XSLT-Debugger

Table of Contents

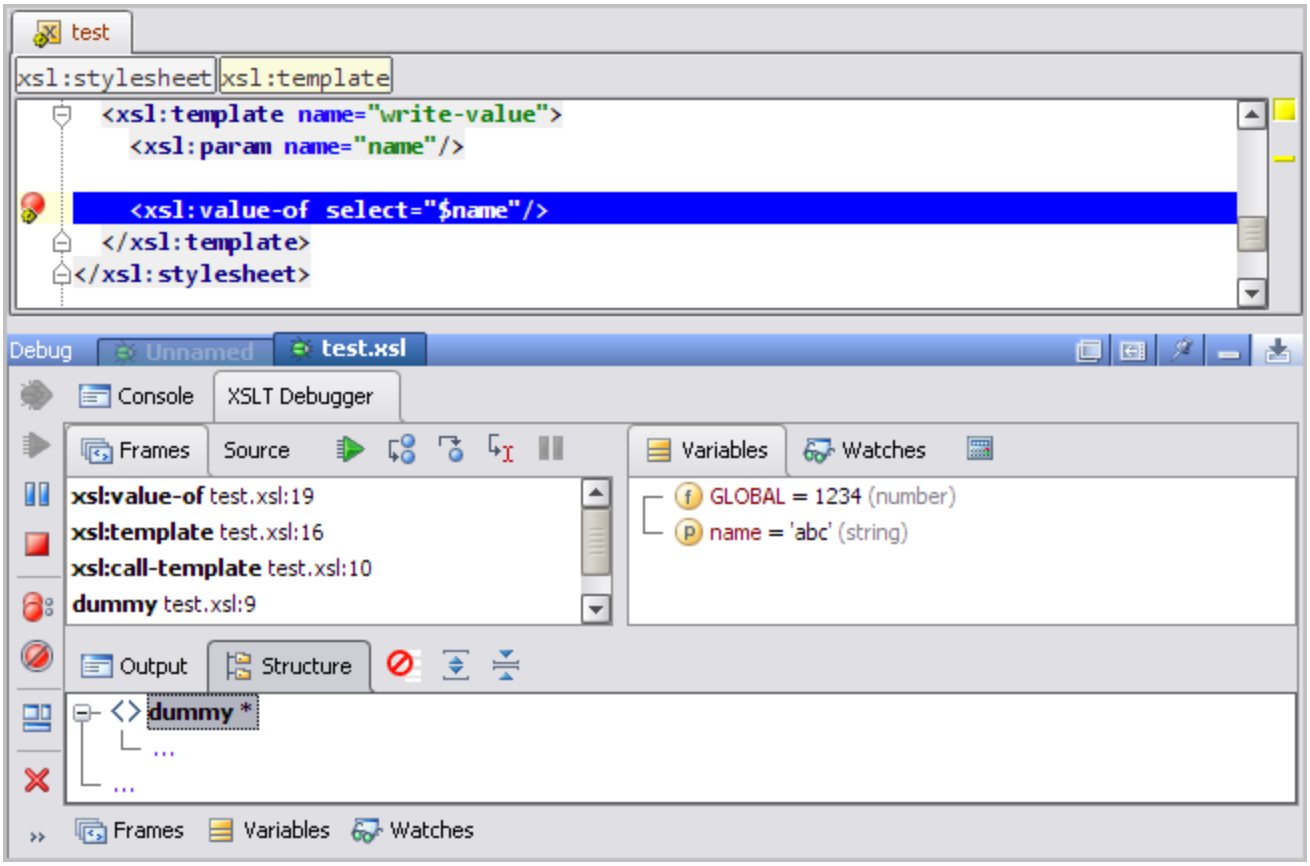
Description

The XSLT-Debugger plugin allows to debug XSLT stylesheets in IntelliJ IDEA with various useful features a debugger should have, including breakpoints, watch expressions, variables display, etc. It is tightly integrated with IDEA which allows it to be efficiently used without much need to familiarize first.

> ⚠ Requirement
> The plugin requires at least version 3 of the "XPathView + XSLT-Support" plugin. If this isn't installed yet, IDEA will automatically download and install it when installing the XSLT-Debugger plugin. If an earlier version is already installed, please upgrade the plugin manually.

Features

The following sections contain a list of the major features that are provided by the XSLT-Debugger that help to develop and debug XSLT stylesheets in IntelliJ IDEA with pleasure.



Breakpoints

Breakpoints are the major feature of any debugger, and so they are for the XSLT-Debugger. Breakpoints can be set in any XSLT file with the same keyboard shortcut to set breakpoints in Java classes, by clicking on the left editor gutter, or by invoking the action "`Toggle Line Breakpoint`" from the "`Run`" menu.

Breakpoints in XSLT files are decorated with a little "gear" icon and behave pretty much the same as breakpoints for Java classes or other languages (e.g. Groovy) that can be debugged in IDEA: They can be enabled and disabled, and it's also possible to configure various properties that control the behavior of a breakpoint.

Breakpoint Properties

Properties of XSLT breakpoints can be configured by either right-clicking the breakpoint-icon in the left editor gutter or by opening the breakpoints dialog via `Run | View Breakpoints` from the main menu. This allows to manage the following properties.

Breakpoint Conditions

Breakpoints can have a condition expression which is evaluated each time the breakpoint is hit. The result of this evaluation determines whether the breakpoint's action is executed, i.e. for a "simple" breakpoint, this would be simply to suspend the execution.

For more advanced debugging, it is also possible to specify either a log- or trace expression that is evaluated when the breakpoint is hit (and the condition, if present, evaluates to `true`).

Log Breakpoints

The result of a log-breakpoint will simply be written into the output console of the debugged process. This can be useful to output the value of a certain variable or maybe the name of the current context node without having to stop the debugger each time the breakpoint is visited.

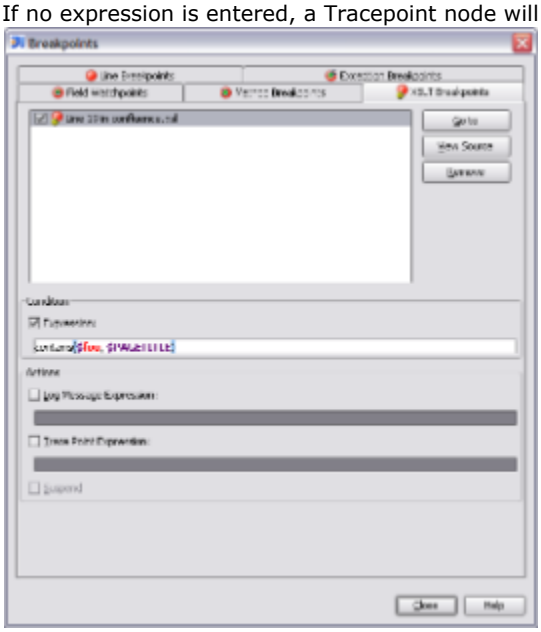
If no expression is entered, a standard message like "`[text.xsl:12]: <no message>`" will be displayed which includes the

name of the XSLT file and the line number in brackets.

Trace Breakpoints

An even more advanced debugging tool are trace-breakpoint. They are evaluated just like log-breakpoints, but their result isn't printed into the console. Instead, for each evaluated trace-expression, a special node in the "Structure" tree will be inserted. The contains the evaluation result and indicates the position during the execution of the stylesheet when the breakpoint was triggered.

If no expression is entered, a Tracepoint node will be generated that just the line number of the breakpoint that triggered it.

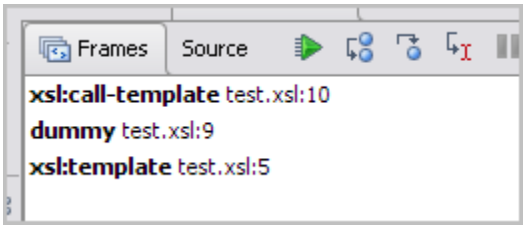


XPath Expressions
Each expression can use any XPath function that is supported by the XSLT processor and any variable that is visible at the breakpoint's location. You'll note that you don't have to type everything yourself: There's code completion and validation for functions and variables.

Suspend Policy
For both log- and trace-expressions it is possible to specify the "Suspend Policy" (as it is called by IDEA's Java Debugger): If the "Suspend" checkbox is checked, the execution of the stylesheet will also be paused when the breakpoint triggered.

XSLT Frames

When paused, the XSLT-Debugger displays the complete call-stack for the current location. The "Frames" tab shows the stack of XSLT-instructions, while the "Source" tab shows the positions in the transformed source document, i.e. the stack of the context nodes.



Frame Navigation

When selecting a different frame in the "Frames" tab, the debugger will open the corresponding XSLT file and highlight the selected frame's position in that file. It will also update the "Variables" tab to show the visible variables and their values. Any expressions in the "Watches" tab will be recomputed for the changed context.

The source frames are displayed as a unique XPath location path together with the containing file's name and the line number. Navigating to these frames is possible as well, either by double-clicking an element or by pressing the "`Navigate to Source`" keyboard shortcut (usually F4). The debugger will then open the corresponding XML file and position the cursor at the selected node's position.

Execution Control

The XSLT-Debugger provides the usual ways to control the execution of a debugged stylesheet, such as "Step Into", "Step Over", "Run to Cursor".

Step Over

Steps over the current XSLT instruction until the next instruction on the same or a higher level is reached. For example, if the current instruction is an `xsl:call-template`, the called template will be executed and the debugger will stop at the next instruction after the call.

> Note that any XML element, including literal result elements, is an instruction in this case. That means, that if stepping over a literal result element, the execution is resumed until the element has been completely generated. Use "Step Into" to step through the generation of literal result elements.

Step Into

Steps into the current XSLT instruction.

Run To Cursor

Resumes execution of the debugged process until the current line is reached. This can be used as a kind of temporary breakpoint that is cleared once it is reached.

> If there are breakpoints set on any lines that will be executed before the specified line is reached, the debugger will pause at the first breakpoint.

Resume

Resumes the execution of the debugged process until another breakpoint is reached or the stylesheet is finished.

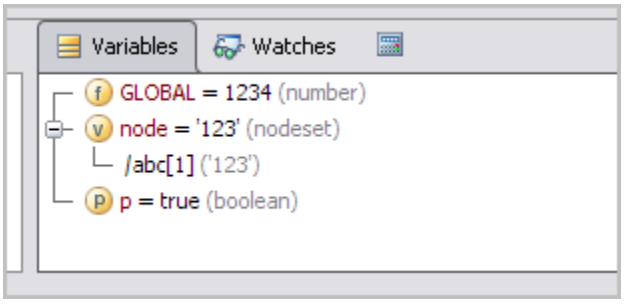
Pause

Tries to pause the debugged stylesheet at the next XSLT instruction.

Variables

When the debugged process is suspended at a breakpoint, the "`Variables`" tab will show all variables visible in the current scope. Global variables, local variables and parameters are distinguished by different icons.

The value of each variable is shown together with its runtime-type. If a variable holds a value of the type `nodeset`, the value can be expanded to show all the nodes contained in the set. Such nodes are displayed with an XPath location-path that uniquely identifies each node, together with the textual value of the node, i.e. the evaluated value of `string(.)`.



It's also possible to navigate to a variable's declaration, either from the variable node's context menu or by the keyboard

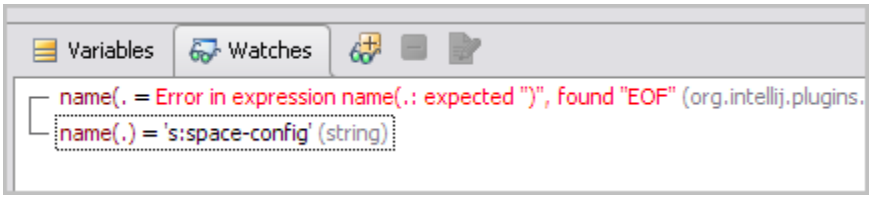
shortcut for "`Navigate to source`". The same is possible for each node in a nodeset which navigates to the source of the selected node in the containing XML file. If for some reason the location of a node cannot be determined, the Navigate to Source action is disabled.

To copy the string-value of a variable or node into the clipboard, press Ctrl-C or invoke the "`Copy Value`" action from the context menu.

Watches

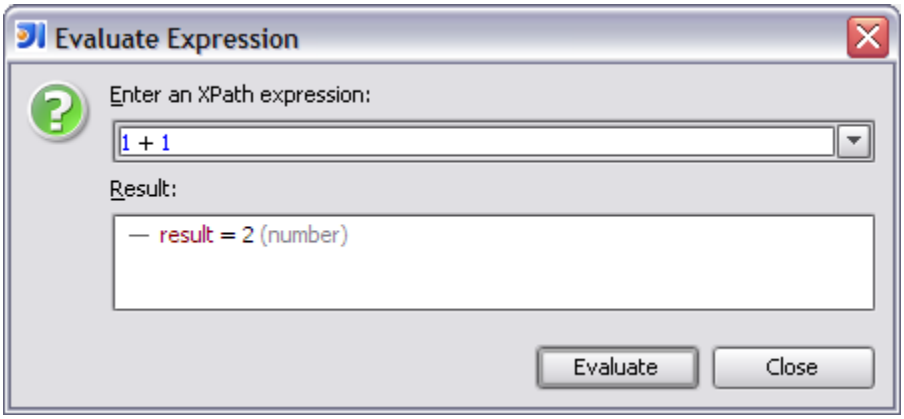
Watches can be used to automatically compute the value of certain expressions each time the debugger is paused. They will be evaluated in the context of the current stack frame and may reference the current context node as as any variables that are visible when the watch expression is evaluated. If the expression contains an error and cannot be evaluated, an error message will be displayed instead of the result.

The evaluated result of an expression is displayed just like the value of a variable. If the expression cannot be evaluated due to a syntax error or a reference to an undefined variable, this will be displayed with an error message.



Expression Evaluation

On the toolbar of the "`Variables`" tab there's an action that opens a dialog which allows to evaluate arbitrary expression. The evaluation works just like the evaluation of a watch expression.



XSLT Output

Text Console

The output that is generated by the transformation is displayed in two different ways: First, there's simple textual console window that displays the output (almost) as soon as it is generated. There may be some delay due to the buffering that is used to reduce the performance impact.

When the debugged process is finished, the output can be opened in an IDEA editor by clicking the "`Open in Editor`" action in the toolbar of the output console.

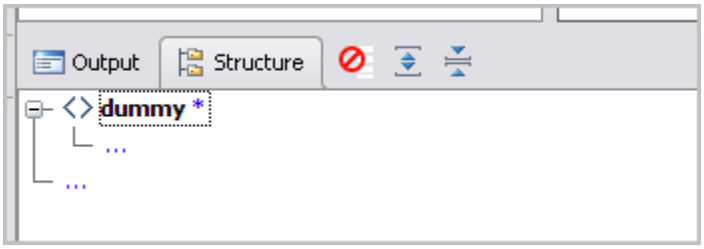
Generated Structure Tree

The second way is much more advanced and allows to track the generation of each XML node as it is produced by the XSLT processor. The generated nodes are displayed in a tree that reflects the XML structure. For each node, its kind (element, attribute, text, etc.), name and value (if applicable for the node type) are displayed.

The tree is updated each time the debugger is paused. Nodes that have been generated since the last time the debugger was suspended are marked with a blue "*" to indicate that it's a new node. Element nodes that aren't closed yet contain a node at the end with the text "..." to indicate that more content is expected.

To find out which XSLT instruction generated a specific node, right click the node and select "`Navigate to source`". This will open the XSLT file at the position that produced the node.

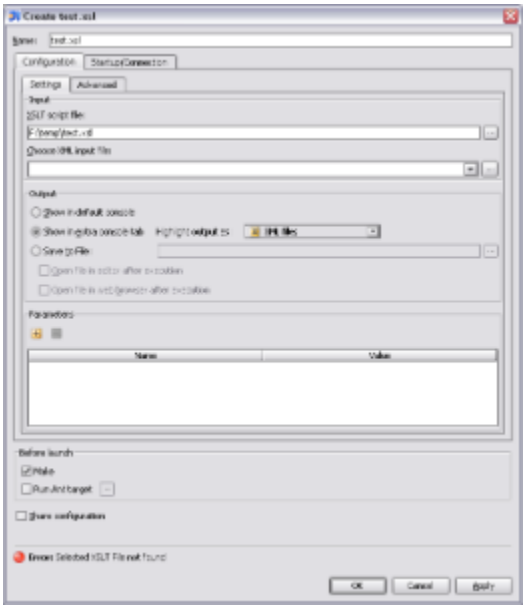
The toolbar of the "`Structure`" tab contains the following actions: The "`Hide Whitespace Nodes`" action will make the tree hide all text-nodes that only consist of whitespace. This can be useful to see only the really relevant nodes. The actions "`Expand All`" and "`Collapse All`" allow quickly expand and collapse the whole tree.



Usage Instructions

Using the debugger is fairly simple. Just follow these few simple steps to create a run configuration and start debugging XSLT stylesheets:

The first step is to create a Run Configuration for the stylesheet that should be debugged. This can be done either by creating a new configuration from scratch via "`Run | Edit Configurations`" action from the main menu, or by right-clicking an XSLT file an choosing "`Create 'xyz.xsl' ...`". This is also the place where the XML input file that should be transformed is selected.



While creating the run configuration, select "Show in extra console tab" as output option. This is important, otherwise the debugger won't start. Also make sure that the JDK to be used is 1.5 or higher.

You'll find more information about creating XSLT run configurations and their options in the online help of the XPathView + XSLT-Support plugin.

Next, set one or more breakpoints, either with your regular keyboard shortcut or by clicking on the left editor gutter. A breakpoint icon with a little gear will appear.

Optionally specify any properties of the breakpoint, such as an XPath-condition on which it should trigger.

Now the debugger can be started in the usual way by clicking the "Debug"-icon in the main toolbar or by invoking the action "`R`

un | Debug" from the main menu.

If the stylesheet is valid and can be loaded by the XSLT processor, the view in the "Debug" toolwindow will switch from the regular output console to the "XSLT Debugger" tab.

Once a breakpoint is hit, the corresponding file will be opened in the editor with the current execution-point being highlighted.

Now it's possible to examine the values of parameters and variables, check watch expressions or evaluate custom XPath expressions in the context of the current execution-frame. See the chapters Variables and Watches more information. To inspect the output that has been generated by the stylesheet, please see the chapter XSLT Output.

When you're done with inspecting the current position, you can either resume the execution continue debugging in single-stepping mode with the "Step Over" or "Step Into" actions. See the chapter Execution Control for more information.

Availability

The plugin is available for IntelliJ IDEA 7.0 and 8.1 via the built-in plugin manager and at the IntelliJ Plugin Repository.

Installation
The XSLT-Debugger requires version 3 or later of the XPathView + XSLT-Support plugin. If it's not installed yet, IDEA will usually automatically download and install it. If an older version is already installed, please upgrade the plugin manually.

Known Issues

Mute Breakpoints
The "Mute Breakpoints" action currently doesn't work for XSLT-breakpoints. This action does not affect XSLT Breakpoints, i.e. they will stay active.

XSLT Processor
The debugger engine currently uses the SAXON 6.5.5 XSLT processor, which might cause the debugged stylesheets to work different or even incorrectly when they're designed for a another processor (e.g. Xalan), especially when extension functions are involved.

Debugging with Xalan is possible as well if either a compatible version of Xalan is found in the classpath or the VM argument "-Dxslt.transformer.type=xalan" is added. Note that the Xalan-support is considered beta-quality.

To switch back to SAXON even if Xalan is present, use "-Dxslt.transformer.type=saxon" to override the auto-detection.

Platform Issues
The plugin has so far only been tested on Windows. Although there's nothing that generally prevents using it on other platforms, the breakpoint matching logic depends on file-URLs which may look differently on Linux/Mac. Even though some effort has been put into normalizing the URLs, I don't know if it actually works everywhere.

Please let me know if there are any problems - on any platform.

Version History

Version 1.0-beta (2007-12-03)

- Initial version

Version 1.1 (2007-12-13)

- Bugfixes, internal refactorings, stability improvements

Version 1.1.1 (2007-12-18)

- Online Help added

Version 1.1.2 (2007-12-21)

- Documentation updated

Version 1.1.6 (2009-01-16)

- Support for IDEA 8

Version 1.2.1 (2009-02-27)

- Beta-support for debugging with Xalan

Version 1.2.2 (2009-03-06)

- Bugfixes