mod-xslt2 Users and Administrators Manual

Carlo Contavalli

Date: 2004/02/17 02:59:38 - Revision: 1.3

mod-xslt2 is a web server module able to transform xml documents in any format using xslt stylesheets, doing what might be called server side parsing of xml files.

1. License, copyright and...

2. Introduction

Nowdays, most of the browsers on the market *do not* support parsing xml files and are not able to correctly apply xslt stylesheets.

Even worse, some browsers are not standard complaiant and do not follow the specifications closely, leading to a world where xml can hardly be used in web applications.

mod-xslt2 is a server side module able to transform “xml” documents in “html” (or to any other format) before they even get back to the browser.

At time of writing, this module can be used with apache 1.3.x (stable) and apache 2.{1,2}.x (stable), but other web servers may get supported in the future.

mod-xslt2 main features include:

- the ability to parse generated xml (ability to parse the output of php or perl scripts).
- the ability to use the “xslt” indicated by the <?xml-stylesheet processing instruction.
- the ability to send back the xml file unparsed to the browser.
- the ability to fetch xslt or DTDs from scripts rather than from static files.
- the ability to fetch a different xslt depending on the content of the request headers, of the get parameters or the web server environment.
- the ability to allow xslt stylesheets to make use of these variables to generate output.

This module is almost a complete rewrite of the original “mod-xslt” written by Philipp Dunkel, containing many improvements, a more generic API, a whole set of new and flexible features and the correction of some bugs that were found in the original sources.

As an effect of a more generic API, this version of mod-xslt2 supports both apache 1 and apache 2, while work is under way to provide a cgi and proxy wrapper to allow mod-xslt2 to be used on any other server.

The more generic API should also be usable to provide support for servers like IIS or others, even if I’m personally not going to write code for closed source products.

The build system has also been completely rewritten and a whole set of new documentation included in the original package.

Keep also in mind that I’m not native english, and as though I’m trying to do my best, *please* report any error or “strange” sentence you may find in this (or any other) document provided with mod-xslt2.

3. History

It all started a rainy day when I decided to try writing some xml pages to put on my web server. I started looking on google to find some way to let apache parse the pages, and a whole bounce of projects came up. I started downloading .tar.gz, compiling, and testing them out... however, I couldn't find anything that suit all my needs: one didn't process pi instructions at all, the other one was not able to read xml to be parsed from php or cgi scripts, another one was not able to load dtds nor to fetch xslt from http urls.

The situation was sad: there were tens of mod xslt available but none of them had even a subset of "standard" features, many add constraints like "you cannot use php xml handling functions, since they use the same xml library as php", and many others were just buggy and almost completely unmaintained. So, I took the most promising one, mod-xslt, written by Philipp Dunkel, and started adding all the features I needed. However, at the time mod-xslt run only on apache2, which was still quite buggy to my eyes.

So, I ended up almost completely rewriting mod-xslt (just a few lines are still there..) and adding a whole bounce of new features.

4. Installation

4.1. Prerequisites

To install this module, you must:

1. Make sure "libxml2" and "libxslt" and their headers (libxml2-dev and libxslt-dev) are correctly installed on your system, and that the commands "xslt-config" and "xml2-config" can be found in your path.
2. Make sure you have a version of "libxslt" above 1.0.30.
3. Make sure "libpcre" (at least version 4.5) and its headers (libpcre-dev) are correctly installed on your system and that the command "pcre-config" can be found in your path.
4. Make sure to have a version of Make from the GNU project, often known as "gmake" on many systems. If your system does provide a "gmake" command, use that instead of "make" in all the provided examples.

You may also consider recompiling libxml2 or libxslt after applying some of the patches provided in the "patches/" directory in mod-xslt2 sources. For further information about the available patches, please read the file "README.Patches".

4.1.1. Apache 1.3.x

To install mod-xslt2 on apache 1.3.x, you need to have the apache headers installed (apache-dev) and the command “apxs” available.

You also need to know the path of “apxs”, which can be found by running something like “locate apxs”, “whereis apxs” or “find / -name apxs”.

4.1.2. Apache 2.{1,2}.x

To install mod-xslt2 on an apache 2.1.x or 2.2.x server, you need to have:

- the apache headers file installed (apache2-dev)
- the command “apxs” or “apxs2” available somewhere on your system (you can find it by running “locate apxs”, “whereis apxs” or “find / -name apxs”)
- “libapr” and its headers installed somewhere on your system (libapr0, libapr0-dev) and the command “apr-config” (the presence of this command usually implies the availability of the library)
- libaprutil and its headers installed somewhere on your system (libaprutil, libaprutil-dev) and the command “apu-config” (the presence of this command usually implies the availability of the library)

4.2. Quick start

From your shell, run as a normal user:

```
$ gzip -cd mod-xslt2*|tar x
$ cd mod-xslt2*
$ mkdir build
$ cd build
$ ../configure --with-sapi=apache1
```

Where “apache1” is the name of the web server you want mod-xslt2 to be compiled for. In place of apache1, at time of writing, you can specify apache2 or none. If you specify “none” you will get the library and the utilities that constitute most of mod-xslt2 compiled and installed.

If you do not specify any “--with-sapi” option, configure will try to figure out which web server is installed on your system. However, if you have more than one web server supported by mod-xslt2 installed, configure will compile mod-xslt2 for the first detected one.

Configure will also try to autodetect most of the needed parameters and will warn you about small incompatibilities it *will* find in the libraries on your system. Sometimes it will also suggest you to apply a

particular patch to your libxml2 or libxslt. Read “README.Patches” to have more information about those patches.

Anyway, after successful configure completion, you should run:

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

If you ever want to remove mod-xslt2, you could also run:

```
# make uninstall
```

4.3. Configure parameters

As any “configure” script, mod-xslt2 configure accepts many parameters. Some of those parameters are always available while a few of them depend on the “web server” mod-xslt2 is being compiled for and on the libraries available on your system.

The following sections will discuss all the parameters accepted by configure. However, the “configure --help” screen is authoritative, as this document may get outdated.

4.3.1. Installation related parameters

The following parameters can be used to choose where mod-xslt2 will be installed:

- *--bindir=BINDIR* where BINDIR is the directory where you want the binaries (modxslt-parse, modxslt-perror, modxslt-config) to be put during installtion
- *--libdir=LIBDIR* where LIBDIR is the directory where you want the library (libmodxslt0) to be put during installation
- *--includedir=INCLUDEDIR* where INCLUDEDIR is the directory where you want libmodxslt headers to be put during installation
- *--mandir=MANDIR* where MANDIR is the directory where you want the manual pages for the above listed commands to be put during installation

Usually, the place where to put sapi specific binaries generated by the compilation process is automatically detected. In any case, the following options may influence how the above paths are calculated:

- *--prefix=PREFIX* where, if no “--mandir” is specified, MANDIR is calculated to be “PREFIX/man”, while if no “--includedir” is specified, INCLUDEDIR is calculated to be “PREFIX/include”.

- *--exec-prefix=EPREFIX* where, if no "--bindir" is specified, BINDIR is calculated to be "EPREFIX/bindir", while if no "--libdir" is specified, LIBDIR is calculated to be "PREFIX/libdir".

Note that if you are, for example, packaging mod-xslt2 and want all the files to be installed under a particular directory, you can use the DESTDIR environment variable as usual.

4.3.2. Compilation related parameters

The following parameters can be used to enable/disable mod-xslt2 features or to choose compilation parameters:

- *--enable-debug* starting from version 1.3.9, this option just indicates that mod-xslt should be compiled with the options -ggdb and -O0, which means: with support for running mod-xslt under gdb, and without optimizations. Before version 1.3.9 this parameter used to be useful to flood your web server error.log with information regarding mod-xslt operations and inner working (it was useful for spotting out problems). This compile time option has been superseeded by a more flexible mechanism which allows users to choose at run time which output they want in their own error.log.
- *--enable-memory* available from version 1.3.9, this option enables strict checking of memory allocations and frees. It is used when running regression tests, and should not be used otherwise.
- *--enable-xslt-debug* enables libxslt debugging functions. This is useful to verify correct behavior of libxslt. Take a look to libxslt documentation for more details.
- *--disable-thread* due to lack or bad multithreading/multimodules support in some of the libraries used by mod-xslt2, some global variables allocated in TSD have been used (do not complain to me). This parameter disables libmodxslt multithreading support (the usage of a TSD to keep global variables).
- *--disable-extensions* mod-xslt2 makes available some xslt extensions, useful for web programmers. Compiling mod-xslt2 with this parameter disables those extensions.
- *--enable-libxslt-hack* libmodxslt sometimes calls error handlers with the wrong descriptors. Enabling this option, will provide one more layer of protection against this kind of error in libxslt (that, at time of writing, has not been corrected by the authors). Another solution is to patch the library to avoid the problem. Please read "README.Patches".
- *--enable-fallback-wraparound* once you get the errors working correctly, if you use the "fallback" tag, you may see strange messages flooding your logs. This option tells mod-xslt2 to remove fallback nodes at least when used inside mod-xslt2 extension elements, to reduce the number of those messages. Another solution is to patch the library to avoid the problem. Please read "README.Patches".
- *--enable-exslt* (obsolete, available only for mod-xslt $<$ 1.3.9) enable exslt extensions for libxslt. Default is disabled. If you need or want to use the advanced string operators, remember to enable this option.
- *--disable-exslt* (available starting with mod-xslt 1.3.9) disable exslt extensions for libxslt. Default is enabled.

- *--disable-xinclude* disable modxslt usage of xinclude. Default is enabled. If you want to disable xinclude or maintain backward compatibility, you are free to disable xinclude.
- *--with-sapi=apache1|apache2|none* allows you to specify for which web server (sapi) to compile mod-xslt2. Currently, you can specify “apache1” for apache version 1.3.x, “apache2” for apache version > 2.x.44, “apache” to enable apache version autodetection or “none” to compile only mod-xslt2 utilities and libraries. More sapi modules are currently being developed.
- *--with-xml2-config=path* allows you to specify where the libxml2 “xml2-config” script is located. If not specifyed, the first one found in the search path will be used. As an example, you could specify something like: “--with-xml2-config=/usr/local/libxml2-2.5.57/bin/xml2-config”

  If you don’t have xml2-config on your system, you probably haven’t installed libxml2 correctly or you haven’t installed the “-dev” version of the packages (rpm & deb). If you don’t know where it is, you can run “locate xml2-config” or “find / -name xml2-config” to locate it.

- *--with-xslt-config=path* allows you to specify where the “xslt-config” script is located. If not specifyed, the first one found in the search path will be used.
- *--with-pcre-config=path* allows you to specify where the “pcre-config” script is located. If not specifyed, the first one found in the search path will be used. If not found, support for “Perl Compatible Regular Expressions” will be *disabled*.

4.3.3. SAPI Specific configure parameters

4.3.3.1. Apache 1

- *--with-apxs* allows you to specify the “apxs” that should be used. By default, the first “apxs” found in the “PATH” or in “/usr/bin”, “/usr/local/bin”, “/usr/local/apache/bin” is used. If you have no apxs on your system, you probably don’t have apache headers (or the package apache-dev) installed correctly.

Note that running “make install” will install mod-xslt2 in the path returned by the “apxs” found (“apxs -q LIBEXECDIR”), eventually prefixed by the DESTDIR environment variabe as usual.

4.3.3.2. Apache 2.x.x

- *--with-apxs* allows you to specify the “apxs” that should be used. By default, the first “apxs” found in the “PATH” or in “/usr/bin”, “/usr/local/bin”, “/usr/local/apache/bin” is used. If you have no apxs on your system, you probably don’t have apache headers (or the package apache-dev) installed correctly.
- *--with-apr-config* allows you to specify the “apr-config” script that should be used. By default, the first “apr-config” found is used.
- *--with-apu-config* allows you to specify the “apu-config” script that should be used. By default, the first “apu-config” found is used.

Note that running “make install” will install mod-xslt2 in the path returned by the “apxs” found (“apxs -q LIBEXECDIR”), eventually prefixed by the prefix specified with “--prefix” to configure.

5. mod-xslt2 Setup and Usage

5.1. Apache 1.3.x

mod-xslt2 can be configured in several ways to be used on apache 1.3. To choose which one suits best your needs, you need a good knowledge of how apache works. The following sections will try to give you the basic knowledge needed to configure mod-xslt2.

5.1.1. Request life

When requesting a document to an Apache 1.3 server through your browser, apache

1. takes the requested URL and remaps it to a “file location”, to a path on the local file system (as an example, “http://www.masobit.net/foo/bar.xml” may become “/opt/array-00/customers/masobit.net/http/bar.xml”)
2. tries to understand the format the document is written into (it looks for the mime type of the document)
3. looks for someone or something able to “read” the provided document type (an “handler”)
4. the handler is passed the job to send the document “over the wire” back to the browser.

As an example, when you request a .php file with something like “www.masobit.net/info.php”, on our server the first step remaps “www.masobit.net/info.php” in something like “/opt/array-00/customers/masobit.net/http/info.php”. Apache then looks in the mime.magic or mime.types (or the AddType directives) for the mime type of the file. Provided the content of those files and those directives are correct, apache will decide the requested file is of type “application/x-httpd-php”.

Apache will then look for a handler able to serve this kind of document, and it will see that “application/x-httpd-php” is handled by the “libphp4.so” module.

Apache will then call a function in this module and let the module directly write the answer back to the browser.

5.1.2. Using the “AddHandler” directive

One good way to let mod-xslt2 handle a request is to use the “AddHandler” or “SetHandler” directive.

Using those directives you can tell apache you want a particular kind of file being handled *directly* by mod-xslt2. For example, you could use something like:

```
AddHandler mod-xslt .xml
```

To tell apache the handler for all xml files needs to be "mod-xslt2". AddHandler can be even activated on a per directory/per location or per file basis. For example, you could activate xml parsing in a given directory by using something like:

```
<Directory "/opt/foo/">
  AddHandler mod-xslt .xml
</Directory>
```

If you want to parse all the files in a given directory as xml files regardless of their extension you could use something like:

```
<Directory "/opt/foo/">
  SetHandler mod-xslt
</Directory>
```

AddHandler and SetHandler are the "fastest" way to use mod-xslt2. The drawback is that *this method won't work if you set mod-xslt2 up to handle .php files, since they won't be parsed* by the php module. Infact, as explained previously, apache will call mod-xslt2 instead of "libphp4.so" to send the document back to the browser.

5.1.3. Using the XSLT directives

In case you need to apply stylesheets to dynamically generated documents, you thus need to use the mechanism provided by mod-xslt2.

This mechanism has nothing to do with the mechanism described in the previous sections and does not conflict with it. Keep in mind, however, that the following directives need to be used *only if you want to parse dynamically generated files*, like php, perl or cgi.

Before anything else, you need to enable the XSLT Engine for a given directory, using the "XSLTEngine <on|off>" directive. Once enabled, mod-xslt will be called for *every* file in the given directory that apache will be required to serve.

However, while coding the module, we had the choice to:

- check the mime type of every apache reply, and parse it if it was of type text/xml (note: on most systems, text/xml is application/xml...).
- check the mime type only of *some* requests, and parse them only if they were of type text/xml.

Since checking the type of a reply is quite expensive in terms of system resources, we decided to go with the second choice. You thus need to tell mod-xslt2 which requests you want it to check for xml output to

parse, by using the “XSLTAddFilter” parameter. As an example, if you want to apply an xslt stylesheet to the output of the php scripts in one of your directories, you need to use something like:

```
<Directory
  "/opt/array-00/customers/masobit.net/http/php-xml/">
  XSLTEngine on
  XSLTAddFilter application/x-httpd-php
</Directory>
```

However, keep in mind that the output of a given script will be parsed if and only if it outputs xml data and sets the mime type to “text/xml”, so, in php, you need to use something like “header("Content-Type: text/xml")” before anything else in your scripts.

Remember: you need to use “XSLTEngine on” only if you need to parse dynamic pages.

5.1.4. Mixing the two

As a rule of thumb, you can use “AddHandler” for any “static document” and “XSLTEngine” with “XSLTAddFilter” with any “dynamic document”.

A complete example could be the following:

```
...
LoadModule mxslt_module /usr/lib/apache/modules/mod_xslt.so
AddModule modxslt.c
...

XSLTTmpDir /tmp

  # Always parse .xml files using the
  # specified stylesheets
AddHandler mod-xslt .xml

  # In this directory, some .php scripts
  # output xml to be parsed - those
  # scripts need to set the “Content-Type”
  # header to text/xml if they want
  # a stylesheet to be applied. Otherwise,
  # they will be ignored
  # header("Content-Type: text/xml")

  # Note also that it is sometime useful
  # to specify application/xml instead,
  # which is the default for most systems
<Directory /var/www/xml>
  XSLTEngine on
  XSLTAddFilter application/x-httpd-php
</Directory>
```

In the example above, only php scripts in "/var/www/xml" will be parsed provided they output a Content-Type header set to "text/xml". If you want to parse them regardless of the Content-Type, thus regardless of the type of data they are outputting, you can use the apache directive "XSLTAddForce" with the same syntax of XSLTAddFilter.

5.1.5. Loading the module

Regardless of which method you may decide to use to parse your xml data, keep in mind you always need to tell apache to load the module. To do so, add a line like the following to your httpd.conf:

```
LoadModule mxslt_module /usr/lib/apache/modules/mod_xslt.so
AddModule modxslt.c
```

Beware that the second parameter must be the full path were mod_xslt got installed. Since the path is detected by querying "apxs", it will probably be the same as any other apache module. If you don't know where apache modules are kept on your system, use something like "apxs -q LIBEXECDIR" or look to other LoadModule directives in your configuration files.

5.1.6. mod-xslt Configuration parameters

- *XSLTEngine <on|off>* per directory, per file, per virtual host or in global configuration file, allows you to enable or disable XSLT extra features.
- *XSLTTmpDir <directory>* per directory, per file, per virtual host or in global configuration file, allows you to specify which directory mod-xslt2 will use to create temporary files. By default, "/tmp/mod-xslt2" is used. Keep in mind that "/tmp/mod-xslt2" must exist in your system. Path must be absolute: "/tmp" good, "/var/tmp" good, "tmp" *bad*, "./tmp" *bad*.
- *XSLTAddFilter <MimeType>* per directory, per file, per virtual host, or in global configuration file, tells mod-xslt2 to parse files of the given mime type as if they were xml files. Keep in mind that the file is parsed only if the content type is set to "text/xml" or "application/xml".
- *XSLTDelFilter <MimeType>* per directory, per file, per virtual host, or in global configuration file, tells mod-xslt2 not to parse files of the given mime type anymore. This is needed since mod-xslt2 per directory configurations are hinerited from parent directories.
- *XSLTAddForce <MimeType>* per directory, per file, per virtual host, or in global configuration file, tells mod-xslt2 to parse files of the given mime type as if they were xml files, independently from the resulting content type.
- *XSLTDelForce <MimeType>* per directory, per file, per virtual host, or in global configuration file, tells mod-xslt2 not to parse files of the given mime type anymore. This is needed since mod-xslt2 per directory configurations are hinerited from parent directories.
- *XSLTSetStylesheet <MimeType> <Stylesheet>* per directory, per file, per virtual host or in global configuration file, tells mod-xslt2 to use the given stylesheet for all files of the given MimeType, independently from any "<xml-stylesheet..." or processing instruction available into the document. The MimeType is usually something like text/xml or application/xml, telling all such documents need to be transformed using the specified stylesheet.

- *XSLTUnSetStylesheet <MimeType> <Stylesheet>* per directory, per file, per virtual host or in global configuration file, tells mod-xslt2 to forget about a previous “XSLTSetStylesheet”. This is needed since mod-xslt2 per directory configurations are hinerited from parent directories.
- *XSLTDefaultStylesheet <MimeType> <Stylesheet>* per directory, per file, per virtual host or in global configuration file, tells mod-xslt2 that, in case an xml file does not contain any “<xml-stylesheet...” or “<xslt-stylesheet...”, for the given MimeType the specified xslt stylesheet should be used. Same things as for XSLTSetStylesheet.
- *XSLTNoDefaultStylesheet <MimeType>* per directory, per file, per virtual host or in global configuration file, tells mod-xslt2 that to forget about a previous “XSLTDefaultStylesheet”. This is needed since mod-xslt2 per directory configurations are hinerited from parent directories.
- *XSLTUnlink <on|off>* per directory, per file, per virtual host or in global configuration file, tells mod-xslt2 that temporary files are not to be deleted. This option was provided to simplify debugging of newly created documents: combined with a per directory “XSLTTmpDir” and using dynamic documents provided by php or perl, the temporary file will keep the xml document generated by your scripts, simplifying debugging. You can find the temporary file that generated an error by reading the error log.
- *XSLTParam "variable" "value"* per directory, per file, per virtual host or in global configuration file, tells mod-xslt2 to pass the given “variable” to the stylesheet with the indicated “value”. Those variables are accessible from the stylesheet using the mod-xslt2 extension value-of, with something like: <mxslt:value-of select="$MODXSLT[variable]" ... look to the variable substitution paragraph for more details...
- *XSLTAddRule "stylesheet" "condition"* per directory, per file, per virtual host or in global configuration file, tells mod-xslt2 to use the specified stylesheet if all conditions specified in “condition” are met. Any modxslt-stylesheet or xml-stylesheet contained in the document is then ignored, unless the selected stylesheet is not loadable or does not work, in which case the rule is ignored.
- *XSLTDelRule "stylesheet"* per directory, per file, per virtual host or in global configuration file, tells mod-xslt2 to forget about the rule regarding the specified stylesheet
- *XSLTDebug category, category, category, ...* per directory, per file, per virtual host or in global configuration file, increases verbosity of mod-xslt2 logging. Within mod-xslt, each message is tied to one or more categories. When XSLTDebug is specified, all messages whose categories have been specified in XSLTDebug are printed, and usually stored in error.log. Categories are: "config" (parsing of configurations), "debug" (useful when debugging with gdb or other tools), "flags" (causes the category of messages to be output as well), "libxml" (parsing done by libxml), "parser" (parsing of PI and mod-xslt expressions at top of .xml files), "proto" (related to HTTP protocol, headers, and protocol parsing), "rules" (for apache1, output debugging messages related to application of *Rules*), "sapi" (output messages related with the interaction of the specified SAPI), "variables" (output messages regarding parsing and handling of variables), "verbose0", "verbose1", "verbose2" (to enable outputting of messages at verbosity 0, 1, and 2), "all" to enable all debugging messages (probably too verbose for normal usage).
- *XSLTDebugMask <number>* per directory, per file, per virtual host or in global configuration file, equivalent to XSLTDebug. The only difference is that instead of specifying categories by name, you can use an integer. The integer represents the bitwise OR of the values corresponding to each category. Note that the integer representation of categories can change from mod-xslt version to version. You should normally not use this parameter. Have a look into modxslt0/modxslt-debug.h, mxslt_debug_e, to see the mapping betweeen categories and parameters.

- *XSLTDisableSignature<number>* per directory, per file, per virtual host or in global configuration file, doesn't do anything, only for backward compatibility.

5.1.7. Parameters usage examples

5.1.7.1. XSLTSetStylesheet

XSLTSetStylesheet is most useful when you have .xml documents that do not specify any xslt stylesheet to be used for the parsing.

You can put all those documents in a given directory on your web server, and then use something like:

```
<Directory /documents/without/stylesheet>
  XSLTSetStylesheet default_stylesheet.xsl
</Directory>
```

All the files in /documents/without/stylesheet would then be parsed using default_stylesheet.xsl, independently from any <?xml-stylesheet or <?modxslt-stylesheet specifyed in the document.

XSLTSetStylesheet parameters are hierarchically propagated in subdirectories. This means that if you want to disable one of the stylesheet you previously set, you need to use XSLTUnSetStylesheet.

5.1.7.2. XSLTAddFilter and XSLTAddForce

XSLTAddFilter and XSLTAddForce can be used to tell mod-xslt which files to parse.

They both take as a first argument a MIME type. This MIME type is used by mod-xslt to identify the scripts/files that may output xml to be parsed.

So, in order for mod-xslt to parse dynamic documents, you need to tell him which "kind of documents" may output xml.

Weel, those "dynamic documents", however, may decide not to output xml and output something else.

XSLTAddFilter thus tells mod-xslt to watch a given mime type, verify if the output is xml, and only if it is, parse it into something else.

XSLTAddForce, instead, watches a given mime-type, and it tells mod-xslt to parse the output in any case, even if it doesn't look like xml. This instruction may be used if you have cgi or dynamic scripts which output the wrong mime type.

5.1.7.3. XSLTAddRule

XSLTAddRule has been added since mod-xslt 1.3.6, snapshot >= 2004100100. This parameter allows you to specify a stylesheet to be used for all documents selected by the apache directive being used only if the specified condition, written as a mod-xslt expression (see the dedicated section), is met.

Rules are checked by mod-xslt in the same order as specified, and the first one matching specifies the stylesheet to be used to parse the document, independently from any <xml-stylesheet or <modxslt-stylesheet being specified in the document.

Here are some examples:

```
<Directory /xml>
  XSLTAddRule "local://style_mozilla.php?LANG=$GET[LANG]"
   "$HEADER[User-Agent] =~ '/mozilla/'"

  XSLTAddRule "local://style_printer.php?LANG=$GET[LANG]"
   "$GET[format] =~ '/printer/'"
</Directory>
```

(above examples have been split on multiple lines for readability) Note that the stylesheets can be of any of the supported kinds, and that mod-xslt performs variable substitution in the stylesheet URL.

Also note that in case the stylesheet contains errors or is not loadable for any reason, the rule is ignored and parsing goes on using the stylesheets specified by the document.

5.1.8. Logging

In order to process requests, mod-xslt2 needs to create temporary files. Temporary files are used to process dynamic requests, and contain the XML that got to mod-xslt2 to be parsed. It is often useful to know which temporary file was associated with which request, especially if the unlinking of temporary files is disabled.

mod-xslt2 saves the name of the temporary file being used in a "request note" that can be retrieved by using something like "%{mod-xslt-tmp}n" in the "LogFormat" directive, with something like:

```
LogFormat "%v %h %l %u %t \"%r\" %>s %b
 (%{mod-xslt-tmp}n)" mxslt_format
```

```
CustomLog logs/mod-xslt2.log mxslt_format
```

5.1.9. Increasing performance

The only way in apache 1.3.x to intercept the output of other modules is to provide a suitable file descriptor where to store data.

Since mod-xslt2 is part of apache itself, a pipe is impossible to use, unless we fork apache one more time, slowing things down.

The simplest approach has thus been used: creating a temporary file, let other modules write the replies in there, and then parse the temporary file. However, by using temporary files, we hit I/O performance issues.

One of the greatest performance improvements would thus be to mount a ramdisk (either a “shm” or “rd” in linux), over the directory used by mod-xslt2 for temporary files.

Other methods are under investigation and may get supported in future versions of mod-xslt2:

- Having an external daemon parse data, transmitted from apache through a unix socket. This will be done after implementing the proxy module, which is almost the same. This would also be useful to simplify cache implementation.
- Provide, as file descriptor, the file descriptor of /dev/null, and use the “callback” provided by apache to store data in memory. In this case, however, we would hit memory problems for big files. However, other solutions may be used (mmapping a file? using libxml push method? does it parse data on the fly or simply keeps the buffers for later parsing?)

Another performance issue is due to:

- external http or ftp connections to fetch .xsl or .dtd files
- dns lookups to understand if a remote host is in practice a remote host or a local one

The latest of the two problems can be solved either by using a faster name resolution mechanism (take a look to nsswitch.conf or to the hosts file) or by paying some attention while writing .xml+.xslt file and by explicitly telling mod-xslt2 when to use local connections (will be explained later on).

5.1.10. Subrequest Issues

To avoid security and some concurrency issues (see the section about security concerns), mod-xslt2 for apache 1 tryes to avoid remote connections as much as possible, specially if those connections will loop back to the localhost.

However, apache accepts any connection it receives on any of the addresses it is listening on, and is thus hard to understand which connections will loop back to the local host.

By default, when mod-xslt2 starts, it tries to understand on which addresses apache is listening on. However, when you write your apache configuration file, you have two choices:

- Explicitly listing all the ip addresses to listen on (using the "Listen" directive with something like "Listen 127.0.0.1:80" or by using "BindAddress" - which is deprecated by the apache group)
- Just specify one or more ports, and let apache listen on all interfaces on all ip addresses (simply using the "Port" directive without any "Listen", or by using one or more "Listen" with something like "Listen 80 8080")

In the first case, mod-xslt2 will use the ip addresses provided with the "Listen" directive to detect remote connections.

However, if you use the "Listen" directive by just specifing the port(s) to listen on or you just use the "Port" directive, mod-xslt2 will have to try to understand which are all the ip addresses available on the operating system, which is very system dependent and quite unportable.

At time of writing, mod-xslt2 configure script will try to detect if the needed functions to get all the ip addresses of the operative system are available, in which case the autodetection code is compiled in.

However, if those functions are not available, mod-xslt2 will complain any time you use the "Port" directive or the "Listen" directive without explicitly specifing the ip addresses to listen on, by printing in the logs something like:

```
INADDR_ANY is being used without ioctl support -
 read mod-xslt2 README!
```

In this case, just change any "Listen" directive you have like this:

```
Listen 80 8080
```

in something like

```
Listen 127.0.0.1:80 192.168.0.1:80
 127.0.0.1:8080 192.168.0.1:8080
```

where "127.0.0.1" and "192.168.0.1" are the only ip addresses apache will listen on. If you don't have any listen directive, just add them. Watch out that, if you have many ip addresses to listen on, apache performance will decrease (by listing them all instead). In this case, the best bet would be to improve mod-xslt2 detection code and write some that will work on your platform. Please mail me if you do so, or mail me if you need help in doing so. Unfortunately, at time of writing, I have access only to "Debian GNU/Linux" machines, and cannot tell if the detection code will work on any other platform.

5.2. Apache 2.x.x

Apache 2.x support is now considered stable (2.0, 2.2, ...), thanks to all the people who provided feedbacks on mod-xslt mailing lists.

At time of writing, there is only one known issue about mod-xslt and apache 2.x.x: as a filter, it is not very easy for mod-xslt to return status pages different than those set by the handler (like 404 or 500 pages), and while it works with most document types, it may not work with _all_ document types (depending on the handler providing the given type).

For example, if a php4 script (where php4 is handled thanks to the php4 apache2handler sapi) outputs invalid xml code, mod-xslt tries to tell apache2 to output a 500 error page. However, the mod-xslt request is handled by the php4 handler and the connection is instead dropped. Other handlers may have similar problems. If you encounter some, please report them to one of the mailing lists. At time of writing, I have no idea on how to correct this problem, beshide handling error documents by myself (in mod-xslt) or patching php4 apache2handler. If anyone has suggestions, please contact me.

5.2.1. Configuring Apache 2.x for mod-xslt

To use mod-xslt with apache 2.x.x, you just need to tell apache you want to use mod-xslt, by inserting a line like the following in your httpd.conf (or apache.conf):

```
LoadModule mxslt_module /usr/lib/apache2/modules/mod_xslt.so
```

Where `/usr/lib/apache2/` is the path where all your modules are kept. Note that on *most* systems, apache2 modules are kept in `/usr/local/libexec`, so the correct LoadModule directive should be:

```
LoadModule mxslt_module /usr/local/libexec/mod_xslt.so
```

Note however that this path can be changed during apache2 configuration, so please look to where other modules are kept, or run the command “apxs2 -q LIBEXECDIR” or “apxs -q LIBEXECDIR”.

If you don’t know this path, just look for other “LoadModule” directives in your configuration file or run the command “apxs2 -q LIBEXECDIR”, which will show you the correct path.

Once you tell apache to load mod-xslt, you need to tell him for which files you want mod-xslt to be used. To do so, you can use one of the following directives:

- *AddOutputFilter mod-xslt <extension>...* tells apache we want mod-xslt to parse all files with extension “extension”.
- *AddOutputFilterByType mod-xslt <mime-type>...* tells apache we want mod-xslt to parse all files with the specified mime-type. Note that the mime-type should indicate which files we want mod-xslt to parse. Most common values are text/xml or application/xml, depending upon the configuration of your system.

- *SetOutputFilter mod-xslt* tells apache that we want all files in a given directory or location or virtual host to be parsed by mod-xslt.

Watch out! Just use one of those directives. If you use more than one, your documents will be parsed more than once, and unless your first pass outputs .xml to be parsed again, an error will be signaled by mod-xslt.

For example, you may enable mod-xslt in a given directory with something like:

```
<Directory /this/is/a/directory>
  AddOutputFilterByType mod-xslt text/xml
  ...
</Directory>
```

Note that on most system both .xml and .xsl files are considered of mime type application/xml. We often suggest to change that default and set the mime type of .xml files to text/xml and of .xsl files of text/xsl. You can usually use constructs like “AddType text/xml .xml” to force a mime type of text/xml to .xml files...

If you know before hand that all files in a given directory should be parsed using mod-xslt, you may also use something like:

```
<Directory /this/is/another/directory>
  ...
  SetOutputFilter mod-xslt
</Directory>
```

To have further details about the discussed parameters, please take a look to the apache manual, http://httpd.apache.org/.

It is important to note that it is possible to block mod-xslt from parsing a particular document by using the SetEnvIf parameter and the SetEnv family of configuration directives. mod-xslt will infact *NOT* perform any transformation if the “no-xslt” environment variable is set.

5.2.2. mod-xslt Configuration parameters

- *XSLTSetStylesheet <MimeType> <Stylesheet>* per directory, per file, per virtual host or in global configuration file, tells mod-xslt2 to use the given stylesheet for all files of the given MimeType, independently from any “<xml-stylesheet...” or processing instruction available into the document.
- *XSLTUnSetStylesheet <MimeType> <Stylesheet>* per directory, per file, per virtual host or in global configuration file, tells mod-xslt2 to forget about a previous “XSLTSetStylesheet”. This is needed since mod-xslt2 per directory configurations are hinerited from parent directories.

- *XSLTDefaultStylesheet <MimeType> <Stylesheet>* per directory, per file, per virtual host or in global configuration file, tells mod-xslt2 that, in case an xml file does not contain any "<xml-stylesheet..." or "<xslt-stylesheet...", for the given MimeType the specified xslt stylesheet should be used.
- *XSLTNoDefaultStylesheet <MimeType>* per directory, per file, per virtual host or in global configuration file, tells mod-xslt2 that to forget about a previous "XSLTDefaultStylesheet". This is needed since mod-xslt2 per directory configurations are hinerited from parent directories.
- *XSLTParam "variable" "value"* per directory, per file, per virtual host or in global configuration file, tells mod-xslt2 to pass the given "variable" to the stylesheet with the indicated "value". Those variables are accessible from the stylesheet using the mod-xslt2 extension value-of, with something like: <mxslt:value-of select="$MODXSLT[variable]" ... look to the variable substitution paragraph for more details...
- *XSLTAddRule "stylesheet" "condition"* per directory, per file, per virtual host or in global configuration file, tells mod-xslt2 to use the specified stylesheet if all conditions specified in "condition" are met. Any modxslt-stylesheet or xml-stylesheet contained in the document is then ignored, unless the selected stylesheet is not loadable or does not work, in which case the rule is ignored.
- *XSLTDelRule "stylesheet"* per directory, per file, per virtual host or in global configuration file, tells mod-xslt2 to forget about the rule regarding the specified stylesheet
- *XSLTDebug category, category, category, ...* per directory, per file, per virtual host or in global configuration file, tells mod-xslt2 to print (in the log files) the debugging messages corresponding to the specified categories. Note that any given message can be part of more than one category. It will be outputted only if all the categories are to be outputted according to this parameter. Currently defined categories are: config (parsing of configurations), debug (useful when running gdb, debugging crashes or trying to find errors in the debug system itself), flags (if enabled, the category of the messages/mask is also outputted in integer format), libxml (parsing done by libxml), parser (parsing of PI and mod-xslt expressions at top of .xml files), proto (related to HTTP protocol, headers, and so on), rules (for apache1, output debugging messages related to application of *Rules*), sapi (output messages related with the interaction of the specified SAPI), variables (output messages regarding parsing and handling of variables), verbose0, verbose1, verbose2 (to enable outputting of messages at verbosity 0, 1, and 2).
- *XSLTDebugMask <number>* per directory, per file, per virtual host or in global configuration file, tells mod-xslt2 to print (in the log files) the debugging messages corresponding to the specified categories. This is exactly like the XSLTDebug parameter, but allows to specify which messages to output as an integer. The integer represents the OR of the integer value corresponding to each category whose output is desired. Note that internally mod-xslt maps every debug category to an int, and every category is mapped to a "name", as indicated above. This parameter is useful when you either want to output all messages (specify -1 as the number) or when there is some category of messages that has not been assigned a str name yet. Have a look into modxslt0/modxslt-debug.h, mxslt_debug_e, to see the mapping betweeen categories and parameters.
- *XSLTDisableSignature<number>* per directory, per file, per virtual host or in global configuration file, doesn't do anything, for backward compatibility only.

5.2.3. Apache 2.x.x, mod-xslt and PHP4

In order to use php4 with apache2, you can compile it using two different SAPI:

- *apache2filter* - where php4 is used as an Apache 2.x.x FILTER
- *apache2handler* - where php4 is used as an Apache 2.x.x HANDLER

To know more about the differences between HANDLERs and FILTERs in apache 2.x.x, please refer to apache 2.0.x documentation.

To know more about how to compile php4 using the two SAPI or about the differences between the two, please refer to php4 documentation.

At time of writing, however, if you compile php4 to run under apache2 it will be compiled using the HANDLER sapi.

mod-xslt is now being tested using only this SAPI, and only very old versions of mod-xslt have been tested with the FILTER sapi.

6. Writing XML for mod-xslt2

Although mod-xslt2 uses standard libraries to parse xml data and transform it, there are few things to keep in mind while writing xml/xslt to be parsed by mod-xslt2.

6.1. XSLT Parameters

Before anything else, you can access many mod-xslt2 parameters using the standard “value-of” xslt tag or standard XPath expressions.

As an example, to put the value of the version of modxslt being used in the output generated by a stylesheet, you could use something like:

```
  # To output version of mod-xslt2
<xsl:value-of select="$modxslt-version" />

  # To check interface version
<xsl:if test="$modxslt-interface &lt; 1">
  mod-xslt2 interface version is greater than 1!
</xsl:if>
```

At time of writing, the following variables are made available to the xslt file by mod-xslt2:

- *modxslt-interface* - holds the interface version being used by mod-xslt2. It is changed every time a new variable is added to this list and every time a new element is introduced (see section element extensions). Variables are not supposed to be ever removed, so you can safely assume any greater version number is backward compatible. Current version is “2”. Unless otherwise specified, variables have been introduced in interface version 1. In version 2, the only added variable is modxslt-conf-xinclude.
- *modxslt-sapi* - name of the sapi which is now parsing the xml document. It is usually set by the application making use of libmodxslt0.
- *modxslt-name* - name of mod-xslt2.
- *modxslt-handler* - name of handler being used by mod-xslt2.
- *modxslt-namespace* - URL of the namespace for mod-xslt2 extensions.
- *modxslt-conf-libpcre* - has value “true” if mod-xslt2 was compiled with libpcre support.
- *modxslt-conf-exslt* - has value “true” if mod-xslt2 was compiled with exslt support.
- *modxslt-conf-xinclude* - introduced in interface version 2 - has value “true” if mod-xslt2 was compiled with xinclude support.
- *modxslt-conf-extensions* - has value “true” if mod-xslt2 was compiled to provide extension elements (see dedicated section)
- *modxslt-conf-libxmlthreads* - has value “true” if libxml supports threads.
- *modxslt-conf-libxslthack* - has value “true” if configure was given the parameter “--enable-libxslt-hack”
- *modxslt-conf-fallbackwrap* - has value “true” if configure was given the parameter “--enable-fallback-wraparound”
- *modxslt-version* - its value is the current version of the mod-xslt2 being used (example: "1.2.3")
- *modxslt-version-major* - its value is the first digit of the version (in the example above, "1")
- *modxslt-version-minor* - its value is the second digit of the version (in the example above, "2")
- *modxslt-version-patchlevel* - its value is the third digit of the version (in the example above, "3")

Variables that have value “true” when a feature is enabled, have value “false” when it is not.

6.2. mod-xslt2 Extensions

mod-xslt2 allows you to access many other variables by providing custom extension tags.

Those extension tags are available only when you compile mod-xslt2 without “--disable-extensions” and if you enable them in your xsl by specifying something like:

```
<xsl:stylesheet version="1.0"
```

```
    extension-element-prefixes="yaslt"
    xmlns:yaslt="http://www.mod-xslt2.com/ns/1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

Note the “extension-element-prefixes” and “xmlns:yaslt="http://www.mod-xslt2.com/ns/1.0" that specify that the extensions will live in the “yaslt:” namespace.

However, enabling the extensions will allow you to use two more additional tags:

- *header-set* - to set an output header. The only valid attribute is “name”, you can use to specify the name of the output header you want to set.
- *value-of* - to fetch a mod-xslt2 specific variable. The only valid attribute is “select”. The value of the “select” attribute will be parsed as a “mod-xslt2” expression, which follows completely different rules than XPath expressions.

Since those tags are provided by extensions, you need to specify the namespace every time you use them. In the example above, the namespace to use would be “yaslt:”, as specified by the “extension-element-prefixes” and “xmlns” attributes.

A more complete example may be the following one:

```
<xsl:stylesheet version="1.0"
    extension-element-prefixes="yaslt"
    xmlns:yaslt="http://www.mod-xslt2.com/ns/1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="faq">
    <yaslt:value-of select="$HEADER[Host]" />

    <yaslt:header-set name="X-Powered-by">
      <xsl:value-of select="$modxslt-version" />
    </yaslt:header-set>

  </xsl:template>

</xsl:stylesheet>
```

6.2.1. header-set

header-set allows you to set a value in the http headers that will be returned back to the client. Any name is accepted, as is any value. If “strip-space” (<xsl:strip-space elements=...) is active for the given element, any sequence of blank characters or new lines is replaced by a single space. This feature allows you to specify multi-line headers (probably invalid for the http protocol) while keeping everything working. Note that header-set won’t try to prevent you from doing stupid things. That’s up to you. The only thing you won’t be able to do is to set the “Content-Type”, which is handled by some specific code.

Anyway, “header-set” requires just the “name” attribute to be specified, to select which header you want to set. Between the opening “header-set” and the closing “/header-set”, you can use any any element you want, just watch out for new lines and strange characters that may invalidate your headers.

The following are all valid usage examples of “header-set”:

```
<xsl:stylesheet version="1.0"
    extension-element-prefixes="yaslt"
    xmlns:yaslt="http://www.mod-xslt2.com/ns/1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  [...]
    <yaslt:header-set name="X-Powered-by">
      <xsl:value-of select="$modxslt-version" />
    </yaslt:header-set>

    <yaslt:header-set name="X-Fuffa">
      fuffa
    </yaslt:header-set>
    <yaslt:header-set name="X-Foo">
      <xsl:apply-templates />
    </yaslt:header-set>
  [...]

</xsl:stylesheet>
```

6.2.2. value-of - modxslt expressions

value-of allows you to fetch any mod-xslt2 specific variable or expression.

While writing mod-xslt2 code, I decided to keep most of mod-xslt2 variables in a completely independent and isolated namespace mainly for three reasons:

- Some of the variables names and values are gathered from the running environment, which, by no means, can be considered safe or trusted.
- For the same reason, mod-xslt2 variable names can violate XPath specifications, and I didn’t want to employ weird name mangling routines.
- Work is underway to cache xml and xslt meta data to speed things up. In order to do so, it is however necessary to intercept any access to modxslt variables.

For the same reasons, I decided to go with my own (simple) language to parse expressions.

6.2.2.1. Simple expressions

An expression is a list of characters which contains one or more variables. Each variable starts with a “$” symbol and is followed by the name of the variable or by the name of the variable enclosed in curly brackets (“{”, “}”). The following are all valid expressions:

```
0: "this is fuffa"
1: "$fuffa"
2: "${fuffa}"
3: "this is ${fuffa}"
4: "this is much $fuffa more"
```

Expression 0 is a “simple” string, no replacements are done, while expression 1 is replaced by the value of variable “fuffa”.

Expression 2 is exactly like expression 1, beside the fact that using “{” “}” would allow this variable to be correctly replaced even in a string like “ababa${fuffa}ababa”. Expression 3 would be replaced by the value of variable “$fuffa” preceded by “this is”, much like in expression 4.

Non existent variables are replaced by the empty string (they are removed), while a “$” which should be part of the string should be escaped by preceding it with a “\”. This also implies that any “\” should be itself escaped by using a double back slash (“\\”).

6.2.2.2. Indirect references and built up variable names

The usage of “${” and “}” allows you to use indirect references: let’s say that “$foo=bar” and that “$bar=fuffa”, evaluating the expression “${$foo}” would at first be replaced by “${bar}” and then replaced by “fuffa”, much like in bash programming.

There is more to say: inside a “{” and “}” you could even put more than one variable or character constants, to “build” up the name of the variable you want to be replaced.

Let’s make one more example and let’s say “$fuffa=hello”, “$foo=fuf” and “$bar=fa”. In this case, we would have the following output (given the expressions on the left):

```
${$foo$bar} -> ${fuffa} -> hello
${${foo}$bar} -> ${fuffa} -> hello
${${foo}fa} -> ${fuffa} -> hello
${fuf$bar} -> ${fuffa} -> hello
```

Using those expressions, you could have as much fun as you want and recurse as much as you want (and your stack holds).

6.2.2.3. Predefined variables (array like variables)

The ability of parsing variables is not such a good thing if we don't have variables to parse. However, mod-xslt2 provides a rich set of predefined variables. Variables are grouped in classes that look like "arrays". The main arrays available are:

- *MODXSLT* - holds the same variables as passed as parameters to the xslt parser (described in the previous sections). Those variables hold informations like how mod-xslt2 was compiled or what features are enabled
- *GET* - contains variables passed to your xml file as GET parameters (http get)
- *HEADER* - contains the headers passed to mod-xslt2 by your web server (headers that, in turn, were given by the client)

As often happens, it is easier to explain the usage of something by showing some examples than trying to explain how it works:

```
$GET[fuffa]
$HEADER[User-Agent]
$MODXSLT[version]
$MODXSLT[namespace]
```

In the examples above, the first line fetches the variable "fuffa" that was passed as a get parameter to the xml file (with something like http://host.fqdn/file.xml?fuffa=value).

The second line is replaced by the header "User-Agent" provided by the client, while the third and fourth lines are replaced respectively by the value of the parameter $modxslt-version and $modxslt-namespace.

Using GET, you can access any "get" parameter that was passed to your xml file while HEADER gives you access to any header that was sent to you by the client.

Keep also in mind that, as explained in the previous sections, it is possible to build up the name of a variable from other variables. So, the following is also valid:

```
$HEADER[$GET[header-to-check]]
${$source[User-Agent]}
```

Watch out not to insert any space between the square brackets ("[", "]") and the array index, since they are not allowed and the variable won't be substituted.

6.2.2.4. Setting variables

It is also possible to create custom variables from .xml files to be passed over to the xslt. To do so, you need to use the Processing Instruction "modxslt-param" with the attributes "name" and "value", where "name" specifies the name of the variable to be set while "value" specifies its value.

Using this processing instruction, you can also override the value of predefined variables, like $GET[fuffa], but not of mod-xslt2 constants, like $MODXSLT[version].

However, let's see a complete example of using modxslt-param:

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

<!DOCTYPE fuffa SYSTEM "dtd/fuffa.dtd">

<?modxslt-param name="variable" value="its value" ?>

[...]
```

The variable "$variable" would thus be accessible from your xslt using modxslt own "value-of", with something like "...:value-of select="$variable"".

Note also that some SAPI allow you to pass over parameters to your xml or xsl files from configuration files. Refer to the SAPI specific section of this manual.

6.2.3. Verifying availability of mod-xslt2 extensions

Before using any of the mod-xslt2 xslt extensions described in the previous sections, you should make sure they are available on your system and version of mod-xslt2.

As a first test, you can verify your xslt is being used by mod-xslt2 by checking the value of the XPath param "modxslt-interface". If available, you can then verify that "modxslt-conf-extensions" has value true, and assume extensions are available.

One alternative to using mod-xslt2 params is to use the standard functions defined in http://www.w3.org/TR/1999/REC-xslt-19991116#extensions:

```
boolean element-available(string)
boolean function-available(string)
```

to verify mod-xslt2 extension tags are available, with something like:

```
<xsl:if test="element-available('yaslt:value-of')">
  <yaslt:value-of select="$HEADER[User-Agent]" />
</xsl:if>
```

assuming that in the opening "xsl:stylesheet" tag you specified as extension name "yaslt".

There's one more way to handle the availability of mod-xslt2 extensions: using the "xsl:fallback", like in this example:

```
<yaslt:value-of select="$HEADER[User-Agent]">
  <xsl:fallback>
    Sorry, cannot access headers
     (no mod-xslt2 extensions available)
  </xsl:fallback>
</yaslt:value-of>
```

In this case, if “yaslt:value-of” is not available the xslt processor will parse the content of the “xsl:fallback” node. In any other case, the “xsl:fallback” node will be completely ignored.

However, at time of writing, libxslt-1.0.32 has a few bugs handling fallback nodes:

- when the extension is available, the fallback node is not ignored as ought to be and a warning is printed in your web server logs (read the FAQ for more information about this issue)
- when the warning is printed, libxslt-1.0.32 calls the error handler with the wrong parameters, possibly causing a segmentation fault (it happens if libxslt debugging was enabled, in which case one of the pointer passed to mod-xslt2 is not null and considered to be valid)

There are a few ways to avoid this problem:

- patch the library in order to avoid the warning to be printed
- patch the library in order for the correct arguments to be always passed to error functions
- compile mod-xslt2 specifying “--enable-fallback-wraparound”, to ask mod-xslt2 to remove “fallback” nodes when the extensions are available
- compile mod-xslt2 specifying “--enable-libxslt-hack”, to ask mod-xslt2 to always enable debugging using some wrappers, in order to avoid the error highlighted above

Those are really two libxslt bugs, where the first one triggers the second one. Correcting one of the two should be enough for mod-xslt2 to work. However, the best way to solve the problem is by using the two highlighted patches. Please read “README.Patches” to know more about those issues.

6.3. Setting the Content-Type (MIME type) of the parsed document

As indicated in the previous sections, you are not allowed to use “header-set” to try to change the “Content-Type” of the document.

However, you can choose the mime type of the document by specifying the attribute “media-type” in the “xsl:output” tag, as shown in the example below:

```
<xsl:output media-type="text/html"
 encoding="ISO-8859-1" />
```

If no media-type is specified, mod-xslt2 will try to guess it (relying on libxml2 parsing), probably returning back to the client a media-type of “text/plain”, “text/xml” or “text/html”.

Keep in mind that you can specify any “media-type” you want.

6.4. Choosing the stylesheet to use

mod-xslt2 decides which stylesheet to use thanks to a 3 steps procedure:

1. if any suitable <?xml-stylesheet or <?modxslt-stylesheet processing instruction is found, the specified stylesheet is used, regardless of any XSLTSetStylesheet parameter. The first suitable xml-stylesheet or modxslt-stylesheet is chosen.
2. if a XSLTSetStylesheet is provided for the given document type, the specified stylesheet is used.
3. in any other case, a 500 server error is returned (the rationale is that once mod-xslt2 is told to parse the document, mod-xslt2 will either output the parsed document or send an error back to the client).

We already talked about XSLTSetStylesheet in the SAPI specific sections, and there’s not much more to say here.

However, as you can see in the first step, mod-xslt2 can be told which stylesheet to use using two different processing instructions: xml-stylesheet and modxslt-stylesheet.

6.4.1. xml-stylesheet and modxslt-stylesheet

xml-stylesheet is a standard xml directive, and its description and usage can be found on http://www.w3.org/TR/xml-stylesheet. As you probably already know, xml-stylesheet can be used to associate a given xml file with the xslt to be used and generally looks something like:

```
<?xml-stylesheet
 type="text/xsl"
 href="./xslt/faq-http.xsl"
 media="screen"
 alternate="no"
 title="For any web browser"
 charset="ISO-8859-1"?>
```

For a given stylesheet to be considered “suitable” by mod-xslt2 to parse an xml file, the following conditions must be met:

- type - must be either “text/xml” or “text/xsl”.
- href - should contain the url of the stylesheet. See the next section on “Accepted urls” .
- media - either “all” or “screen”. In case of “screen”, it can be followed by one or more “media expressions”. The stylesheet will be considered “suitable” only if one of the expressions is evaluated to have a “true” value. The “empty” expression is considered to always match (be true). See the section on “Media expressions” .

At time of writing, the other fields are of no interest to mod-xslt2.

The main difference between “xml-stylesheet” and “modxslt-stylesheet” is that the first one should be used in a way compliant with the highlighted standards, while the second one may use any mod-xslt2 extension.

Keep in mind, however, that mod-xslt2 will not complain if you use its extensions in the standard “xml-stylesheet” tag.

modxslt-stylesheet is a processing instructions that takes exactly the same arguments as xml-stylesheet, and it has been introduced mainly for three reasons:

- As not being standardized, it is interpreted only (and exclusively) by mod-xslt2. If a browser or another xslt processor finds any modxslt-stylesheet directive, it will completely ignore it. This allows few useful tricks that will be discussed in the next few sections.
- currently, the standard says that the “media” attribute “may” contain expressions, which are not yet defined in any standard and that may be defined in the future (AFAIK). Browsers or xslt-processors are thus supposed to ignore them and skip anything after the first “word” followed by a space until the first comma. However, mod-xslt2 supports its own language for expressions, language that may be used in “modxslt-stylesheet” without fears to conflict with future standards or incompatible browsers.
- as you probably know, in case a stylesheet is not found or is not valid, mod-xslt2 will give the user back a “server error” (the rationale is that if a document was meant to be parsed the inability to do so is an error). However, the “modxslt-stylesheet” pi provides a simple mechanism to return back to the browser the plain xml in order to let the browser parse it. This feature, combined with “media expressions”, would allow you to configure mod-xslt2 to parse only those xml files that would cause problems to the various browsers on the market.

A couple examples will be given in the following sections.

6.4.1.1. Media expressions

As you probably know, the “media” attribute in a xml-stylesheet (or modxslt-stylesheet) processing instructions may contain a comma separated list of “media types” for which the stylesheet should be used to parse the xml data.

A media attribute usually looks something like:

```
<xml-stylesheet ... media="screen, printer"...
```

where the media “all” is a sort of wildcard, specifying that a stylesheet could be used to parse xml data to be outputted on any media.

Any xslt processor should also ignore anything from the first space following the name of a media type to the first comma (or end of attribute), since “in the future” some kind of “expressions” may be introduced by the standards.

Any xslt processor looking for a “screen” media stylesheet, should thus accept anything like

```
<xml-stylesheet media="screen fuffa fuffa, printer"
or
<xml-stylesheet media="screen foo bar"
```

where “fuffa fuffa” or “foo bar” are “Media expressions” that should be ignored.

mod-xslt2 introduces its own language to specify media expressions.

A mod-xslt2 expression is usually a boolean expression preceded by an “and” made of a list of tests that must be passed for the stylesheet to be used. Tests may make use of mod-xslt2 variables and of many operators that will be described in the following sections.

Mod-xslt grammar to parse expressions could be described by a BNF similar to the following:

```
bool_expr:
 | cmp_expr
 | cmp_expr ’,’
;

cmp_expr: ’(’ cmp_expr ’)’
 | ’!’ cmp_expr
 | cmp_expr BooleanOperator cmp_expr
 | String StringOperator String
 | String
;
```

Where capitalized strings are terminals and lowercase strings are non-terminals.

Associativity of operators and their precedence will be shown in the next sections.

6.4.1.1.1. Strings

A string can be specified by using the name of a variable, a sequence of characters that match the regular expression:

```
[^[:blank:]*+%/\"\’,!><=~()-]*
```

or a sequence of any character enclosed in single or double quotes (', "), where any “'” or “"” part of the sequence itself must be escaped by prepending it with a “\”, while a “\” with no special meaning should be escaped with another slash.

> **Note:** Since mod-xslt2 1.3.9, escaping rules have changed. Version preceeding (<) 1.3.9 used “\” as a normal escape character. Any occurrence of “\” followed by any other character was replaced by the character following without “\” (\a was replaced by just a). Since version 1.3.9, a “\” is considered *only* (and only in this case!) when followed by some sensible character, where the only sensible character are: $, ' and ". For example, \a in version 1.3.9 is left as \a, while \" is transformed into ". This change has been introduced in order to simplify escaping of regular expressions in mod-xslt2

media expressions. Also note that the parsing code has been changed, so a $ followed by an invalid character for a variable name does not need escaping.

Strings enclosed in single or double quotes may also contain “mod-xslt2 expressions”, as described in the section “value-of - mod-xslt2 expressions” and contain any of the specified variables.

6.4.1.1.2. String Evaluation

As shown in the BNF grammar, a String can be used both in a “boolean” or “cmp” context (either, checked with a BooleanOperator or StringOperator). In boolean context, a String is considered true if it does not correspond to the empty string ("") or if the variable is defined (has a value associated with it, regardless of what the value is).

6.4.1.1.3. Boolean Operators

mod-xslt2 recognizes the following left associative boolean operators:

- 1 - “!” - logical negation (left associative)
- 2 - “and” - logical and (left associative)
- 2 - “or” - logical or (left associative)

Where the precedence of the operators is determined by the number (lower the number, higher the precedence).

6.4.1.1.4. String Operators

mod-xslt2 recognizes the following string operators:

- “==” or “=” - equal, true if the left side of the operator is equal to the right side. At time of writing, “=” and “==” have the same meaning and return a true value if the memory representation of the string on the left is the same of that on the right. In the future, “==” will maintain this meaning, while “=” will be used to compare the value of the number on the left with that of the number on the right (taking care of roundings and of processor precision limits).
- “!=” - unequal, true if the left side of the operator is not equal to the right side. By equal we mean that the memory representation of the string on the left is the same as that on the right.
- “=~” - perl regular expression matches, true if the regular expression on the right of the operator matches the string on the left. See next section on regular expressions for more details.
- “!~” - perl regular expression does not match, true if the regular expression on the right of the operator does not match the string on the left. See next section on regular expressions for more details.

- “>”, “>=”, “<”, “<=” - true respectively when the string on the left of the operator, converted to a “real”, is greater, greater or equal, less, less or equal, to the value of the string on the right converted to a “real”.

6.4.1.1.5. Perl Compatible Regular Expressions

The operators “=~” and “!~” allow you to match a String with a perl compatible regular expressions, also known as PCRE.

mod-xslt2 makes use of “libpcre” to parse and apply those expressions. The complete reference of those regular expressions can thus be found in perlre(1) on any unix system with perl installed, while a quick tutor and introduction can be found on perlretut(1) or perlquick(1) and on any book about perl programming.

In mod-xslt2, PCRE are specified by enclosing them in a “separator”, and by indicating one or more options after the second occurrence of the separator. A separator may be any character beside “\”, which can be used to escape the separator itself if needed in the regular expression. Additionally, regular expressions may be enclosed in single or double quotes, to overcome the limits of characters a String can contain as specified in the previous sections.

Keep also in mind that, as being specified in a xml attribute, any entity must also be escaped using standard xml notation.

In any case, the following options may be specified:

- *i* - using this option, case insensitive matching is performed
- *e* - using this option, the “$” matches only the end of the string and does not match any newline the string may contain
- *a* - using this option, the match must be “anchored”, which means it must match from the beginning to the end of the string
- *s* - using this option, the “.” matches also newlines
- *x* - using this option, spaces inside a regular expression are ignored unless they are escaped
- *X* - using this option, you will enable libpcre features not compatible with perl. At time of writing, enabling this option will cause an error every time an unknown character is escaped (prepended with a “\”)
- *m* - using this option, you will enable multiline matching
- *y* - this option “inverts the greediness of the quantifiers, so that they are not greedy by default, bug become greedy if followed by ?” (from pcreapi(3))
- *u* - this option causes PCRE to consider both the pattern and the string as made of UTF-8 encoded characters

The following are all examples of valid regular expressions:

```
'/fuffa/i' - match "fuffa", "Fuffa", "FUFFA",
 "abfuffabc"...
'$bap$ia' - match "bap", "Bap", ...
'$a\\\$$i' - match any string containing "a$". In this
 case, "$" is escaped twice to avoid it being
 considered "the terminator" and to avoid any
 meaning as regular expression special character.
'&amp;fuffa&amp;i' - exactly like the first example,
 but using "&" as separator
```

Note that I have always enclosed them in quotes, to avoid causing problems to the parser.

6.4.1.1.6. Examples of complete media expressions

```
<modxslt-stylesheet ... media="fuffa" ... >
```

will return false, and the stylesheet not applied.

```
<modxslt-stylesheet ... media="all" ... >
```

will return true, and the stylesheet applied.

```
<modxslt-stylesheet ... media="screen" ... >
```

will return true, and the stylesheet applied.

```
<modxslt-stylesheet ... media="screen and
  '$HEADER[User-Agent]' =~ '/msie/i'" ... >
```

will return true only if the header "User-Agent" contains the string "msie" compared using case insensitive matching.

```
<modxslt-stylesheet ...
  media="screen and
     '$HEADER[User-Agent]' =~ '/msie/i' or
          '$HEADER[User-Agent]' =~ '/Moz.*1\.0/'"
   ... >
```

will return true if the header "User-Agent" contains the string "msie" compared using case insensitive matching or contains the string "Moz" followed by any number of any character as long as it is followed by the string "1.0".

```
<modxslt-stylesheet ...
  media="fuffa, screen and $GET[ignorebrowser] or
          '$HEADER[User-Agent]' =~ '/Moz.*1\.0/'"
   ... >
```

will return true when the xml page was called by a browser with a get parameter "ignorebrowser" with any value (with something like

http://url.of.xml.document.org/path/to/xml/document.xml?ignorebrowser=1) or if the header "User-Agent" contains the string "Moz" followed by "1.0".

```
<modxslt-stylesheet ...
  media="fuffa, screen and
           $MODXSLT[interface] >= 1 and
       $MODXSLT[sapi] = apache1"
```

will return true when the interface version of mod-xslt2 is greater than or equal to 1 and when the sapi which is parsing the xml document is "apache1". This is especially useful if your xsl stylesheet rely on some server specific variable/feature being available.

Keep also in mind that you can use as many "moxslt-stylesheet" or "xml-stylesheet" as you want. In any case, the first matching one will be used by mod-xslt2. If none applyable will be found, a 500 error page will be returned. The rationale is that if you want a document to be parsed (and configure mod-xslt2 to do the parsing), the document will be either be parsed and returned back to the browser or an error given back without disclosing of any additional information.

6.4.1.2. href URLs

Well, up to now we have seen how our .xml files can specify which xslt to be used. However, we haven't seen where the .xsl files can be stored and how we can specify their path.

6.4.1.2.1. Supported URL schemes

At time of writing, mod-xslt2 (thanks to libxml2), can fetch xsl, dtd or other .xml files using one of the following methods:

1. remote http URL
2. remote ftp URL
3. local file URL
4. local http URL

The first three methods are quite standard and made available thanks to libxml2 handlers. However, there are a few things to keep in mind:

- a http URL *must* start with "http://" and could contain a port number, an username and a password, like in `http://carlo%password@www.masobit.net:7568/file.php`.
- a ftp URL *must* start with "ftp://" and follow the same conventions as the http URL shown above.
- a file URL *may* start with "file://" or just be an absolute or relative path. If it starts with "file://", "file://" is removed from the URL and the remaining path is opened. Thus, something like "file:///file.xsl" points to "file.xsl" in the "root" (/) of the file system, while "file://file.xsl" indicates "file.xsl" in the same directory of the xml file referring to that url. Both could been given using a path like "/file.xsl" or "file.xsl", without any preceding "file://" (which is stripped anyway).

In short, it should follow the same syntax and behavior as indicated on rfc 1738, 1808 and 2396.

There is a fourth url scheme supported by mod-xslt2: local://.

local:// behaves exactly like a local file URL, but tells mod-xslt2 that the file should be fetched using the http protocol. This scheme allows you to easily use .xsl or dtd generated by cgi or php scripts, without using a remote connection.

As you may notice, there are thus at least three good reasons to use “local://” instead of “http://”:

- Before anything else, “local://” is like the “file://” scheme, and allows you to easily specify urls relative to the path of the file using the url itself. For example, if you fetch http://www.masobit.net/fuffa/doc.xml and doc.xml requires “local://xslt/doc.xsl”, the file “http://www.masobit.net/fuffa/xslt/doc.xsl” will be fetched using the http protocol (and thus, allowing doc.xsl to be a php script or cgi-bin).

  In contrast, “local:///xslt/doc.xsl” would refer to “http://www.masobit.net/xslt/doc.xsl”, much like “file://”.

- In second instance, if you use virtual domains and you want to make use of locally generated xsl files, you cannot reliably use the “http://” scheme. As you may guess, in “virtual” environments “http://localhost/xslt/doc.xsl” would not be necessarily the same as “http://www.masobit.net/xslt/doc.xsl”, and there would be no way to specify a relative url if “local://” was not available.
- Instead of using an outgoing connection, when a “local://” url is specified a “subrequest” is made, allowing
  - faster retrieval of generated documents
  - easy detection of loops (more about loops will be discussed later)
  - to avoid a deadlock that may halt mod-xslt2 (and that halts any mod-xslt2 I have seen on the internet) under very high loads (which a malicious user may use to cause a denial of service)

6.4.1.3. HTTP glinces

If you use http urls, either for external DTDs or for xsl stylesheets, watch out for a small problem it may arise: libxml2, by default, when fetching a remote document, does not check for the http status of the web server. The result is that, libxml2, will try to parse everything that will be returned back to the browser, even a 404 or 500 error page.

Some believe this behavior is correct, others believe error pages should be considered exactly like a file system error. However, independently from what I think, this behavior leads to strange results when

parsing documents: for example, if a DTD is missing on the local file system, the problem is ignored, but if it is missing from a remote url, the 404 page is parsed as the looked up DTD, and a fatal error is produced, telling the DTD is invalid. Additionally, in your error log you would see wierd errors telling you about invalid lines in DTDs you know nothing about and you cannot find anywhere in the file system.

So, beware, if you find strange errors regarding DTDs or xslt you never wrote, verify they are not the 404 or 500 error pages returned back by a remote server.

There's one more thing to say: I personally don't like this behavior. I tryed to get rid of it in several ways, but the only way out I found was either rewrite the http client (or include my own with mod-xslt2) or patch libxml2 library.

The first solution didn't seem quite realistic to me, so, in the "/patches" directory of the mod-xslt2 tarball, you'll find two patches:

- The first one makes libxml2 verify the error status of the remote web server, and return an error if it is not a 200 or 3xx (in which case the page the client has been redirected to will be fetched)
- The second one makes libxml2 export some private data allowing anybody making use of it to freely choose what to do in case of errors

Personally, I believe the first patch may break things up. On your system, you may have applications that rely on libxml2 parsing error pages (I personally believe those applications should be considered broken anyway). The second patch, instead, should not break anything.

6.4.1.3.1. URL, media and type substitutions

Additionally, any URL specified in a "href" attribute or the value of the "type" and "media" attribute in a <xml-stylesheet or <modxslt-stylesheet may contain any mod-xslt2 expression , that will be replaced the first time the expression itself is used.

As an example, you may specify something like:

```
<?xml-stylesheet type="text/xsl"
  href="http://www.mbit.net/xslt/faq.php?
    lang=$GET[lang]&agent=$HEADER[User-Agent]"
```

This is useful to pass over parameters to php or cgi scripts. As you may already have noted, any generated xsl stylesheet is able to access to mod-xslt2 variables, but the cgi or php script itself does not have any access to them, unless you pass them over as get parameters.

Right now, POST requests are not supported and will never be unless somebody decides it worth and either works on it or bagges me to do so.

Note also that URL should be correctly encoded by replacing xml entities and by replacing any dangerous character with the corresponding "%" value.

6.4.1.3.2. Returning raw xml back to the browser

Using the processing instruction “xml-stylesheet” to select a stylesheet to be used, you must specify the “href” of the stylesheet to be used. Failure of doing so will result in an error be outputted in your logs. However, if you use “modxslt-stylesheet”, you can omit the “href” of the stylesheet to be used. In this case, the xml will be returned back to the browser without further processing.

This is quite useful if you know a particular browser is able to correctly parse your xml files: in this case, you can use a “media expression” to match the browser, and specify no href for the xml to be returned back raw to the client, with something like:

```
[...]
<modxslt-stylesheet
  type="text/xsl"
  media="screen and
    '$HEADER[User-Agent]' =~ '@Gecko.*1\.4@'"
  alternate="no" title="For Mozilla web browser"
  charset="ISO-8859-1" ?>

<modxslt-stylesheet
  type="text/xsl"
  href="local://xslt/links.xsl"
  media="screen and
    '$HEADER[User-Agent]' =~ '@Links@'"
  alternate="no" title="For Links web browser"
  charset="ISO-8859-1" ?>

<xml-stylesheet
  type="text/xsl"
  href="local://xslt/any.xsl"
  media="screen"
  alternate="no" title="For any other web browser"
  charset="ISO-8859-1" ?>
[...]
```

In the example above, the xml file would be returned raw if the request was made by “mozilla”, would be parsed using “xslt/links.xsl” if the request was made by “Links” while it would be parsed using “any.xsl” if the request was made by any other web browser.

If “mozilla” was used, the raw document would be then parsed by mozilla itself, that would ignore any “modxslt-stylesheet” and use as a stylesheet “local://any.xsl”. However, mozilla itself wouldn’t understand “local://” urls and return an error. Thus, in any “raw” document returned by mod-xslt2, “local://” urls are replaced by standard “http://” urls pointing back to the virtual domain that was used to issue the request, allowing mozilla to parse the document without problems.

Additionally, any mod-xslt2 variable used in <xml-stylesheet processing instruction is replaced before returning back the raw xml document to the browser.

This feature will be especially useful as more browsers will correctly support xml and xsl transformations.

Note, however, that variables used in <modxslt-stylesheet pi are not replaced: the main reason not to parse them is that expressions may loose their meaning if variables are replaced (... screen and 'Mozilla/5.0 Gecho/20031010 Debian/1.4-6' =~ '@Gecko/.*1\.4@' ??, doesn't seem too smart), and that a browser is able to understand those PI it should be given all the needed informations to perform expression evaluation by itself.

6.5. Using external DTDs

Some believe DTDs are useless to parse xml documents or to generate http output. However, as you probably know, DTDs may be used to provide defaults for certain tags or provide the definition of entities used by your xml document.

However, DTDs are not always useful and parsing one additional document may be not a bearable overhead.

The approach used by mod-xslt2 is to parse externa DTDs only if the document is declared not to be standalone. Thus, the standalone attribute in your xml declaration is not ignored:

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

tells mod-xslt2 not to load external DTDs, while:

```
<?xml version="1.0" encoding="ISO-8859-1"
 standalone="no" ?>
```

tells mod-xslt2 your xml file needs them. Even if you tell mod-xslt2 to make use of DTDs, just an error will be printed if they are missing (unless they are fetched using the http protocol, look to the section "HTTP Glinces").

6.6. Testing xml files and stylesheets from the command line

Before putting a xml page or newly created php script on your web server you may want to test the generated output statically on your local machine.

Since it may not always be possible to use a web server to verify them, you may be interested in some command line utilities you may found useful.

6.6.1. xsltproc

xsltproc is a tool provided in the libxslt package which can be used to process xml files from the command line. As being provided by libxslt, however, it does not support mod-xslt2 extensions and uses different error handling routines.

To use it, you just need to type "xsltproc file_to_parse.xml". The output will be printed on stdout. In case of errors, they will be printed on stderr.

xsltproc may be useful mainly for two purposes: you can see what a standard browser (that does not support mod-xslt2 extensions) would do with your xml documents, and you can profile the parsing times. xsltproc provides the "--timing" parameter which allows you to know which xslt instructions required the greatest amount of time to generate the output document, with something like "xsltproc --timing file_to_parse.xml".

6.6.2. modxslt-parse

modxslt-parse looks quite similare to xsltproc. The main difference is that modxslt-parse supports all mod-xslt2 extensions and that does not support any command line parameter, beside the name of the file to parse (output is sent to standard output). It is quite useful to verify a particular .xml file off-line from your command line. You can use it by simply typing something like:

```
modxslt-parse file.xml
```

from your command line. Output will be sent to stdout, while errors to stderr. Headers set will be discarded. It internally uses exactly the same engine as mod-xslt2.

6.6.3. rxp

"rxp" is a tool provided in the rxp package on http://www.cogsci.ed.ac.uk/~richard/rxp.html.

It can be used to verify validity of xml files or well-formedness. It should be used to verify the output of your scripts or the validity of your xml files before putting them on line.

6.7. Other tools provided

6.7.1. modxslt-perror

Since strerror is not thread safe on many systems, it cannot be used to translate "errno" error codes in to more readeable (for human beings) strings.

If you see strange “errno: x” error codes in your logs, just use something like:

```
modxslt-perror x
```

to know which error verifyed during parsing. The value of “x” is really system dependent, so, I cannot tell you beforehand what the “x” errno error means on your systme, unless you run modxslt-perror (which asks your operative system what it means).

6.7.2. modxslt-config

modxslt-config can be used to query mod-xslt2 configure and installation parameters. It is usually useful only if you are encountering problems in using mod-xslt2 (problems like inability to load libraries, linking failures...) or if you are writing code for mod-xslt2 (to know the build parameters to be used).

It can also be used to verify if mod-xslt2 is installed on the system.

7. Security considerations

As any code that runs with the same privileges as your web server, there are some dangers you should be aware of and some considerations that should be made.

7.1. Variables substitution

In the previous sections, you have seen you can use mod-xslt2 variables to build up hrefs to be used as the url of the xslt to be used.

Keep in mind, however, that GET and HEADER variables were given to you by the browser and that their values cannot be trusted.

As an example, you could specify something like:

```
<modxslt-stylesheet ... href="/data/xslt/$GET[lang].xsl" ...
```

But what happens here if “$GET[lang]” contains characters like “../../../etc/passwd”? We cannot tell for sure. For sure, mod-xslt2 would try to open a file like “/etc/passwd” and use it as a stylesheet. Obviously, this wouldn’t be considered a valid stylesheet and a 500 error returned back to the client. But, keep in mind that mod-xslt2 both in apache 1.3 and 2.x runs with the same privileges as the web server itself, and would thus be able to read others .xsl files if you specify variables like that in the href of the stylesheet.

Unless you have very good reasons to do so, I would suggest you to always use “local://” or “http://” urls when you need to make use of untrusted variables, and specify those variables as arguments to cgi or php scripts, which, in turn, may verify the genuinity of untrusted variables. The example above, could be made more “security aware” by using something like:

```
<modxslt-stylesheet ...
    href="local:///getxsllang.php?lang=$GET[lang]" ...
```

Another solution would be to use one single stylesheet, and verify from there the validity of variables by using standard “xsl:...” constructs and act accordingly.

7.2. Avoiding deadlocks under heavy loads

Ok, let’s say you have a .xml file that needs a .xsl file to be fetched from a remote url in order to be correctly parsed.

Let’s say you are using apache 1.3 and that a browser issues a request to get the content of that file. Now, apache would receive the request, mod-xslt2 would be called to handle that request and a connection be open to fetch the remote stylesheet. Now, beshide the risks involved with letting your customers upload “static” pages able to force your apache to make outgoing connections, if, for any reason, that connection comes back to your site, your apache is at risk: if we set up apache to handle at most 100 requests at a time (100 children) and we get 100 requests for that page *at the same time*, mod-xslt2 will not be able to connect to the remote host to fetch the stylesheet (no more children available), and will wait for an apache child to become available. However, no apache child will become available until at least one mod-xslt2 is done, and no mod-xslt2 will be done until at least one child becomes available or the tcp connection expires (after a long time).

Thus, by having .xml files that use .xsl stylesheet through http urls that in a way or another point back to the same apache server, we are at risk. Somebody could easily DoS us by simply running something like: “while :; do wget http://remote.url/file.xml & done;”

Most of the other apache modules (not limited to .xml parsing modules) out there on the internet simply don’t worry about this kind of problem.

However, mod-xslt tries as hard as it can to avoid this kind of deadlock: any local:// request is handled using a sub request, which doesn’t use any other apache process, exactly like http:// connections that resolve to any of the addresses you set up apache to listen on.

Beware, however, that if you have NAT boxes or redirectors, mod-xslt2 will not be able to distinguish between local and remote urls and won’t be able to avoid this kind of deadlock.

Keep also in mind that if mod-xslt2 does not know the ip addresses your server is listening on, it won't be able to help you out neither. So, your OS must support the IOCTL needed to get all the ip addresses it is listening on or you must explicitly list them in the apache "Listen" directive (instead of using "*").

Experimental code to autodetect this sort of condition is under testing, but it may introduce security risks. If you want to know more about it, please contact one of the authors.

7.3. Avoiding remote URLs in substitutions

If I were you, I'd also try to avoid as much as possible using variables when indicating path for:

- other files to be included
- dtds or xslt to be used

or, at least, I'd try to explicitly specify the url scheme to be used. For example, if I'd really need to do something like:

```
<?modxslt-stylesheet ... href="$GET[theme].xsl" ... ?>
```

I'd change "$GET[theme].xsl" either in "file://$GET[theme].xsl", "http://hostname/dir/$GET[theme].xsl" or "local:///path/$GET[theme].xsl". The first one in facts is quite dangerous:

- an attacker could use your server for a DoS againsta somebody else (by specifying the http:// url of somebody else)
- could specify urls to gather data about remote hosts
- could specify urls to very big files and make lot of requests and make you eat up all your bandwidth

By always specifying the url scheme and host explicitly, we would limitate the range of action of an attacker significantly.

This is just another issue about using untrusted variables.

8. Reporting BUGS / Helping out the project

To report a bug, just drop a mail to one of the mod-xslt mailing lists. This version of mod-xslt is quite new, and has been used on few platforms and operative systems... so, it is very important to us to get bug reports and users feedback, so please make sure, if you find a bug to drop us a mail.

If you do so, please include as much information as you have: something like "it doesn't work" does not help us much in fixing the problem. What does not work? Can you configure it? Compile it? Install it?

How did you configure Apache? How can you say it does not work? Are you trying to access a document on your web server? Which document? Under which folder? What kind of document is that? Is it a POST request or a GET request? Is it a static document or a dynamic one? Does it use DTDs? How is the xslt referenced? What do the logs say? ...

Sometimes, it may be useful to have the .xml you used at hand, some other times it may be useful to get a dump of your client connection. In the first case, just attach your xml file. In the second, run something like “tcpdump -nei eth0 -s 8192 -w ./file.dump host yourclientipaddress and port 80” to get a trace in “file.dump”.

If you want to help mod-xslt development, just drop us a mail in one of the mailing lists. We are always happy to get new hands at work.