Toolkit Lite Administration

Adam Dickmeiss

Sebastian Hammer

Mike Taylor

Marc Cromme

Toolkit Lite Administration

by Adam Dickmeiss

by Sebastian Hammer

by Mike Taylor

by Marc Cromme

Copyright © 2002, 2003 by Index Data ApS

This document tells you how to install Toolkit Lite (http://www.indexdata.dk/tkl/) on your system.

CVS id: $Id: tkl.xml,v 1.2 2003/11/03 13:44:14 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 ... 4
2.2. Red Hat ... 6
2.3. Web Server ... 8
2.4. XML parser ... 8
2.5. DOM XML ... 9
2.6. Sablotron XSLT ... 9
2.7. Z39.50 Support ... 9
2.8. PHP ... 9
2.9. Installing TKL files ... 10
2.10. Enabling TKL in Apache 1.3.X ... 11
2.11. Enabling TKL in Apache 2 ... 12
2.12. Installing the Sample Portal ... 12
2.13. Starting the Zebra server ... 13
2.14. Testing it ... 14
2.15. Virtual hosting ... 14

3. Content administration ... 16

3.1. Directory structure ... 16
3.2. Documents ... 18
3.3. Schemas ... 18
3.4. User Administration ... 19
3.5. The base portal Bibliotheca ... 19
3.6. Searching ... 20

4. Managing end-user interfaces ... 21

4.1. Document context and the user shell ... 21
4.2. Parameters ... 22
4.3. Session parameters ... 22
4.4. Supporting functions ... 22
4.4.1. tkl-find ... 23
4.4.2. tkl-path ... 25
4.4.3. tkl-search ... 26
4.4.4. tkl-soap ... 27
4.5. Debugging ... 29
4.6. The base portal Bibliotheca ... 29
4.6.1. User interface ... 29
4.6.2. Overall page structure - interface.xsl ... 29
4.6.3. portal.xsd/xsl ... 30
4.6.4. subject.xsd/xsl ... 30
4.6.5. link.xsd ... 31
4.6.6. document.xsd/xsl ... 31
4.6.7. search.xsl ... 31
4.6.8. news.xsd - newspage.xsl ... 32

5. TKLite and OAI Metadata Services 33
5.1. TKL as an OAI repository 33
5.2. OAI harvesting from within TKL 33

Chapter 1. Introduction

TKL (http://www.indexdata.dk/tkl/) is a software system for operating information-driven web portals. (The name originally stood for “portal ToolKit, Light”, but as the level of ambition for the tool has increased, the abbreviation has become largely meaningless).

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 TKL, 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.

TKL 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, TKL can be a useful tool. The system is particularly well-suited to situations where the separation of content from presentation is important.

TKL 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

TKL characterises an information portal as 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, mandatory, what their types are, etc. The XML schema also contains information specific to TKL, for instance whether a given element is multi-lingual, corresponds to a restricted vocabulary, etc.

TKL 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 TKL 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 TKL “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 TKL 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 style sheet, 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 TKL portal generally consists of three basic elements:

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

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 TKL 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 TKL 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 TKL we provide an example of a fairly basic type of portal. It is included as an example of how TKL 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

TKL 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.

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

You need to install the following Debian packages before installing the TKL binaries: `apache`, `php4`, `php4-xslt`, `php4-domxml`, `php4-yaz`, `libxml-libxslt-perl`, and `idzebra`.

The TKL binaries are found in the packages `tklite_1.4.2-1_all.deb` (core PHP and XSLT functionality, necessary package), `libtkl-perl_1.4.2-1_all.deb` (Perl libraries, man pages, necessary package), `tkl_1.4.2-1_all.deb` (init scripts, man pages, cron scripts, Perl binaries, necessary package), `tkl-oai-harvester_1.4.2-1_all.deb` (complete OAI harvester including init scripts, man pages, cron scripts, Perl binaries, additional package), `tkl-web-harvester_1.4.2-1_all.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_1.4.2-1_all.deb, additional package), and `tkl-doc_1.4.2-1_all.deb` (this documentation in HTML and PDF.

It is recommended to think twice before upgrading the `TKL` packages: Some functionality has changed and will not be compatible with behavior fond in the pre 1.2.3 releases. We do recommend to test an upgrade in a safe, non-production sandbox before applying to a production server. In any case should old

`TKL` installations be un-installed using the **dpkg -P** purge options to avoid problems with older init and cron scripts.

The image uploading features are not part of the core TKL package 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 Toolkit Lite 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 TKL 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.

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

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 TKL due to misconfiguring of the bundled Java interface. Therefore, get the tarball from TKL 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. You can also get the source rpm from gingerall, and install that. Version 0.98 (with js-support) is said to work on Red Hat 7 based systems.

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 TKL 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.

Next step is to make a PHP/YAZ dynamic shared object (DSO). We can not use the official PHP source tarball since this version of PHP/YAZ is not compatible with YAZ 2. Therefore we need to fetch the latest PHP/YAZ tarball named `php4-yaz_4.1.2-9.tar.gz` or similar from our Debian php4-yaz (http://ftp.indexdata.dk/pub/phpyaz/debian) download area.

```
$ tar zxf php4-yaz_4.1.2-9.tar.gz
$ cd php4-yaz
$ phpize
$ ./configure
$ make
# make install
```

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

Finally, we need to enable YAZ and 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
# echo 'extension=yaz.so' >/etc/php.d/yaz.ini
```

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

```
extension=xslt.so
extension=yaz.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. TKL has only been tested with Apache (http://httpd.apache.org/) versions 1.3.23-1.3.27 on Linux. We have also tried TKL 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.

2.7. Z39.50 Support

If TKL 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 TKL 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-yaz=/usr/local \
```

```
--with-dom=/usr/local \
--with-xslt-sablot=/usr/local \
--enable-xslt
```

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-yaz` tells PHP to include Z39.50 support. The argument must reflect the location of YAZ. 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. Installing TKL files

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

Get the TKL 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 TKL scripts through the alias, the XSLT processor - which needs to call back into TKL - knows nothing of Apache's `Aliases`, and so can not find the necessary scripts.

Check that the TKL 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.

2.10. Enabling TKL in Apache 1.3.X

The Apache configuration must be modified in order to enable TKL 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

2.11. Enabling TKL 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
```

2.12. 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.

2.13. 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 &
```

2.14. 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 `tkl-portal` on `localhost` you would use `http://localhost/tklite/admin.php?cwd=bibliotheca`.

2.15. Virtual hosting

For TKL 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 TKL 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 TKL 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 TKL'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/tkl on the server at www.indexdata.dk (so that the portal can be reached using the URL http://www.indexdata.dk/tkl/), and the administrative scripts are available under the directory HTDOCS/tkl-admin, the admin interface can be invoked like this:

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

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: TKL 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 TKL 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 TKL distribution. This is the user shell for TKL; 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 TKL 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 (TKL) 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:parameter>, 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/TKL"/>
```

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 TKL.

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 TKL 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-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.

4.4.2. 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.

4.4.3. 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>.

4.4.4. 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.

4.5. 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 TKL-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.

4.6. The base portal Bibliotheca

This section describes the structure of the base portal Bibliotheca, which is included with TKL. 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 TKL.

4.6.1. 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 TKL.

4.6.2. 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]

4.6.3. 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 TKL 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.

4.6.4. 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.

4.6.5. 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.

4.6.6. 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.

4.6.7. 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.

4.6.8. 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. TKLite and OAI Metadata Services

TKLite 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 TKLite record posts using any browser.

5.1. TKL as an OAI repository

TKL 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 asct as a basic OAI data provider, but the level of protocol support is intended to develop over time as requirements warrant. At present, the OAI-PMH ’verbs’ Identify, ListMetdataFormat, and ListRecords, are supported.

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

TKL 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 `schemaname--metadataprefix.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 tkl-test1 contains a file named `tkl-test1/schemas/link--oai_dc.xsl` which defines the translation from a link-type document to the oai_dc format.

The base URL for the OAI server associated with a TKL 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
```

5.2. OAI harvesting from within TKL

In addition, TKLite allows the system administrator to define OAI harvesting tasks. To do so, one must install the `libtklite-perl` and the `tklite-oai` Debain packages (Not really packaged yet...).

The OAI daemon called `tkl-oai` will in future be packaged such that it will restart on every boot, and can be started and interrupted manually issuing the commands

```
/etc/init.d/tklite-oai start
/etc/init.d/tklite-oai stop
/etc/init.d/tklite-oai restart .
```

Meanwhile, one unpacks the tarball somewhere, and issues the following commands as user `www-data`:

```
cd /some/where/tklite-utils
source scripts/setenv.sh
perl/bin/tkl-oai -D /var/spool/tklite ,
```

where one must make sure that the spool directory `/var/spool/tklite` exists and is owned by the user `www-data`. In case that debugging output is wanted, use the following command:

```
perl/bin/tkl-oai -D /var/spool/tklite -v .
```

Now the daemon is ready to accept OAI harvesting tasks.

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 TKLite 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 task file. Choose "oai" as task type, fill in the starting url (remember the trailing slash when adrssing a TKLite OAI server!), for the moment ignore the "filter" and "depth" options by filling a "0", and type the target directory relative to the portal root- for example "/tasks/oaitklite/". 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>
  <filter type="domain" action="allow">0</filter>
  <distance>0</distance>
  <target>/tasks/oaitklite/</target>
  <description>OAI harvesting job at our toolkit lite server</description>
  <status>finished</status>
</task>
```

When a task file is saved, a spool file is automatically placed in the `/var/spool/tklite` 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/tklite index ,
```

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