.

In portals like the base portal Bibliotheca, the tkl-search function is used to search the portal's index, which are hosted by a Zebra server. Note that Zebra for each record returns the path of the record in the element <idzebra:filename>.

4.4.13. tkl-set-footer

Function: Name a custom handler to be executed after XSLT processing and output has completed.

Synopsis: `tkl-set-footer:handler-name:?parameters`

Result: Returns an empty XML document. Has the side effect of executing the named custom handler AFTER the current stylesheet has completed. The most obvious use for this is to execute a PHP or Perl script after the main page HTML has been output. This script in turn may perform a time-consuming process, i.e. a search, and provide a dynamic update of the user's screen by emitting a series of Javascript commands which alter the document HTML contents.

4.4.14. tkl-soap

Function: Provides access to procedure calls on remote systems via the SOAP protocol.

Synopsis: `tkl-soap:/MyTestService.wsdl?tkl:fun=weirdFunction&Hello` World

Example:

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2001/12/soap-envelope" >
```

```
<env:Header>
<t:transaction
xmlns:t="http://thirdparty.example.org/transaction"
env:encodingStyle="http://example.com/encoding"
env:mustUnderstand="true" >
5
</t:transaction>
</env:Header>
<env:Body>
<m:reserveAndChargeResponse
env:encodingStyle="http://www.w3.org/2001/12/soap-encoding"
xmlns:rpc="http://www.w3.org/2001/12/soap-rpc"
xmlns:m="http://travelcompany.example.org/">
<rpc:result>m:status</rpc:result>
<m:status>confirmed</m:status>
<m:reference>FT35ZBQ</m:reference>
<m:viewAt>
http://travelcompany.example.org/reservations?code=FT35ZBQ
</m:viewAt>
</m:reserveAndChargeResponse>
</env:Body>
</env:Envelope>
```

Tkl-soap provides access to any SOAP-based, so-called web-service (or remote API) anywhere on the Internet. The parameters to the the function consist of a reference to a service definition file (WSDL), a function name, and a list of parameters. The parameters can be structured in a variety of ways, to enable the use of different types of remote function.

The WSDL file can be identified using a relative or portal-absolute path, or as an HTTP URL.

Different SOAP functions pose different requirements to the structure of their parameters. In the example here, concat() is used to construct an argument list soleley in order to increase readability. You can imagine that "$query" in the example is a parameter which has been supplied by the user (see above about accessing user-supplied parameters in XSLT stylesheets).

INSERT EXAMPLE FROM bibliotheca/soap/google/google.xsl

Parameters for the SOAP functions can also be named explicitly, for example:

tkl-soap:MyTestService.wsdl?tkl:fun=myFunction&alpha=1&beta=2

Some functions require structured data, for instance, the arguments

primitive=test&alpha.a=bob&alpha.b=bub

yields the structure:

primitive=>test, alpha=>{a=>bob, b=>bub}

Substructures can be anonymous, as in

.a=bob&.b=bub

which yields

{a=bob, b=>bub}

As a concrete example, Amazon.com can be searched like this:

```
<xsl:variable name="result" select="document(concat(
'tkl-soap:AmazonWebServices.wsdl?',
'tkl:fun=KeywordSearchRequest',
'&amp;.keyword=', $squery,
'&amp;.page=2',
'&amp;.mode=books',
'&amp;.tag=webservices-20',
'&amp;.type=lite',
'&amp;.devtag=', token,
'&amp;.format=xml',
'&amp;.version=1.0'
))"/>
```

If debugging has been enabled (adding the parameter debug=1 to the URL line), both the SOAP request and response packages will be displayed so it is easy to determine if you have constructed the right parameters for a given web service.

4.4.15. tkl-sql

Function: Connect to a MySQL database server and execute SQL instructions.

Synopsis I: `tkl-sql:?type=connect&host=mysqlhost.mydomain.org&db=sqlbase`

Synopsis II: `tkl-sql:?type=exec&sql=SELECT * FROM xxxx WHERE yyyy=zzzz`

Result: This Keystone handler sends an SQL query to the database specified and encode the returned records as XML.

Before any SQL instruction can be carried out, a database connection must be established. This is performed by calling

```
tkl-sql:?type=connect&host=mysqlhost.mydomain.org&db=sqlbase
```

If you need to login as a specific user with password, simply add a `&db_user=mysql_user&db_pwd=secret` to the URI. A database connection has a global scope in the sense that you can only have one open connection at a time. If you attempt to open 2 connections, the last connection will overwrite the first. A connection request with `type=connect` does not return any meaningful XML. In the future, some kind of reporting should be returned, i.e. success/failure/diagnostics.

Assuming one have an open MySQL connection, any number of manipulations can be made during one HTTP session. You execute SQL instructions the following way:

```
tkl-sql:?type=exec&sql=SELECT this FROM that WHERE xxxx
```

When `type=exec`, the returned XML should be interpreted as the matching records. The XML structure is the following:

```
<sql hits="xx" affected="yy">
  <record>
    <field1>value1</field1>
    <field2>value2</field2>
      ...
...
  </record>
  <record>
    <field1>value1</field1>
    <field2>value2</field2>
      ...
...
  </record>
    ...
    ...
</sql>
```

The attribute `hits` represents the number of matching records, which is only meaningful for `SELECT` SQL instructions. For `INSERT` and `UPDATE` instructions, the `affected` attribute will be set to the number of rows affected by the operation.

The XML encoding is crucial when non-XML data from an external database is converted into well-formed XML. Therefore, one needs to decide how to handle two aspects of this situation:

- Character encoding, ISO-8859-1 or UTF-8. - If the data is stored as ISO-8859-1 (also known as latin-1), then we need to transform the data into the UTF-8 character set. In this case, you will need to specify the `encoding=ISO-8859-1` URI argument when calling the tkl-sql handler. If the data is already stored as UTF-8, you don't need to specify the `encoding` argument.
- Encoding of special characters, i.e. <, >, ", ', & - If such special characters are already escaped in the raw data, please specify the `xml=1` arguments, telling the tkl-sql handler not to attempt to encode anything with entities. When this setting is not specified, all 5 characters will be replaced by their corresponding named XML entities.

4.4.16. tkl-time

Function: Return the current time.

Synopsis: `tkl-time:?format=xxx`

Result: Returns the time as a raw string. The optional setting `format` specifies how the time should be formatted. Use the conversion specifiers given at the PHP manual pages (http://www.php.net/manual/en/function.strftime.php).

4.4.17. tkl-unique

Function: Filenaming support function for the admin interface.

Synopsis: `tkl-unique:?format=xxxx&type=yyyy`

Result: This support function is designed mostly for the admin interface. It returns information about how to name new files respecting the local schemes specified in directory.tkl. The `type` argument can be one of the settings allowed in the naming scheme field in directory.tkl: globallyUnique, fixedName and userProvided. The `format` setting is the format used when naming files according to the globallyUnique scheme. Use the reserved character % to mark where to put the unique file identifier, e.g. `format=link-%.tkl`.

The returned XML document has this form:

```
<tkl-unique>
  <unique>link-12345678.tkl</unique>
  <exists>0</exists>
  <file_spec>0</file_spec>
</tkl-unique>
```

When `type=globallyUnique`, the `<unique>` tag contains the unique filename based on the specified format and on some dynamically constructed unique ID. For `type=fixed`, the `<exists>` tag will be set to 1, if the fixed file already exists, otherwise it will be set to 0. Finally, if `type=userProvided`, the `<file_spec>` tag will be set to 1.

Warning: Don't overwrite this function unless you know exactly what you are doing!

4.4.18. tkl-user

Function: Get information about user currently logged into the admin interface.

Synopsis: `tkl-user:`

Result: Returns an XML document with information about the user currently logged into the admin interface. The structure of the XML is the following:

```
<tkl-user>
  <login>user_login</login>
  <password>ff402ace53f370435</password>
  <name>Human Readable Name Of This User;/name>
  <email>user_login@domain.org</email>
  <usertype>administrator</usertype>
  <area>/</area>
  <area>/news</area>
  <pass_type>md5</pass_type>
</tkl-user>
```

The `<login>` tag contains the user's unique ID used when logging in. The `<password>` is a representation of the password used when logging in. Since Keystone version 1.4.1 new passwords will automatically be stored as MD-5 checksums. To ensure backwards compatibility, the tag `<pass_type>` is introduced to provide information about the password type used. Currently, this can only be one of the types `md5` or `clear` or empty/non-existent, the two latter having the same meaning. The `<name>` and `<email>` are self-explaining. The `<usertype>` can be one of the following:

- `administrator` - The user logged in belongs to the super-user class and is allowed to do anything within the web-admin-interface.
- `user` - Normal user privileges, i.e. have read privileges anywhere and write privileges in home areas, see below.
- `spectator` - This is a read-only user.

Finally, we have the repeatable `<area>` tag. This is a list of home directories, where the user has write permissions in addition to the granted read permissions given by default.

Since this is an admin interface handler, please don't attempt to overwrite it, unless you know what you are doing!

4.5. Portal configuration

Besides defining the Keystone portal root, the file `tkl.config` plays the role of the master portal configuration file.

Using `tkl.config` as a configuration file is not required. If there is no XML header in the beginning of `tkl.config`, it will not be treated as a configuration file, and the default settings will be used instead.

If, on the other hand, there is an XML header in the beginning of `tkl.config`, the configuration file parser expects an XML structure like this:

```
<?xml version="1.0"?>
<config>

  <!-- Simple setting -->
  <setting name="setting_name1" value="setting_value1" />

  <!-- Complex setting -->
  <setting name="setting_name2">
    <setting name="sub_setting_a" value="value A" />
    <setting name="sub_setting_b" value="value B" />
  </setting>

  <!-- Another simple setting -->
  <setting name="setting_name2" value="setting_value2" />

</config>
```

Let's run through the configuration settings currently supported by the system:

- `base_url` - The public base URL of the Keystone portal.
- `from` - The "from" email address used by various robots attempting to send email to users.
- `userlang` - Default user interface language. This language will be used, if (a) no explicit language is specified as a URL parameter `lang=xx`, and (b) no language is stored in the session variables.
- `cache_dir` - Use this directory (under the portal root directory) for the caching. If this directory exists, caching will take place. NB: This feature is only experimental!
- `cache_expires` - Number of seconds after which a cached version of a portal web page expires.
- `allow_symlinks` - This setting can be set to "yes" or "no" depending on whether you want to allow users to make symlinks between tkl files. Default is "yes".

- `languages` - List of languages in which you want to keep language dependent fields. You must specify your language list using this pattern:

  ```
  <setting name="languages">
    <lang name="da" value="Danish"/>
    <lang name="en" value="English"/>
    <lang name="xx" value="My language"/>
  </setting>
  ```

- `xform_xsl` - Here you can specify a stylesheet - as usual relative to the Keystone portal root - which over-writes the built-in Keystone xml editor display stylesheet (`xform2.xsl`). Normally, you would like to xsl:import the built-in version of the stylesheet and only overwrite on a template basis. This should be done in your stylesheet using this instruction:

  ```
  <xsl:import href="xform-xsl:"/>
  ```

Some of the settings can be overwritten by dedicated config files, for instance, `urlcheck.tkl` which is the config file for the Keystone urlchecker. Confirm man-pages for the Keystone run-time environment.

4.6. Debugging

By adding the HTTP parameter debug=1 to any given page in a portal, a diagnostic output is produced. The pre-processed document and stylesheet are displayed, as are any documents loaded using Keystone-specific file schemes. The debug option is a very useful tool in tracking down problems with a portal, or simply for finding the best way to handle complex output for one of the support functions.

4.7. The base portal Bibliotheca

This section describes the structure of the base portal Bibliotheca, which is included with Keystone. Go through this section if you're contemplating substantial changes to the stylesheets of that portal, or if you're looking for trips about good portal design in Keystone.

4.7.1. User interface

The user interface of the base portal is implemented as standard XHTML with a minimum use of graphical elements, and conventional use of CSS stylesheets. This means that elements of the portals appearance can be fine-tuned simply by modifying the accompanying CSS stylesheet. More substantial changes can be carried out by altering the supplied .xsl files - but this requires at least a passing knowledge of HTML and possibly XHTML (although much of the included XSLT_code is fairly self-explanatory). In addition, it helps to have some knowledge about the extension functions made available to XSLT by Keystone.

4.7.2. Overall page structure - interface.xsl

This file, which is located in the root directory of the portal (remember that the root directory is the directory containing the file tkl.config), defined the overall framework of the portal interface. Interface.xsl doesn't correspond to any specific document type, but it is included by most of the other stylesheets. If you look in the file, the bulk of its content is a template called "main-page". This template produces the overall HTML (XHTML) structure of each page. Inside the HTML code are calls to different templates which in turn produce the real content of each page. These content-producing templates are provided by the stylesheet which includes interface.xsl, in an interaction which can be compared to "callback functions" in several other programming languages.

Interface.xsl provides default versions of the callback templates, mostly to remind the programmer to replace them with something else. Depending on your temperament, you can compare the approach around interface.xsl with object-oriented programming where each document type inherits a basic layout from interface.xsl, overriding specific details of the interface; or as a traditional, callback-based paradigm.

In addition the main-html, interface.xsl also defines the templates insert-path, which produces a graphical bread-crumb path to the root of the portal, and menu, which constructs the left-hand side menu, based on data stored in the file index.tkl in the portal root directory [2]

4.7.3. portal.xsd/xsl

Typically, there is only going to be one document of the type 'portal' in any given portal, and that's the file index.tkl in the portal root directory, which defines some overall structural information for the portal, and the corresponding portal.xsl, which presents the front page of the portal. But portal.xsl is also a rather typical example of a sylesheet for a document type, and so it's worth looking at it in slightly greater detail.

The processing commences by a template which matches the document element - in this case the xpath "/portal". The only thing this template does - and this is typical - is to call the template main-page, which is defined by interface.xsl. It is the template main-page which subsequently calls the other templates in the file - specifically main-news-content and main-body.

Main-news-content doesn't do much other than calling an external utility (from news.xsl) to display the latest news. All document pages in this portal use the right-hand column for news, but they don't have to - some designs might use the right-hand column for context-specific information.

The template main-body does the real work - it defines what content will appear in the main window, in the centre of the display. In this case, it uses the document() function and the special Keystone extension "tkl-find" to find information about all subject groups under the directory "links", two levels down. The code underneath - the two nested for-each loops - are responsible for showing the headlines to the user, with links to the relevant subject groups.

Different types of subject hierarchies might benefit from different presentation styles. If you want to use a radically different subject hierarchy from the simple one shown here, you may want to use a different approach from the one shown here to display the front page of your portal.

4.7.4. subject.xsd/xsl

The display format subject.xsl is asociated with files of the type subject. These files are placed - generally with the name index.tkl - in each directory under the "link" directory, and they represent metadata about the individual subject groups. The file subject.xsl follows the same internal structure as portal.xsl - a template matches the document element -- <subject>. From here, main-page is called, which in turn renders the page with the aid of the callback functions.

The template main-body performs two tasks. First, it examines whether there are any sub-categories to the current subject group. This is done simply by looking for any sub-directories containing an index.tkl file. Next, the template looks for resources cataloged in the current group. This is done simply by searching for files in the current directory that match the pattern "link*.tkl".

Finally, the lists of sub-categories and cataloged files are displayed to the user.

4.7.5. link.xsd

The link type is noticeable in that it does not have its own stylesheet associated in this portal. This is because the portal end-user never actually views a link record on its own - they are displayed either by subject.xsl - when the user browses - or by search.xsl (see below) when the user executes a search.

4.7.6. document.xsd/xsl

The document type is designed as a general-purpose workhorse - suitable to maintain any type of s simple text document or hierarchy of documents.

If you look in document.xsl, you'll notice the the stylesheet both displays the content of the current document, and searches for any subdirectories containing an index.tkl file. If any are found, a list of links is shown in the bottom of the displayed document.

In the base portal, document.xsd/xsl are quite simple - they consist only of a title, a creator, an abstract, and a body text (which is assumed to contain either plain text or an XHTML fragment). There are rich possibilities for extending or adapting this document type for specific applications - such as displaying illustrations, chapters, related links (eg. in the right-hand column), etc.

4.7.7. search.xsl

Search.xsl executes a search when the user hits the "search" button in the top menu of the portal (and hence submits the associated form). The stylesheet follows the usual structure, and the template main-body does the real work.

In the top of the template, three parameters are declared: portpath, query, and start. Portpath is a "built-in" parameter which is made available by the user shell. It contains the aboslute path to the root of the portal - it is used get hold of the communications channel for the search server. The parameters query and start come from the user's HTTP request (ie. from the search form).

If a query has been supplied, the real work begins. The search is executed using the function tkl-search, which is made available by the user shell. The search is directed against a Z39.50 server with a Unix domain address called db/socket under the portal root directory. The parameters directing the parsing of the user's query are taken from the index.tkl file.

If the search was successful, the number of hits is displayed, then the template "previous-next" (defined further down in the stylesheet is called to produce links to navigate the result set (if necessary).

Finally, the records are displayed, in a for-each loop. Here, we check whether the documents are link records with metadata for external documents, or whether they represent internal content. The link (internal or external) is placed in the variable $url. Hits corresponding to entire subject groups are also shown, but differentiated with a different graphical symbol. Finally, the title of the resource is shown, as a hyperlink to the resource, followed by a description, if available. The XSLT fragment which displays each record should be easily extensible if there is a requirement to display more information about each record.

4.7.8. news.xsd - newspage.xsl

The news document type is not shown by its own stylesheet. Instead, news documents are displayed either by the template read-news in news.xsl, which produces the summary right-hand news column, or by the stylesheet newspage.xsl, which is located in the news directory. Newspage.xsl produces the page which is displayed when you ask to see a detailed news listing or a single news item.

Notes

1. In most cases, the XSLT stylesheet is expected to produce HTML - or rather XHTML. However, it is easy enough to imagine a portal, or a part of a portal, which outputs structured XML - for instance to support a web services-type interface,
2. If a more complex menu structure is required, it is fairly trivial to extend this template (and the corresponding data schema in the file portal.xsd).

Chapter 5. Keystone and OAI Metadata Services

Keystone has two distinct interfaces to OAI metadata: it can be used as a OAI repository, thus serving OAI metadata upon valid requests, and it has facilities to harvest OAI metadata from other OAI servers, which then can be shown as usual Keystone record posts using any browser.

For general information on the OAI standard we refer to http://www.openarchives.org/ (http://www.openarchives.org/). A nice tutuorial can be found at http://www.oaforum.org/tutorial/ (http://www.oaforum.org/tutorial/).

5.1. Keystone as an OAI repository

Keystone has been designed to share its information contents using the Open Archives Initiative (OAI) protocol for metadata harvesting (PMH). The current version supports enough functionality to act as a basic OAI data provider, but the level of protocol support is intended to develop over time as requirements warrant. At present, the following OAI-PMH 'verbs' are supported:

- Identify
- ListMetdataFormats
- ListSets
- ListIdentifiers
- GetRecord
- ListRecords

OAI-PMH supports different data exchange formats, and requires support for a simple Dublin Core-based format called "oai_dc".

Keystone allows a portal to share any type of documents - not just metadata documents. In general, a document is offered for exchange if it is located in a directory marked for exchange with the oai-exchange flag (in the directory.tkl file), and if a suitable conversion filter can be located. The server looks for the filter in the directory schemas under the portal root, and the filename convention is `schemaname2metadataprefix.xsl` where the schema name is the name of the root element of the given document, and the metadata prefix is simply the metadata prefix requested by the OAI client (or 'service provider', in OAI parlance).

As an example, the base portal Bibliotheca, see Section 4.7, contains a file named `bibliotheca/schemas/link2oai_dc.xsl` which defines the transformation from a link-type

document to the oai_dc format.

The base URL for the OAI server associated with a Keystone portal is simply the URL for the main (front) page of the portal. If, for instance, a portal has been installed under `http://www.indexdata.dk/tkl/` the following request will provide description of the portal according to the OAI-PMH:

```
http://www.indexdata.dk/tkl/?verb=Identify
```

Since version 1.4.4, the Keystone OAI service supports selective harvesting enabling the client to address various sub-sets of the full record collection. Currently, the following OAI harvesting criteria are supported:

- Data/time based harvesting. Use the `from` and `until` parameters to specify bounded or unbouned time intervals.
- Set based harvesting (since version 1.4.5). Use the `set` parameter to specify a set. A record is associated with its set using its document type (XML schema name). Thus, for instance a link record with document type `link` and schema `link.xsd` belongs to the `set=link`. Use the OAI `verb=ListSets` call to find out which sets are supported by the OAI service.

Read more about the OAI protocol at http://www.openarchives.org/.

5.2. Bibliotheca example OAI repository

Assuming that the Bibliotheca portal has been installed under `http://localhost/bibliotheca/` the most important OAI-PMH commands are supported as follows (from version 1.4.5):

5.2.1. Identify

Function: Provide basic server identification and server capabilities. This is always the initial request.

Synopsis: `http://localhost/bibliotheca/?verb=Identify`

5.2.2. ListMetdataFormats

Function: Get information on which metadata formats are supported. This is used to choose the desired metadata format for the later fetched records.

Synopsis: `http://localhost/bibliotheca/?verb=ListMetadataFormats`

```
http://localhost/bibliotheca/?verb=ListMetadataFormats
http://localhost/bibliotheca/?verb=ListIdentifiers&metadataPrefix=oai_dc
http://localhost/bibliotheca/?verb=ListIdentifiers&metadataPrefix=oai_def_portal
```

5.2.3. ListSets

Function: Get information on the names of the possible set to use in an request.

Synopsis: `http://localhost/bibliotheca/?verb=ListSets`

Examples:

```
http://localhost/bibliotheca/?verb=ListSets
http://localhost/bibliotheca/?verb=ListRecords&metadataPrefix=oai_dc&set=subject
http://localhost/bibliotheca/?verb=ListRecords&metadataPrefix=oai_dc&set=link
```

5.2.4. ListIdentifiers

Function: Provide a list of record identifiers which can be used in the next query to determine which records to fetch.

Synopsis:
`http://localhost/bibliotheca/?verb=ListIdentifiers&metadataPrefix=oai_dc`

Examples:

```
http://localhost/bibliotheca/?verb=ListIdentifiers&metadataPrefix=oai_dc
http://localhost/bibliotheca/?verb=ListIdentifiers&metadataPrefix=oai_def_portal
http://localhost/bibliotheca/?verb=ListIdentifiers&metadataPrefix=oai_dc&set=subject
http://localhost/bibliotheca/?verb=ListIdentifiers&metadataPrefix=oai_dc&
     from=1978-01-01T12:00:00Z&until=2013-03-20T20:30:00Z
http://localhost/bibliotheca/?verb=GetRecord&metadataPrefix=oai_dc&
     identifier=oai:Unknown.tkl.indexdata.com:links/27/01/index.tkl
```

5.2.5. GetRecord

Function: Obtain one specific OAI record.

Synopsis: `http://localhost/bibliotheca/?verb=GetRecord&metadataPrefix=oai_dc&identifier=oai:Unknown.tkl.indexdata.com:links/27/01/index.tkl`

Examples:

```
http://localhost/bibliotheca/?verb=ListSets
http://localhost/bibliotheca/?verb=ListIdentifiers&metadataPrefix=oai_dc&set=link
http://localhost/bibliotheca/?verb=GetRecord&metadataPrefix=oai_dc&
     identifier=oai:Unknown.tkl.indexdata.com:links/27/01/index.tkl
```

5.2.6. ListRecords

Function: Obtain a list of OAI records. A time period or a record set can be specified, as well as another metadata prefix.

Synopsis:
`http://localhost/bibliotheca/?verb=ListRecords&metadataPrefix=oai_dc`

Examples:

```
http://localhost/bibliotheca/?verb=ListRecords&amp;metadataPrefix=oai_dc
http://localhost/bibliotheca/?verb=ListRecords&metadataPrefix=oai_def_portal
http://localhost/bibliotheca/?verb=ListRecords&metadataPrefix=oai_dc&set=link
http://localhost/bibliotheca/?verb=ListRecords&metadataPrefix=oai_dc&
     from=1978-01-01T12:00:00Z&until=2013-03-20T20:30:00Z
```

5.3. OAI harvesting from within Keystone

In addition, Keystone allows the system administrator to define OAI harvesting tasks. To do so, one must install the `libtkl-perl` and the `tkl-oai-harvester` Debian packages.

The OAI harvester daemon called `tkl-oai-harvester` is started and stopped with the scripts

```
/etc/init.d/tkl-oai-harvester start
/etc/init.d/tkl-oai-harvester stop
/etc/init.d/tkl-oai-harvester restart
```

When installed with `apt-get` as `.deb` pakages on a Debian system, these start/stop scripts are installed properly, so that the harvesting service is started automatically at boot time. The start/stop scripts are rather simple and it should not be difficult to adjust them to work with other operating systems than Debian GNU/Linux.

Harvesting tasks are created in the admin interface. The `bibliotheca` example portal contains the task directory called `bibliotheca/tasks`, including two subdirectories `oaibizigate`, `oaitklite`, and two Keystone files `directory.tkl`, and `index.tkl`, which are tuned to display the resulting `oai*.tkl` files containing the harvested OAI metadata records.

Navigate within the admin interface to the `bibliotheca/tasks` directory, and add a new oai task file. Fill in the starting url (remember the trailing slash when addressing a Keystone OAI server!). Type the target directory relative to the portal root - for example "/tasks/oaitklite/" - and choose select status = pending. After saving the resulting task file should look like this:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<task creator="admin" created="2003-07-10, 13:59:50" modifier="admin" modified="2003-0
  <tasktype>oai</tasktype>
  <url>http://tkl-cvs.indexdata.dk/bibliotheca/</url>
  <target>/tasks/oaitklite/</target>
  <description>OAI harvesting job at our Keystone server</description>
  <status>pending</status>
  <xslt>oai2link.xsl</xslt>
  <handler></handler>
</task>
```

The content of the `<handler>` tag is interpreted as an optional script called by the harvester, when the job is finished. There is no restrictions upon the script except that it is run by the web server user, typically `www-data`, with the corresponding restrictions. The collection of task handler scripts should be placed in the reserved directory `tasks/handlers`.

Each such task handler script is called with a single argument, the path to the directory, where the harvested records are placed.

Please notice, that when you associate a task handler script other than the trivial one, i.e. `do_nothing.handler`, to your OAI harvesting task, this handler is given the responsibility for indexing the harvested records. Otherwise, the OAI harvester daemon `tkl-oai-harvester` performs the indexing.

The `<xslt>` tag contains the name of an XSLT transformating stylesheet which will be applied to each OAI harvested record before it is stored. The collection of such XSLT transforming stylesheets should be placed in the reserved directory `/authorities/oai`.

If the optional `<prefix>` tag is specified, this will be used as the filename prefix when the OAI harvested records are stored as files on your hard-drive. If nothing is specified, `link-` is used as the default value.

The OAI protocol does not require the repository to support the `set` record filter[1]. If you want to harvest a set-enabled OAI repository, you can optionally use the `<set>` setting for this purpose. An empty set value is interpreted as no set value!

When an oai task file is saved, a spool file is automatically placed in the `/var/spool/tkl` directory, and the OAI harvester will fetch and perform the job within a couple of minutes. During execution of the job, the status tag will change from "pending" over "running" to "finished", and after finishing of the job, the spool file will be removed.

The harvested OAI metadata records can be inspected by directing the usual user web interface to `bibliotheca/tasks/oaitklite`, where all records are displayed in the fetched order. Clicking at the first link of a record displays some more details of it.

Although the OAI records are indexed on system boot, or when running

```
/etc/init.d/tkl index
```

they have been initially marked "hidden" and will not be displayed in search result sets.

Notes

1. The Keystone OAI repository supports the set based record harvesting since version 1.4.5