

XSLT-process minor mode

for version 2.1
August 2001

by Ovidiu Predescu

Copyright © 2000, 2001 Ovidiu Predescu.
All rights reserved.

Distributed under the terms of the [GNU General Public License](#) as published by the [Free Software Foundation](#); either version 2 of the License, or (at your option) any later version.

The *XSLT-process* minor mode manual may be reproduced and distributed in whole or in part, in any medium, physical or electronic, so long as this copyright notice remains intact and unchanged on all copies.

1 What is it?

XSLT-process is a minor mode for XEmacs or GNU Emacs which transforms it into a powerful XML editor with XSLT processing and debugging capabilities.

You can invoke an XSLT processor of choice on the current XML file you're editing, and see the results in another buffer. In case of errors, the error messages generated by the XSLT processor, are shown in a compilation buffer, which allows you to quickly go to the location of the error.

You can also run the XSLT processor in debugging mode, and view step by step what the XSLT processor is doing. You can view the current stack frame in the XSLT program, the current XML context node being processed, what are the local and global variables, set breakpoints in both the XML source file and the XSLT program.

The author of the *XSLT-process* package is **Ovidiu Predescu**, and the package is distributed under the terms of **GNU General Public License**. The project is graciously hosted by SourceForge, and could be found at <http://xslt-process.sourceforge.net/index.php>. The current version is 2.1.

2 Installation and setup

The *XSLT-process* mode is part of XEmacs, but it works with both XEmacs and GNU Emacs. The installation differs slightly between the two editors as we will see shortly.

The mode was tested on Linux and Windows 2000, with XEmacs and GNU Emacs. The development platform however is XEmacs under Linux, so expect this to be the most stable one.

2.1 Package installation under XEmacs

XSLT-process is also released as an XEmacs package, so if you're an XEmacs user, chances are that the package is already installed with your XEmacs installation. If it's not installed, try obtaining the latest XEmacs package from your XEmacs local mirror or from the main XEmacs ftp site. It may happen that the XEmacs package is a little older than the currently released version of *XSLT-process*, in which case you want to follow the steps described below.

You can retrieve the XEmacs package either manually from the ftp site, or by using the “Options” ⇒ “Manage packages” menu. Follow the instructions described under the “Options” ⇒ “Manage packages” ⇒ “Help” menu entry if you're not familiar with this procedure.

2.2 Generic installation

This section describes how to install *XSLT-process* on your Emacs editor, assuming either it is not already installed, or you want to install a new version.

The *XSLT-process* package depends on two other packages:

- [elib 1.0](#)
- [speedbar](#)

2.2.1 Installing Elib

If you're running XEmacs, you don't need to install *Elib*, as XEmacs by default comes with it.

Elib provides some useful functionality to Emacs, among other things a compatibility layer between GNU Emacs and XEmacs.

To install, download it from <ftp://ftp.lysator.liu.se/pub/emacs/elib-1.0.tar.gz>, and unpack it into your `~/emacs` directory. If you don't have a `~/emacs` directory, create one now.

Go to the `elib-1.0` directory and run

```
make
```

2.2.2 Installing Speedbar

XSLT-process depends on *Speedbar*, for displaying breakpoints, stack frames and global and local variables. You need to obtain and install this package first, in case you don't have it already installed.

To verify if you have this package installed, type `M-x speedbar-version`; this will give either an error, if the package is not installed, or the version number of the *speedbar* package if it's already installed. You should have at least version 0.13a for *XSLT-process* to work.

In case you don't have the *speedbar* package, you can obtain it from its [Web site](#). Unpack the package in your `~/emacs` directory. Then go to the *speedbar* directory and run

```
$ make
```

If you're using XEmacs, you need to run:

```
$ make EMACS=xemacs
```

This step byte-compiles all the Lisp files in the *speedbar* package using your Emacs editor. Please follow Speedbar's installation document for more up-to-date information on how to compile it.

2.2.3 Installing *XSLT-process*

To install the *XSLT-process* mode, first obtain the latest stable version from the SourceForge Web site, at <http://sourceforge.net/projects/xslt-process/>. Unpack the distribution in your `~/emacs` directory; this will create the *xslt-process-2.1* directory in your `~/emacs` directory.

You now need to tell Emacs to look for the new package in this newly created directory. You do this by adding the following lines in the Emacs configuration file `~/.emacs` (if you don't have such a file, create one now):

```
(mapc (lambda (x)
  (pushnew (expand-file-name x) load-path))
  (list "~/emacs"
        "~/emacs/xslt-process-2.1/lisp"
        "~/emacs/elib-1.0"
        "~/emacs/speedbar-0.13a"))
```

You can remove the reference to *elib-1.0* or *speedbar-0.13* in the lines above, if you didn't have to install either *elib* or *speedbar* as described in the previous sections.

2.3 Setting up the *XSLT-process* mode

XSLT-process is an Emacs minor mode, which means is extending the functionality of whatever mode you use for editing XML or XSLT files, instead of creating a similar one.

XSLT-process does not try to help in the editing of XML or XSLT files, it just enables Emacs to process such files. Thus *XSLT-process* should work with any XML/XSLT editing mode your Emacs is configured for.

XSLT-process was tested with both Lennart Staflin's [PSGML major mode](#) and James Clark's *sgml-mode.el* major mode (distributed with GNU Emacs) for editing XML files. It also works with Tony Graham's [xslide](#) XSLT editing mode.

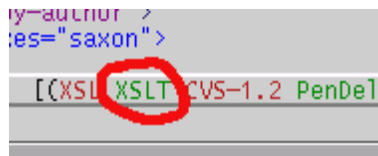
To automatically enable the *XSLT-process* minor mode for the above major modes, add the following lines in your `~/.emacs` file:

```
(autoload 'xslt-process-mode "xslt-process" "Emacs XSLT processing" t)
(autoload 'xslt-process-install-docbook "xslt-process"
  "Register the DocBook package with XSLT-process" t)
(add-hook 'sgml-mode-hook 'xslt-process-mode)
(add-hook 'xml-mode-hook 'xslt-process-mode)
(add-hook 'xsl-mode-hook 'xslt-process-mode)

(defadvice xml-mode (after run-xml-mode-hooks act)
  "Invoke 'xml-mode-hook' hooks in the XML mode."
  (run-hooks 'xml-mode-hook))
```

You can also manually switch to this minor mode by typing `M-x xslt-process-mode`.

You can check the modeline to see whether Emacs is in the *XSLT-process* mode. Look for the “XSLT” indicator in the modeline.

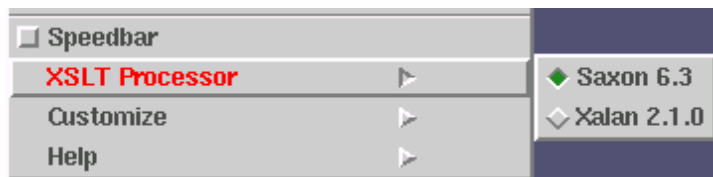


3 Running the XSLT processor

3.1 Setting up the XSLT processor

The *XSLT-process* mode comes by default with two different Java XSLT processors, Saxon and Xalan. These particular versions of the XSLT processors were tested and work with the *XSLT-process* mode. Different versions of the processors may not work with the *XSLT-process* mode.

You can choose either of the above processors to do the XSLT processing by selecting one from the “XSLT” ⇒ “XSLT Processor” menu. The default XSLT processor is Saxon.



3.2 Viewing the results of the XSLT processing

The main functionality of the *XSLT-process* mode is to allow you to edit an XML document, apply an XSLT stylesheet on the document, and view the results either in a buffer or in Web browser.



To run the XSLT processor and view the results in another Emacs buffer, you can enter `C-c C-x v`, while editing the XML document.

If your stylesheet generates HTML as the result, you can view the results directly in a Web browser by typing `C-c C-x n`. If your stylesheet generates XML FO as output, *XSLT-process* can translate it to PDF automatically using the [Apache FOP processor](#). In this case just type `C-c C-x p` and *XSLT-process* will take care of applying the XSLT stylesheet on your input XML document, and applying the FOP processor on the resulting XML FO document.

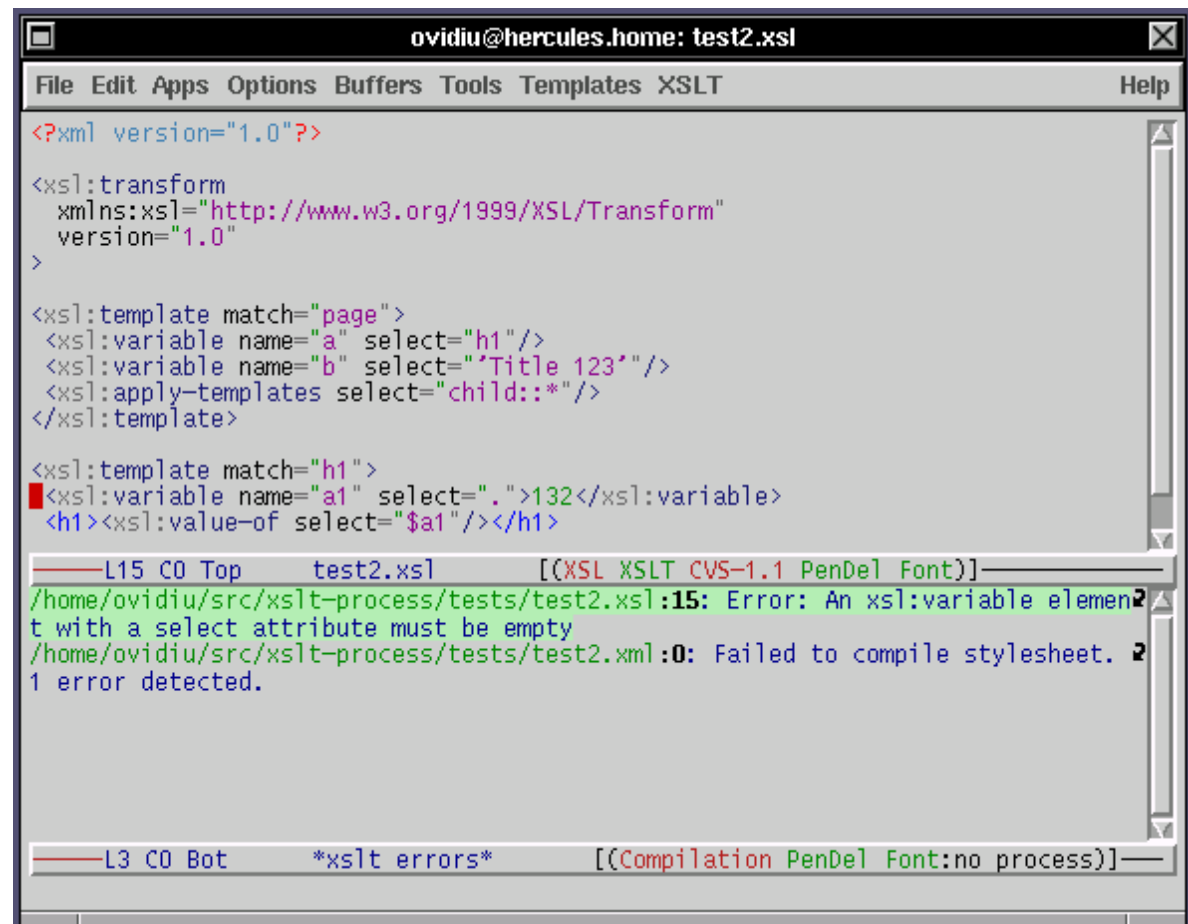
You can customize the Web browser and PDF viewer to be used as described in [Chapter 5 \[Customizing the XSLT-process mode\]](#), page 15. If you don’t like the default key bindings described above, you can also customize them as described in the same chapter.

If you choose to view the results in a buffer, they will be shown in the `*xslt results*` buffer. If any message are generated during the XSLT processing, they are show separately in the `*xslt messages*`.

3.3 Dealing with errors

The *XSLT-process* mode intercepts the error messages generated by the XML parser or XSLT processor and displays them in compilation buffer, which quickly allows you to go to the cause of the error.

When you encounter an error, just click using the middle-button (assuming an X-Windows system) on the error message. Emacs will move the cursor point at the place that caused the error.



3.4 Associating an XSLT stylesheet with an XML document

There are two ways to specify a stylesheet to be used to process an XML file. The first method uses the so-called ‘associated stylesheet’, a XML specific feature, which is specified inside the XML file. The second method is external to the XML, and is specific to the *XSLT-process* mode.

3.4.1 Using the associated stylesheet

The XSLT file that’s used to process the file should be specified inside the XML file using the XML processing instruction ‘xml-stylesheet’, like this:


```
<?xml version="1.0"?>
<?xml-stylesheet href="URL/to/XSLT/file" type="text/xsl"?>
```

...

You can use any URI understood by your Java system, e.g. file, HTTP or FTP, to refer to a stylesheet. The XSLT engine will download the stylesheet from the remote location and make use of it locally. The XSLT processing code of *XSLT-process* is written such that the stylesheet is cached, so further invocations won't parse the stylesheet again, unless the stylesheet is modified.

You can use URLs which are relative to the location of your XML file. For example if you have an XSLT file 'page-html.xsl' in the same directory as the XML file you can simply have inside the XML file the following processing instruction:

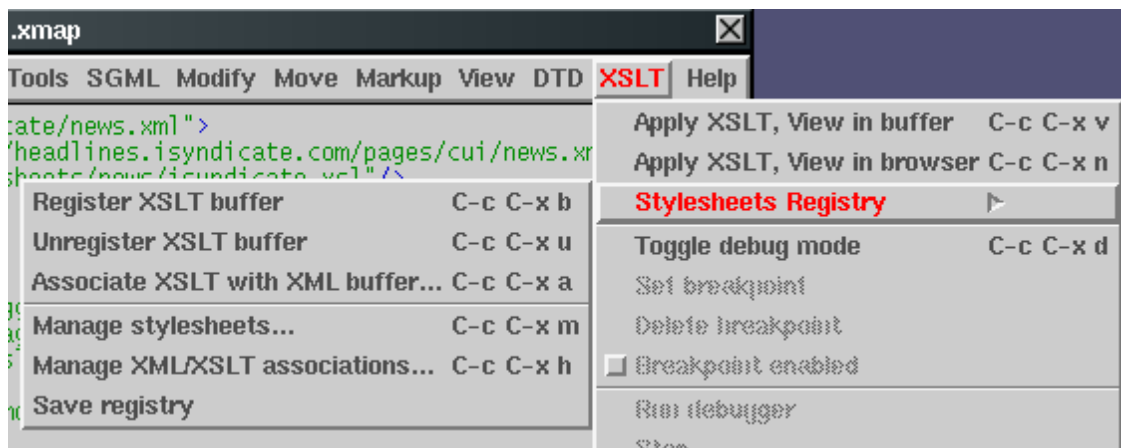
```
<?xml version="1.0"?>
<?xml-stylesheet href="page-html.xsl" type="text/xsl"?>
```

...

3.4.2 The stylesheet registry

Using the associated stylesheet in the XML document requires the XML document to have a `xml-processing` pseudo-instruction. This is not always convenient, so *XSLT-process* provides a way to associate XSLT stylesheets with XML documents.

The stylesheet registry functionality can be found in the menu under "XSLT" ⇒ "Stylesheet Registry".



Associating an XSLT stylesheet with an XML document requires the following steps:

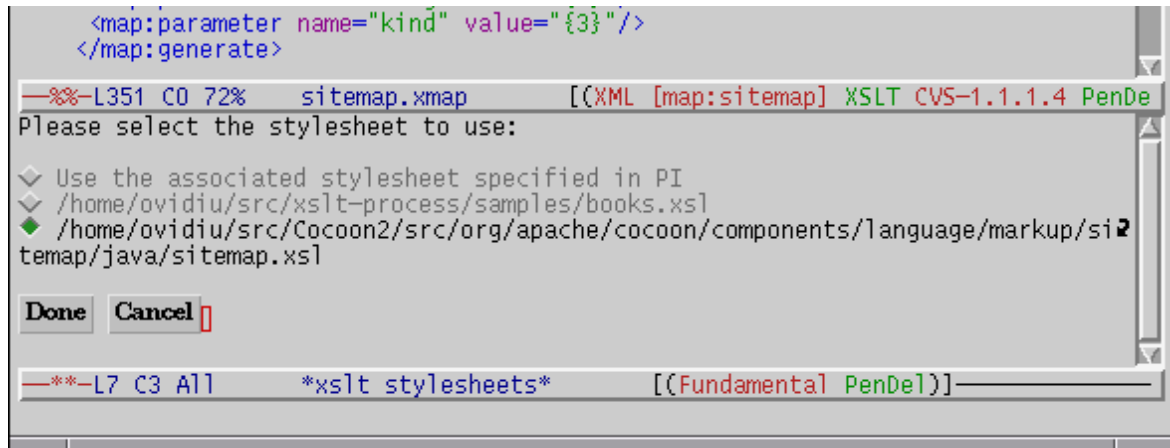
- Registering the XSLT stylesheet into the stylesheet registry. Only stylesheets registered with the registry can be associated with an XML document.

To register a stylesheet in the registry, type `C-c C-x b`. This will prompt you for the buffer name visiting the XSLT stylesheet. Select the buffer visiting the XSLT stylesheet you want; you can type `TAB TAB` for a list of buffers.

- Once you have registered stylesheets with the registry, you can associate XML documents with stylesheets registered with the registry. Select the buffer visiting the XML

document you want and type `C-c C-x a`. You will be prompted in a buffer to choose which currently registered XSLT stylesheet you want to be associated with the current XML document.

If you already have an association between the XML document and an XSLT stylesheet, it will be selected. If there is no association already established, the default is setup to use the associated stylesheet specified with the `xml-stylesheet` pseudo-instruction. There is no check being done to ensure you have such an instruction.



To remove or add new XSLT stylesheets, you can type `C-c C-x m`. This will present you a buffer that allows you to remove existing stylesheets, or to manually add new ones. Adding new ones may be easier though by visiting the file in a buffer, and typing `C-c C-x b`.

Managing the associations between XML documents and XSLT stylesheets is done in a similar way. Just type `C-c C-x h`, and you'll get a buffer that shows you all the currently registered associations.

If you want to persist the registry across multiple invocations of Emacs, you need to save it. You can do this from "XSLT" \Rightarrow "Stylesheet Registry" \Rightarrow "Save registry".

3.5 Changing the XSLT processor

If you want to experiment what are the results of applying your stylesheets using different XSLT processors, you can change the processor using the menu "XSLT" \Rightarrow "XSLT Processor". If you are in the middle of a debugging session, this action will not have effect until the session is finished.

In certain cases, there are XSLT stylesheets that works only when processed through a particular XSLT processor. This may happen because the stylesheet is using XSLT extensions specific to a particular processor. In this case you can specify the XSLT processor to be used in the XML source document (see [Chapter 6 \[Known problems and limitations\]](#), [page 16](#)).

Just add a *Local Variables* section at the end of your XML file and specify within it what should be the XSLT processor to be invoked using the 'processor' variable. For example,

by adding the following section at the end of your file, you specify *Saxon* to be used as the XSLT processor, no matter what is the global XSLT processor setting:

```
<!--  
Local Variables:  
processor: Saxon  
End:  
-->
```

In this release, the acceptable values for ‘**processor**’ are ‘**Saxon**’ and ‘**Xalan**’, as they are the only supported processors. By replacing the value of ‘**processor**’, you can run any of the supported processors on your file.

4 The XSLT debugger

For relatively simple XSLT stylesheets, understanding how a particular stylesheet works can be done by repeatedly running the XSLT processor, and looking at the results. More complex stylesheets however, are difficult to debug this way, especially when you have lots of XSL variables and key indices.

XSLT-process gives you the ability to run the processor in debugging mode, which allows you to set breakpoints, run step by step, view local and global XSLT variables.

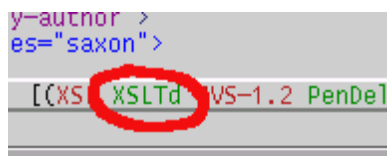
Note: This release supports debugging only with the Saxon XSLT processor. Xalan requires line number support for the XML source document; I have a patch for the 2.1.0 release, but I'm told the Stree tree model this release uses is going away in the next release. So I started to work on a patch for the new release which uses the so-called DTM tree model.

4.1 Entering the debug mode

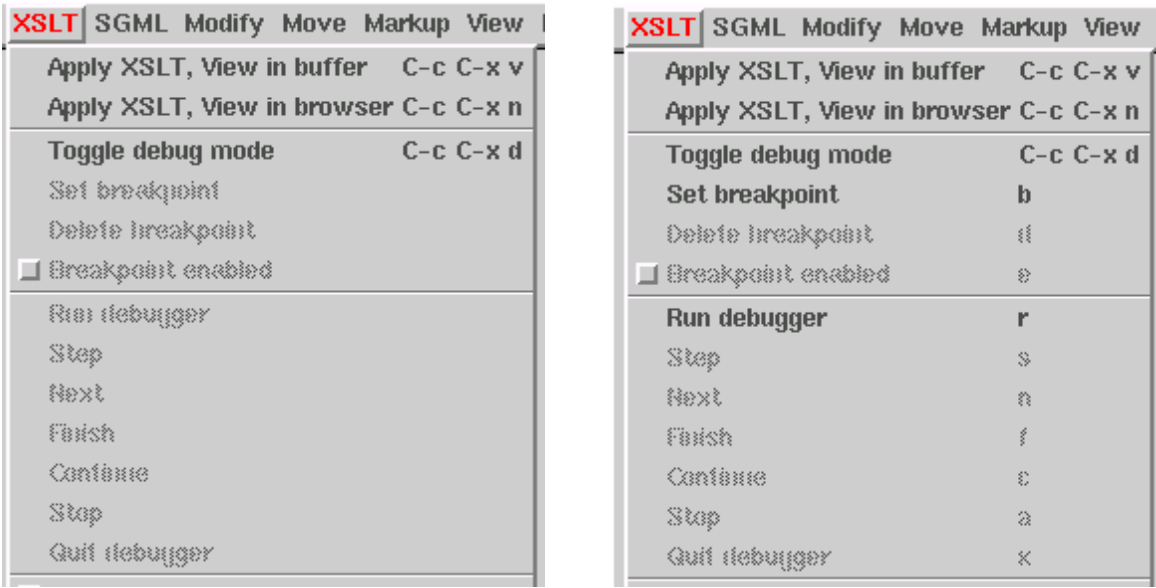
While you're editing an XML or XSLT file, the buffer visiting the file is read-write, allowing you to perform the normal editing operations, and use the functionality specific to the XML/XSLT/SGML mode you're using.

The XSLT debugging mode however does not allow you to modify files, and it binds some of the keys to debugger specific functionality. The XSLT debugging functionality is a per buffer feature, you can enable and disable it for each buffer independently.

To toggle the XSLT debugging functionality, type `C-c C-x d` while you're in a *XSLT-process* mode buffer. You can also toggle the debug mode through the menu at "XSLT" ⇒ "Toggle debug mode". Once you enter the debug mode, you will notice the modeline indicator changes from "XSLT" to "XSLTd".



In the menu, you will observe that until the debug functionality is not enabled, all the debugger related operations in the menu are disabled. Once you enable the debug mode, you will be able to setup breakpoints and run the XSLT debugger.



No debug menu: the debugging functionality is not enabled. XSLT debug menu: debugging functionality enabled.

4.2 Breakpoints

Once you’re in the XSLT debugging mode, you can set or delete, enable or disable breakpoints. During the execution of the XSLT stylesheet, the XSLT processor will stop when it reaches a breakpoint that is enabled.

The keybindings for breakpoints are:

- **b**: Set a new breakpoint at the line containing point. Doesn’t do anything if a breakpoint is already setup at this location.

The corresponding menu item is disabled if the point is on a line where a breakpoint is already set.

- **d**: Delete the breakpoint at the line containing point. Nothing happens if no breakpoint is setup on the line at point.

The corresponding menu item is disabled if the point is on a line where there is no breakpoint.

- **e**: Enable or disable the breakpoint at the line containing point. If there is no breakpoint setup at point, nothing happens. If the breakpoint is enabled, this action will disable it. Similarly, if the breakpoint is disabled, this action will enable it.

The corresponding menu item is disabled if the point is on a line where there is no breakpoint.

The enabled and disabled breakpoints are shown with distinctive colors, so you can easily identify them.

You can setup breakpoints both in the source XML document, and in the XSLT stylesheet program. Note however that the semantic of a breakpoint in the source document is very different from a breakpoint in the XSLT stylesheet. The XSLT processor will stop at a breakpoint in the source document only when an element on that line is *selected*, e.g. when the element becomes the current context node (see the **XSLT**

specification). Compare this with the XSLT stylesheet, which is essentially a program, and where the stop at a breakpoint means that the execution of the program reached the instruction at that line.

4.3 Controlling the debugger

After you setup the breakpoints where you would like the XSLT processor to stop, you can start the XSLT processing. You can do this from the menu at “XSLT” \Rightarrow “Run debugger”, or by typing `r`.

Important: When starting the debugger, the buffer containing the XML source document should be current. If a buffer containing an XSLT document is active instead, the *XSLT-process* mode assumes this as the source document, and will not be able to find the associated stylesheet.

The XSLT processor will start and it will continue running in the background until the execution reaches an enabled breakpoint.

In this moment, Emacs will highlight the line where the XSLT processor has stopped, and it will wait for an action to be taken. The possible actions are:

- *Step* (keybinding `s`): If the XSLT processor stopped when it was about to enter an XML element, it will enter it and stop the execution on the first child element, right before entering it.

If the XSLT processor stopped when it was exiting from an XML element, this action will make the processor exit from the current element, and stop on the immediately following element.

This is equivalent with the *step into* action of debuggers for traditional programming languages.

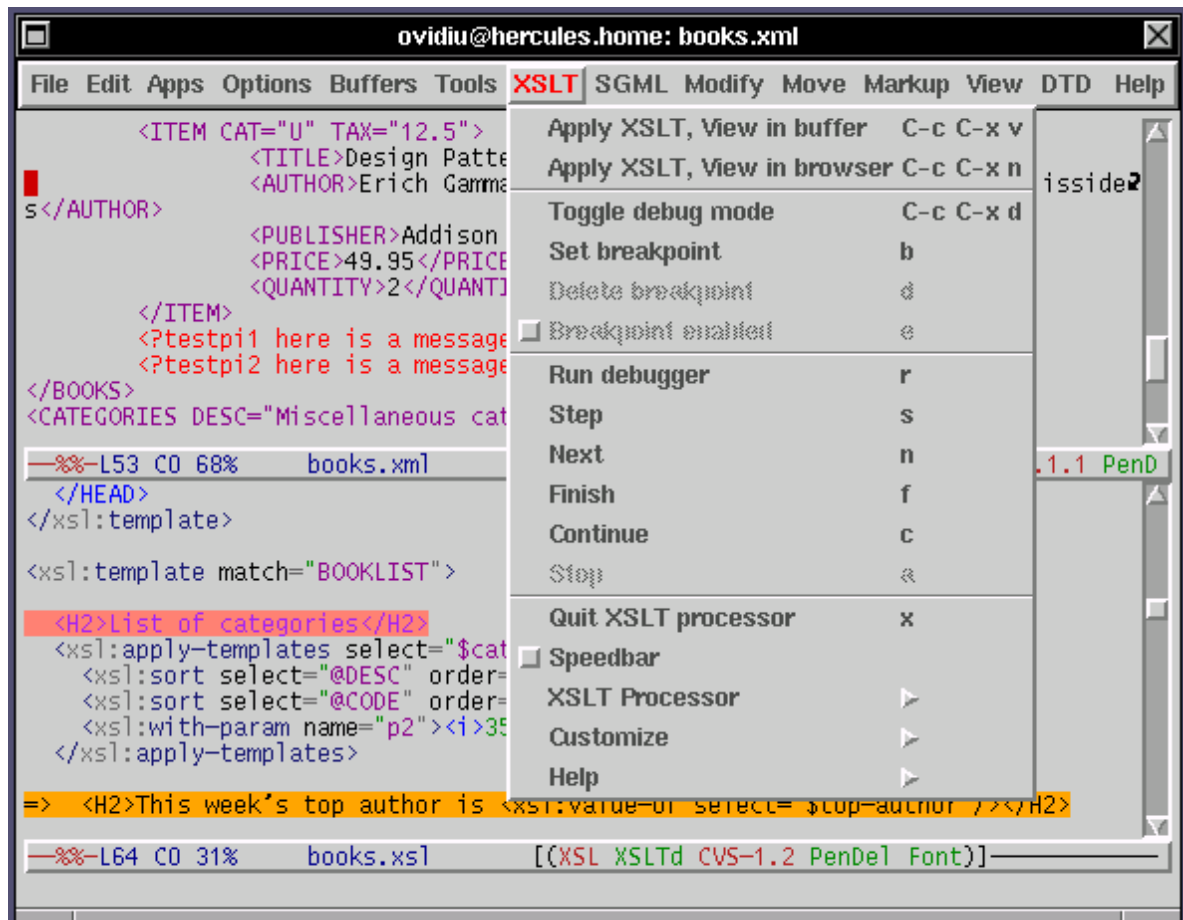
- *Next* (keybinding `n`): If the XSLT processor was about to enter an XML element, this action will instruct the processor to skip the element (this includes on of its sub-elements). The processor will stop when it is about to visit the next sibling XML element. If there is no such element, the processor will stop when entering the next XML element which is to be visited.

This is equivalent with the *step over* action of debuggers for traditional languages.

- *Finish* (keybinding `f`): Instructs the XSLT processor to finish the currently executing XSL template, and stop when exiting from it. If an enabled breakpoint is reached before the end of the template, the XSLT processor will stop there.

This is equivalent with the *finish* action of traditional debuggers.

- *Continue* (keybinding `c`): Instructs the XSLT processor to continue the processing until the next enabled breakpoint is encountered or until the XSLT processing finishes.
- *Stop* (keybinding `a`): In the case of a long processing, instruct the XSLT processor to abort it and stop immediately. This is useful for stylesheets that take a long time to complete, and you forgot to setup the appropriate breakpoints.



4.4 The speedbar

The speedbar is a separate Emacs window that allows you to:

- View the breakpoints that you've set. The enabled and disabled breakpoints are shown with distinctive colors, so you can easily identify them.

Clicking on a breakpoint entry in the speedbar will move the point to the file and line where the breakpoint is set. If the file is not currently opened within Emacs, it will open and shown in a buffer.

- View the stack of source elements that have been selected during the processing. Whenever you click with the middle button on such a entry, the point is positioned in the file and line where the source element starts.







- View the stack of XSLT elements that are currently being executed. Each element that has been entered, but not yet exited is being shown in the speedbar.

Clicking with the middle button on such a node will position the point in the file at line where the element starts.

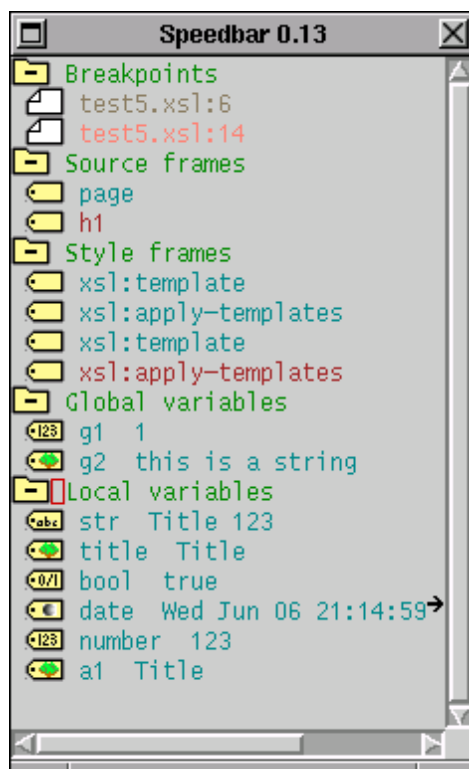
- View the local and global variables declared with `xsl:variable` or `xsl:param`.

Each entry for a variable has a little icon showing the type of the variable, its name and value. If images are not supported by your Emacs (GNU Emacs 20.x and lower), a text indication of the type is shown instead.

Below are all the possible XSLT types and how they are displayed by speedbar:

-  **boolean** - The text representation of an XSLT boolean is {b}.
-  **number** - The text representation of an XSLT number is {n}.
-  **string** - The text representation of an XSLT string is {s}.
-  **nodeset or document fragment** - The text representation of this type is {t}.
-  **object** - The text representation of Java object is {o}.
-  **any** - This is an unknown XSLT type; its text representation is {a}.

Here is a picture of the speedbar in the *XSLT-process* mode:



5 Customizing the *XSLT-process* mode

You can customize the *XSLT-process* mode in a number of ways. There are several groups of customizations you can do:

- Customize the environment for the Java process that's running the XSLT processor.
- Customize the faces (the fonts) used when displaying the breakpoints, the current line etc.
- Customize the key bindings used for invoking the processor and for the debugger.
- Customize the Web browser and PDF viewer to be used to directly view the results of the processing.

You can find all the above customization options under the main menu, in “XSLT” ⇒ “Customize”. Alternatively, you can run `M-x customize-group RET xslt-process` to get to the main customization page for *XSLT-process*.

6 Known problems and limitations

The following is a list of differences between GNU Emacs and XEmacs. As the *XSLT-process* mode is developed under XEmacs, I would appreciate your support in making the GNU Emacs version behave closer to the XEmacs one.

- Under GNU Emacs, the keybinding of a command does not appear in the menu item as they appear under XEmacs.
- The *Local Variables* section that specifies which XSLT processor to be used has to be placed in the XML source document, instead of the XSLT stylesheet which processes the XML document.

This will change when *XSLT-process* has support for registering XSLT stylesheets with it, instead of having them associated with the XML source document through a ‘`xml-stylesheet`’ processing instruction (see [Chapter 7 \[Future enhancements\]](#), [page 17](#)).

- Under certain circumstances it is possible that the *XSLT-process* mode will do an XSLT processing and print the message ‘Done invoking Saxon; no output generated.’, yet there is a result buffer. This is a known problem, that will be fixed in a future release.

7 Future enhancements

This is a list of features planned for future releases. The list doesn't specify any particular order in the priority, and the features may or may not be implemented.

- Ability to pass parameters to the XSLT processor. This could be done at multiple levels:
 - *globally*: these are parameters that are specified to all the stylesheets, in addition to any other parameters. They could be specified through the customization menu.
 - *per XSLT stylesheet*: these are parameters specified to a given XSLT stylesheet, and they are passed only to this stylesheet.
 - *per XML document*: these are parameters specified for a given XML document.

The per XML document or XSLT stylesheet parameters can be specified using the *Local Variables* section in file, the same way as the XSLT processor could be temporarily changed per file.

- Add ability to evaluate XPath expressions. The context node could be either the top node of the XML document or it can be specified by putting the point on it.

This would be a very useful tool for novice XSLT users to learn XPath (and Emacs ;-).

- Add the ability to process and debug JSPs and Cocoon's XSP. This would be very useful and would substantially improve the overall edit-run-debug cycle for this type of applications.
- Add the ability to go both forward and backwards in time during debugging. This would be a nice feature to have for the cases where reaching the point of interest in the stylesheet takes a lot of time, and the pressing the wrong key makes you miss it.
- Integrate the XSLT debugger with the JDE debugger, to give the ability to debug extension functions. This should be possible, but is a lot more complex as it requires the Java debugger to attach to the XSLT processor, and do some magic to get to the right context.

8 Changes

These are the changes since the 2.0 release:

- Added the ability to run Apache's FOP processor on the XML document created as a result of the XSLT processing. Suggestion from [Jeff Rancier](#).
- Added the ability to pass JVM arguments and properties. Thanks to [Phillip Lord](#) for the idea and initial patch.

These are the changes since the 1.2.2 release:

- The functionality has been greatly improved, support for debugging and error reporting has been added.
- The code has been completely overhauled. A new dedicated Java command line tool replaces the more generic BSH process. This command line tool is the low level interface used by the *XSLT-process* to perform its functionality, and interacts directly with the XSLT processors.
- The project has been moved to Sourceforge, including CVS source tree and previous releases of the code, and is open to public participation. Please register on the public mailing list to participate in discussions related to *XSLT-process*.

These are the changes since the 1.2 release:

- Fixed problem in accessing stylesheets referred by *file:* URIs on Windows. Reported by [Nicolas Kessler](#).

This is the list of changes since the 1.1 release.

- Added support for the TrAX interface, part of the JAXP API, thanks to [Allan Erskine](#). Currently Saxon 6.2 and Xalan2 have been tested. The TrAX interface caches the XSLT stylesheets in the processor driver in a compiled form, so the speed of the processing is increased.
- The mode is now running with GNU Emacs on Windows NT/2000, thanks to [Allan Erskine](#) for figuring out the issues.
- Changed again the keyboard binding to *C-c C-x C-v*, as *C-M-x* doesn't work on Windows systems.
- The documentation has been reorganized a little bit to be more logical.

This is the list of changes since the 1.0 release.

- The '*xslt-process-additional-classpath*' customization variable has been introduced. Setup this variable with any additional Java classpath components you want to be passed to the BeanShell when is first invoked. If you already started a BeanShell, you need to kill the corresponding buffer (named '**bsh**') and restart it by invoking *XSLT-process* on a buffer. (Suggestion from [T. V. Raman](#).)
- Allow for passing the user agent to the Cocoon processor so that multiple browser types can be simulated. This works with a patch I submitted against Cocoon 1.8-dev; it was

incorporated and should be available in the 1.8.1 release. If you need the patch before this release, feel free to contact me, I'll gladly send it to you.

- The way the error messages are displayed has changed, now error messages encountered during the JVM startup process also go in the `'*xslt-output*' buffer`.
- The default keybinding has been changed to *C-M-x* instead of *C-c x*, to conform to the (X)Emacs keybinding standards.

Table of Contents

1	What is it?	1
2	Installation and setup	2
2.1	Package installation under XEmacs	2
2.2	Generic installation	2
2.2.1	Installing Elib	2
2.2.2	Installing Speedbar	3
2.2.3	Installing <i>XSLT-process</i>	3
2.3	Setting up the <i>XSLT-process</i> mode	3
3	Running the XSLT processor	5
3.1	Setting up the XSLT processor	5
3.2	Viewing the results of the XSLT processing	5
3.3	Dealing with errors	6
3.4	Associating an XSLT stylesheet with an XML document ...	6
3.4.1	Using the associated stylesheet	6
3.4.2	The stylesheet registry	7
3.5	Changing the XSLT processor	8
4	The XSLT debugger	10
4.1	Entering the debug mode	10
4.2	Breakpoints	11
4.3	Controlling the debugger	12
4.4	The speedbar	13
5	Customizing the <i>XSLT-process</i> mode	15
6	Known problems and limitations	16
7	Future enhancements	17
8	Changes	18