EPIC - User's Guide

Table of Contents

Plug-in Installation 1
- Prerequisites 1
  - Eclipse 1
  - Perl 1
  - Considerations when using Cygwin 1
- Installation 1

Setting Up Preferences 3
- General Preferences 3
- Code Assist 3
- Editor 4
- Source Formatter 5
- Task Tags 6
- Templates 6
- Associating Files with the Perl Editor 7
- CVS Setup 8
- Setting Up the Debugger 9

Perl Projects 10
- Creating a Project 10
- Perl Include Path 10
- Converting an Existing Project 11

Eclipse Basics 12
- Perspectives 12
- Views 12
- Using Perspectives 12
  - New Perspectives 12
  - Configuring Perspectives 14
  - Saving a User Defined Perspective 15
  - Resetting Perspectives 15
- Using Views 16
  - Opening Views 16
  - Moving and Docking Views 16

Working with the Perl Editor 17
- Syntax Check 17
- Explain Errors and Warnings 17
- Open SUB Declaration 18
- Perldoc 19
- Quick Reference 20
- Code Assist 20
  - Variable Inspection 20
  - Module Inspection 21
- Task Markers 21
- Templates 22
  - Defining Templates 22
  - Using Templates 23
- Source Formatter 24
- Source Folding 24
- Refactoring 25
  - Extract Subroutine 25
- HTML Export 27
- Outline View 27

Using the Perl Debugger 28
- Launching Perl Programs 28
  - Launching Perl Programs in Run Mode 28
  - Re-launching a Perl Program 29

Creating Launch Configurations 30
Perl Local: Running a Perl Script on the Local Machine 30
Perl CGI: Run Perl Programs in a CGI Environment 31
Perl Remote: Debug a Perl Script on a Remote Machine 32
Breakpoints 34
Setting Breakpoints 34
Enabling or Disabling Breakpoints 34
Removing Breakpoints 34
Views in the Debug Perspective 35
Debug View 35
Variables View 35
Breakpoints View 36
Perl Expression View 37
Stepping Through the Execution of a Perl Program 37
Step Over 37
Step Into 38
Run to Return 38
RegExp Plug-in 39
Enabling the RegExp View 39
Using the RegExp Plug-in 39
Debugging Regular Expressions 39
Known Bugs & Problems 41
Debugger 41
Problems Showing Local Variables 41
References 42

Plug-in Installation

Prerequisites

Eclipse

Before installing the EPIC plug-in, Eclipse version 3.1 or higher has to be installed.

Eclipse comes in two flavors. The *SDK* version contains Java IDE components and is much larger than the *Platform* version. If you only want to use Eclipse as a Perl IDE, the Platform version is sufficient. If you are in for Perl and Java coding, use the SDK version.

Eclipse does not include a Java Runtime Environment (JRE). You will need a 1.4.1 level or higher Java Runtime or Java Development Kit (JDK) installed on your machine in order to run Eclipse.

Eclipse can be downloaded from www.eclipse.org [http://www.eclipse.org].

Perl

In order to have all EPIC features like Syntax Checking, Source Formatting etc., a Perl interpreter is needed. In principle any Perl interpreter can be used. To use debugging within Eclipse, Perl version 5.8.x or 5.6.x is required. For further requirements concerning the debugger, see the section called “Setting Up the Debugger”.

Most *nix/Linux installations will provide Perl interpreters out of the box.

Perl for Windows can be downloaded from www.activestate.com [http://www.activestate.com]

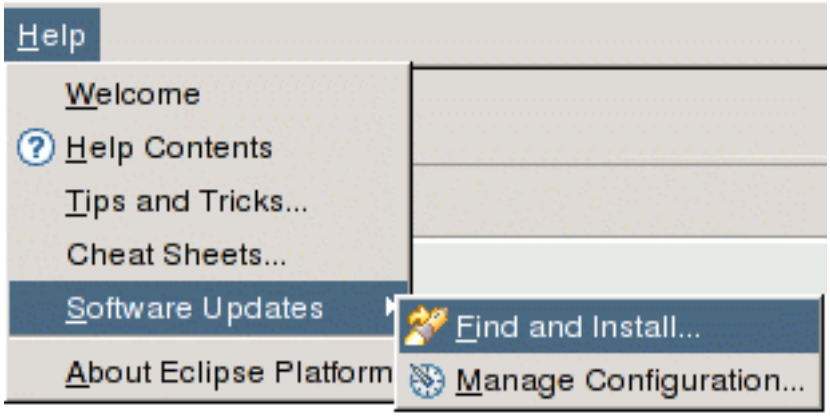
Considerations when using Cygwin

Make sure that the mount command is available and that it is in your system path. As mount is a standard component of Cygwin, you usually just have to add the cygwin\bin directory to your system path.

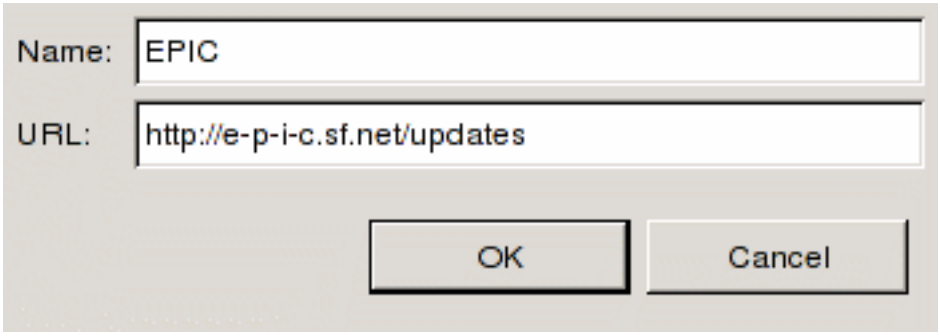
Installation

The installation is done by using the Eclipse Update Manager. The Update Manager connects to an EPIC Update Site [http://e-p-i-c.sourceforge.net/updates]. The Update Site can also be stored locally if no Internet connection is available and can be downloaded from the EPIC project page [http://e-p-i-c.sourceforge.net].

After starting Eclipse, select Help->Software and Updates->Find and Install... from the menu.



Select Search for new features to install an press Next. Press the Add Update Site... button for a remote installation via HTTP or the Add Local Site... button if the Update Site is available locally. When installing EPIC from remote, enter any desired Name and the URL **`http://e-p-i-c.sf.net/updates`**.



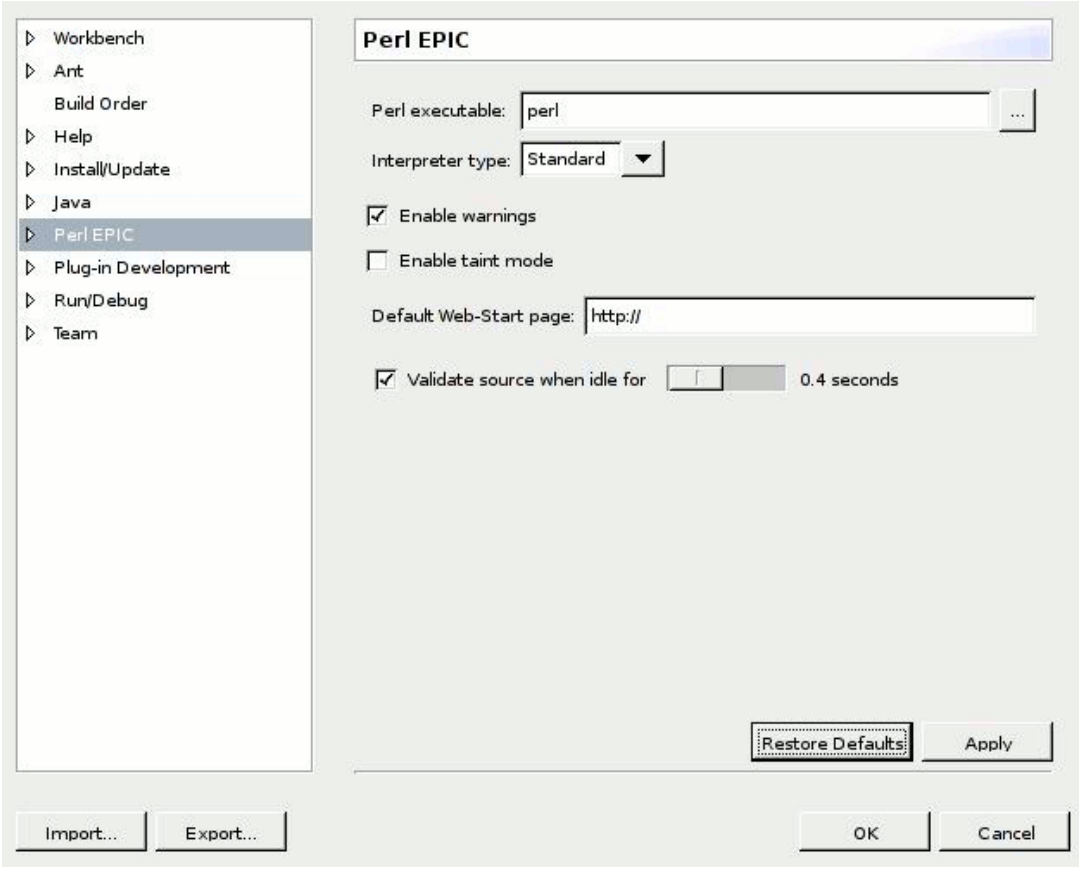
Tick the newly created site and press the Next button. Follow the instructions of the install wizard. The warning about the installation of an unsigned feature can be ignored. Eclipse has to be restarted after installation.

Now the EPIC installation should be complete.

Setting Up Preferences

EPIC preferences can be accessed via Window->Preferences... from the Eclipse Menu.

General Preferences



Click on Perl EPIC to open the General Preferences page.

General preferences include the location of the Perl interpreter, the option to enable warnings, taint mode and the interval of the source validation.

The validation interval indicates when to start validation after the editor becomes idle.

Apart from the standard interpreter type, the type can be switched to *Cygwin*. In this case the @INC path is mapped to be Cygwin compliant.

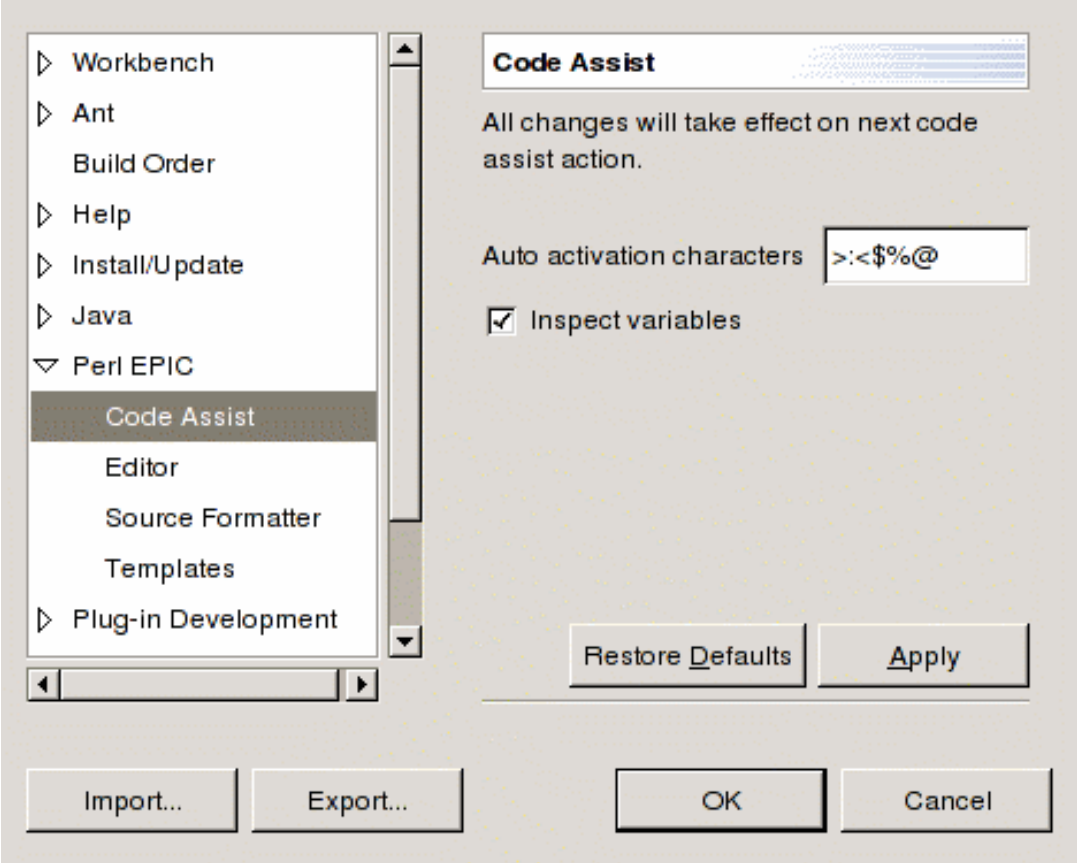
Note

On slower systems it might be useful to disable automatic syntax validation. Syntax validation is still possible by using the **Shift-F5** function key.

Code Assist

On the Code Assist Preference Page the auto completion trigger characters are defined. Normally it should not be necessary to change these values.

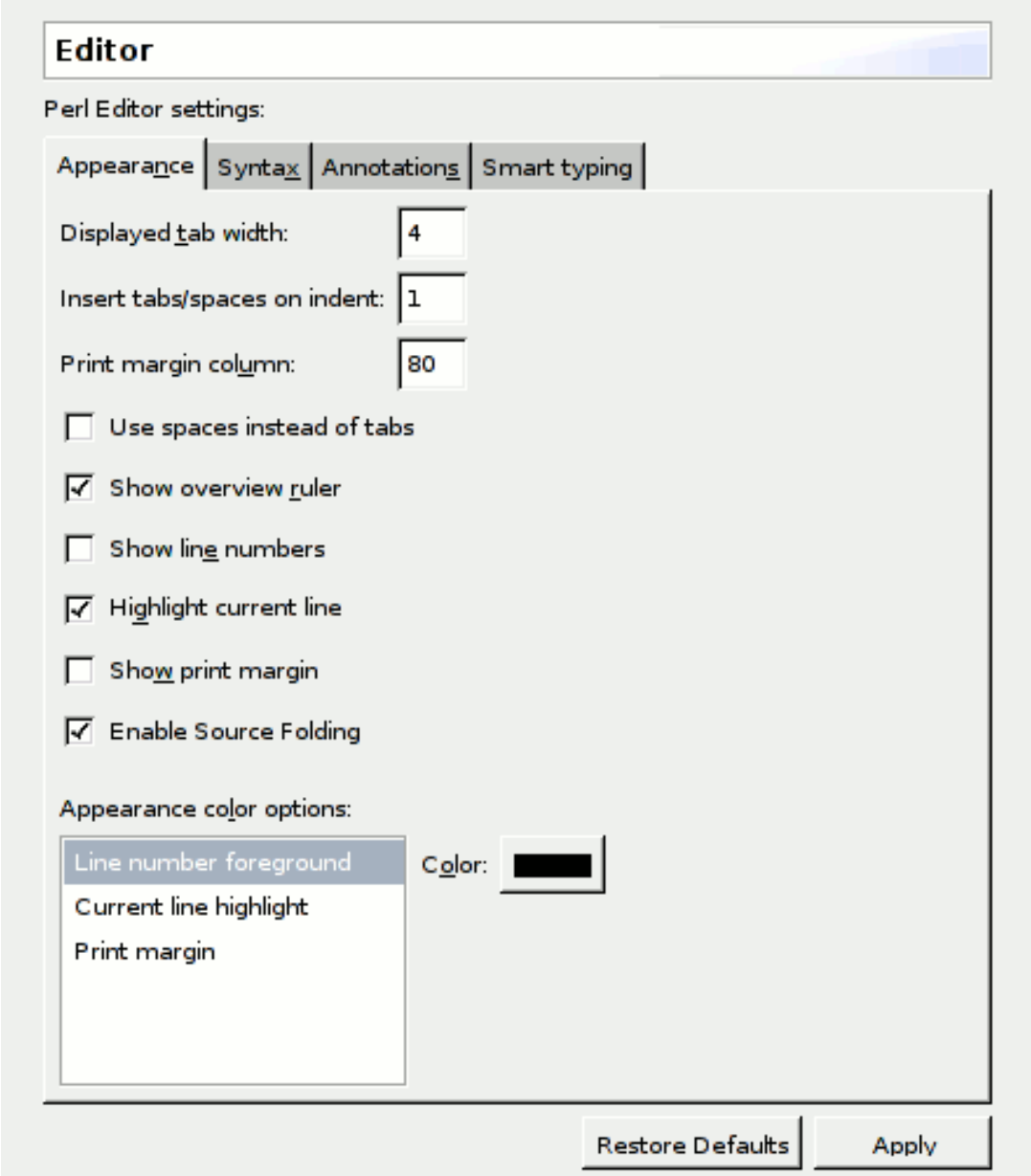
By default the editor suggests a list of already used variables when the characters **$ @ %** are typed. To switch this feature off, deselect the Inspect Variables check box.



Editor

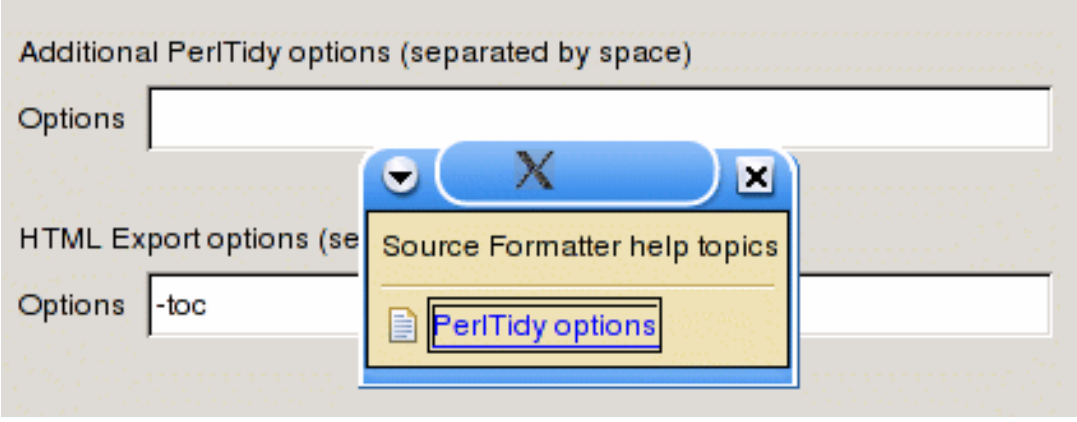
These options define the appearance of the Perl Editor, the coloring of the Perl source code, and how annotations are displayed.

Smart typing settings allow to switch auto-completion of quotes, parenthesis etc. on or off.



Source Formatter

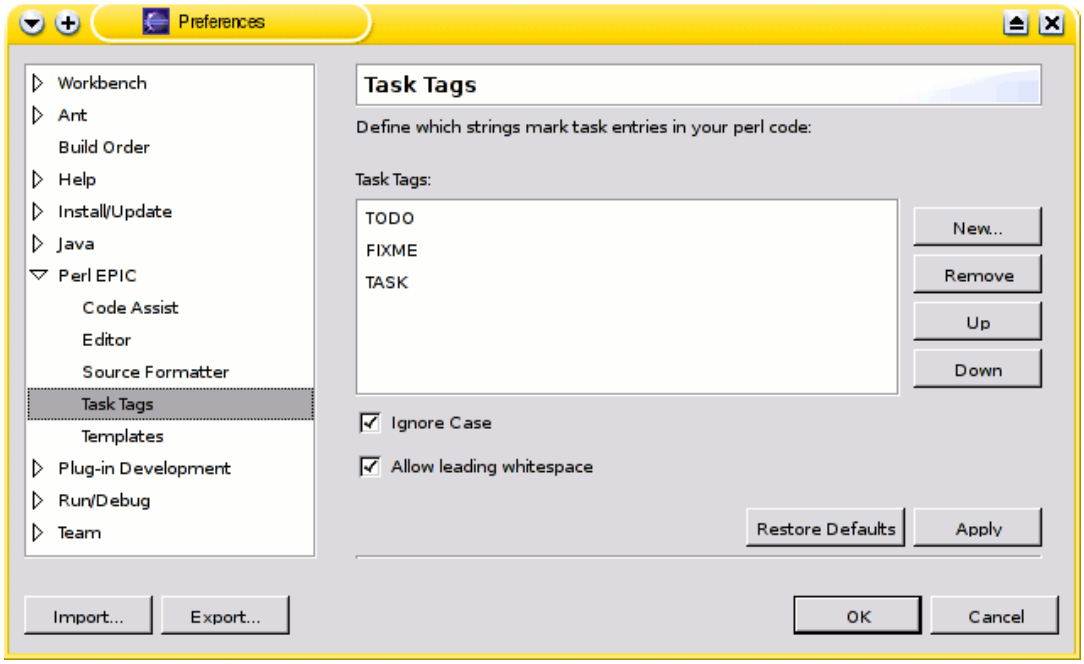
EPIC uses PerlTidy to format source code. The Source Formatter Preference allows to specify PerlTidy command line parameters. To get a description of available parameters, press the Help key and select PerlTidy options from the popup menu.



Note

In order for PerlTidy to work correctly, the Perl Interpreter Preferences have to be setup correctly (see above).

Task Tags



In this section, you can specify a list of keywords that act as markers for tasks inside comments in your perl code, i.e. tags that mark the beginning of a task entry.

By default, the words `TODO` and `TASK` mark the beginning of a task.

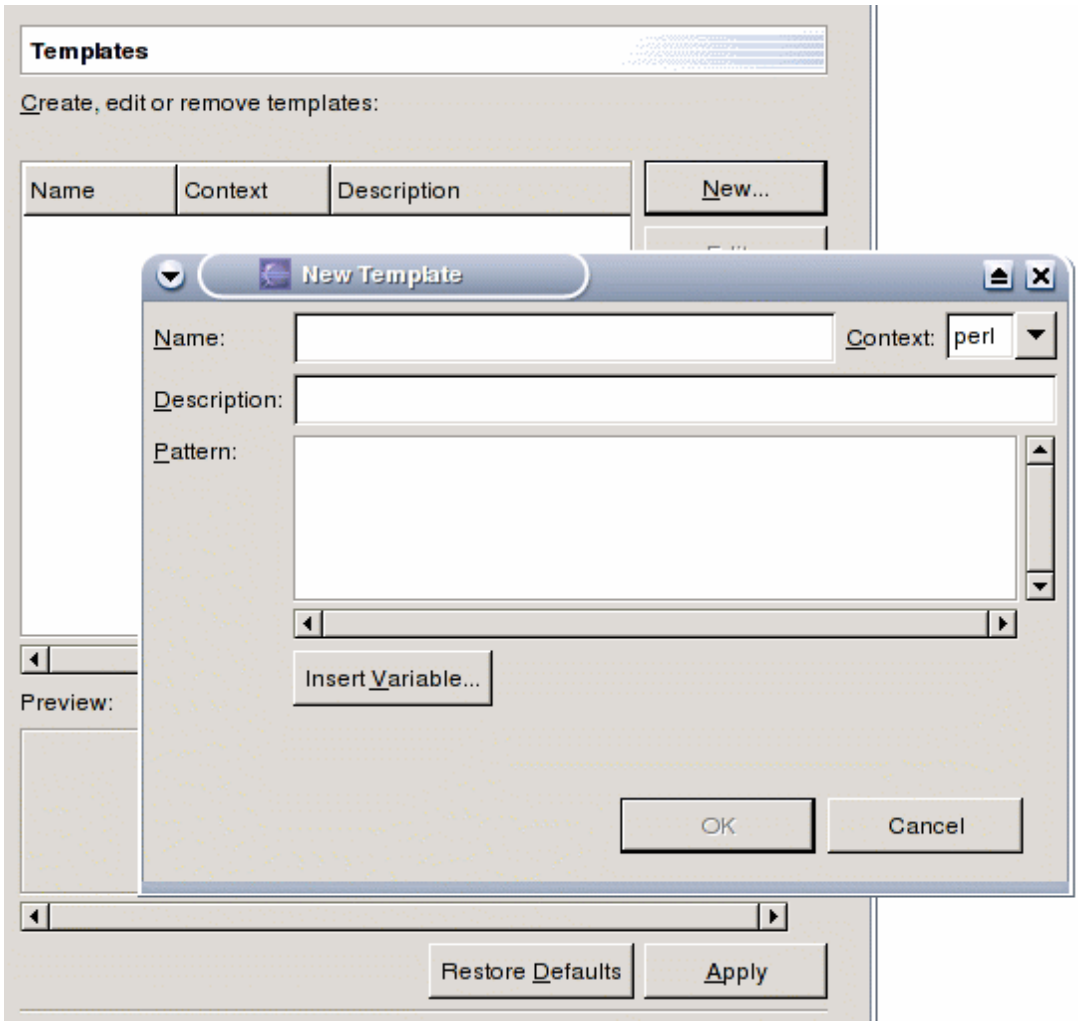
Check the Ignore Case option if you want EPIC to recognize task tags case-insensitively, e.g. `# todo my task`.

If you select Allow whitespace, task tags do not need to follow a comment sign (#) directly, e.g. `# TODO my task` instead of having to write `#TODO my task`.

See the section called “Task Markers” to see how to use Task Tags in the Perl Editor.

Templates

Templates are a powerful tool to insert pre defined code snippets while working with the Perl Editor.

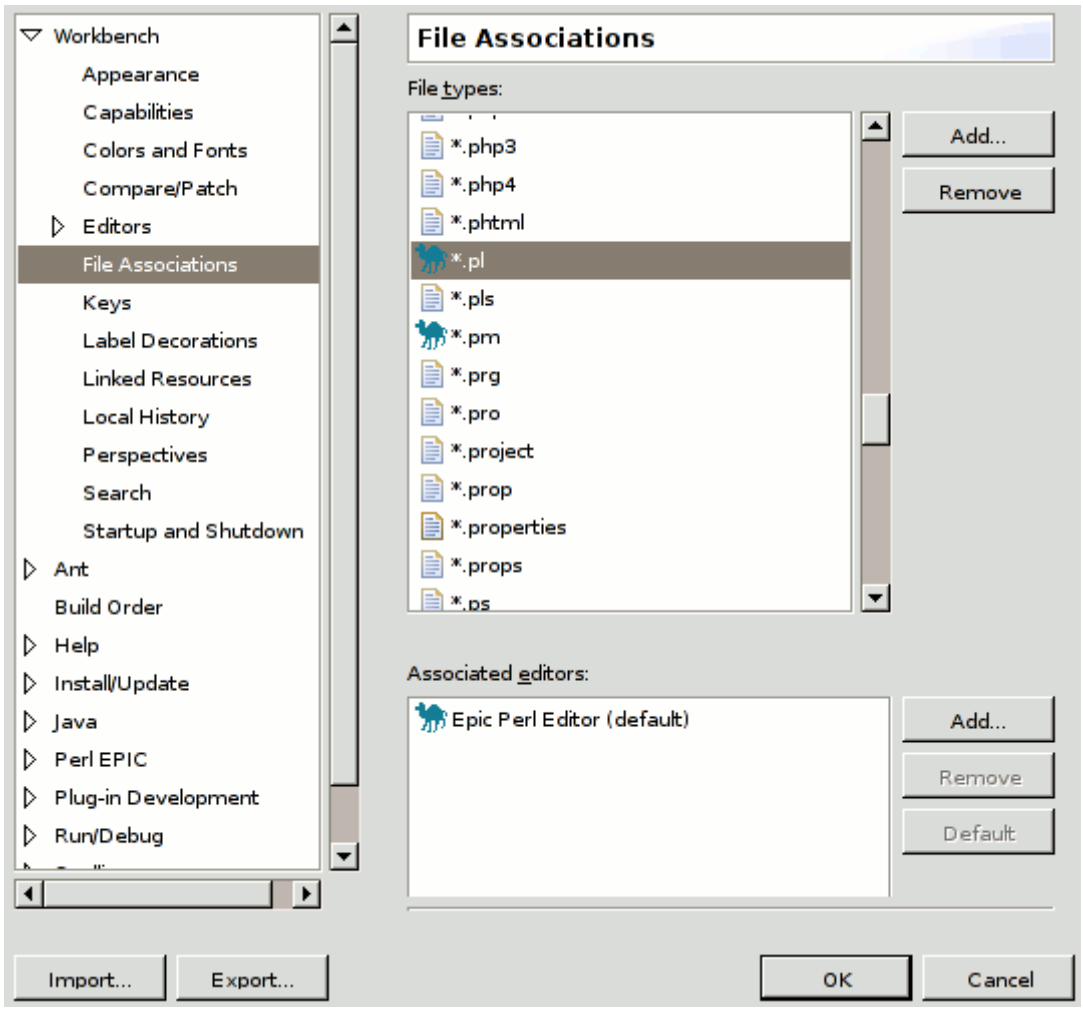


How Templates are use is covered in *Working with the Perl Editor* . The Templates Preference page allows the creation, import and export of Templates.

Exported Templates are stored in XML format.

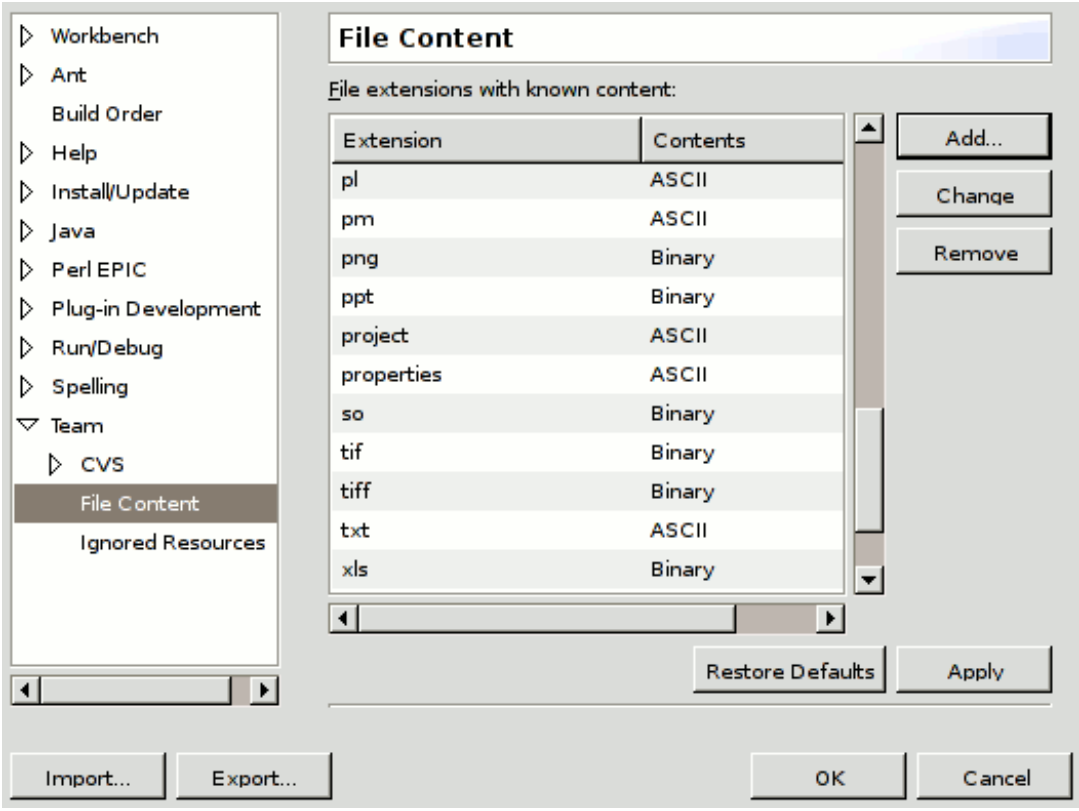
Associating Files with the Perl Editor

Eclipse associates file extensions with editors. If another plug-in is installed, the EPIC Perl Editor might not be used as the default editor when opening *.pl, *.pm or *.cgi files. To associate these file extensions with the Perl Editor, choose Window->Preferences... from the Eclipse menu and select Workbench->File Associations. If the Perl extensions are missing, they can be created by pressing the Add... button. Select the Perl Editor from the list and press the Default button.



CVS Setup

By default, Eclipse stores Perl files as binary when they are added to the CVS repository. To store Perl files as text (ASCII), select Window->Preferences... from the Eclipse menu and modify the Team->File Content settings. Add your Perl extensions (pl, pm etc.) by pressing the Add... button and specify `ASCII` in the Contents column.



Setting Up the Debugger

Setting up the debugger requires two steps:

1. Define the Perl interpreter to use.
2. Install the PadWalker Perl module.

Note

It is possible to use the debugger without installing PadWalker, but in this case local variables won't be shown.

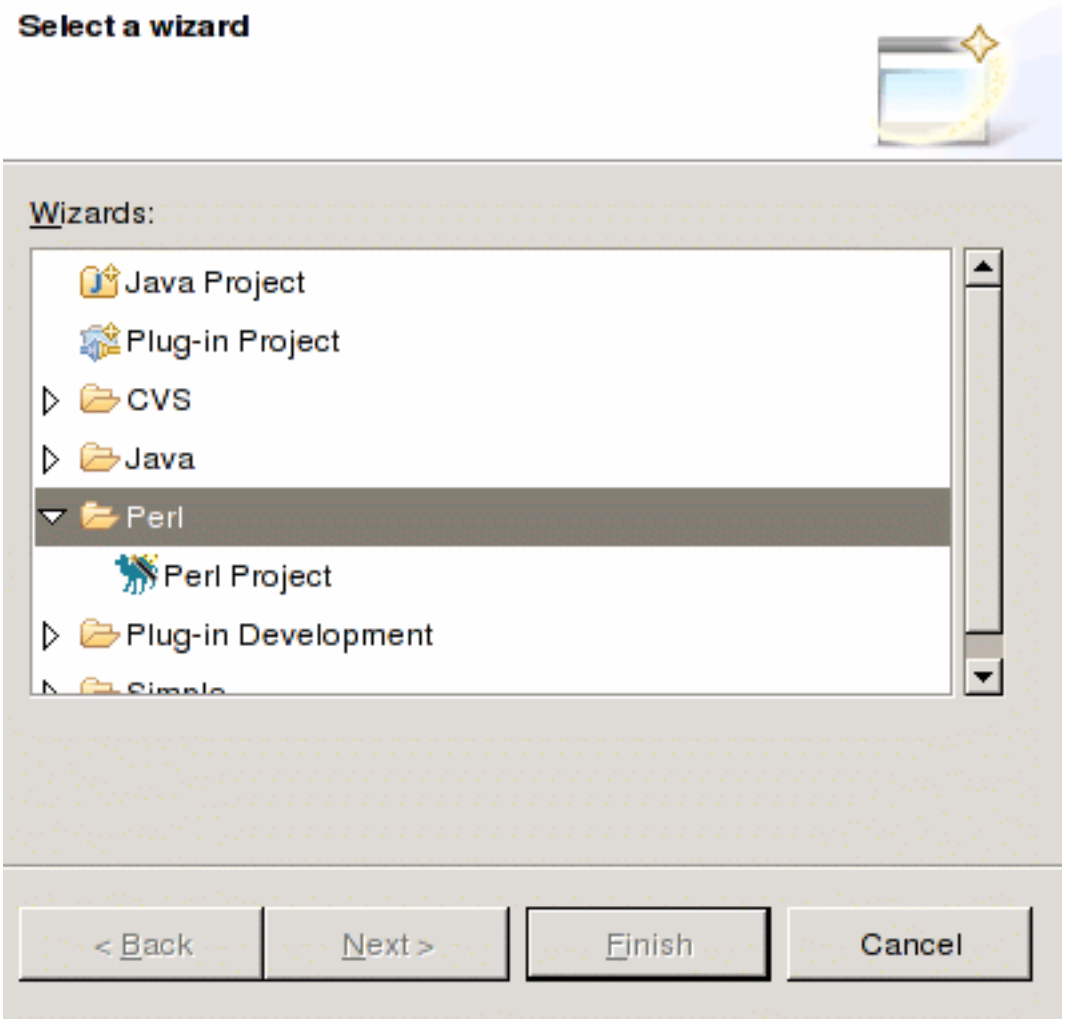
Download the PadWalker module from CPAN [http://www.cpan.org/] (PadWalker 0.10 [http://search.cpan.org/%7Erobin/PadWalker-0.10/]) and install as described in the installation notes or use the installation manager provided with your Perl installation (e.g. PPM for ActiveState installations).

If you are using ActiveState 5.8.x on a Windows system, you can download a pre-compiled PadWalker module from the EPIC homepage [http://e-p-i-c.sourceforge.net/] as the version provided by ActiveState doesn't seem to work for Perl 5.8.x (but it does for 5.6.x).

Perl Projects

Creating a Project

Perl projects are created (like any other project) by selecting File->New->Project... from the Eclipse menu.

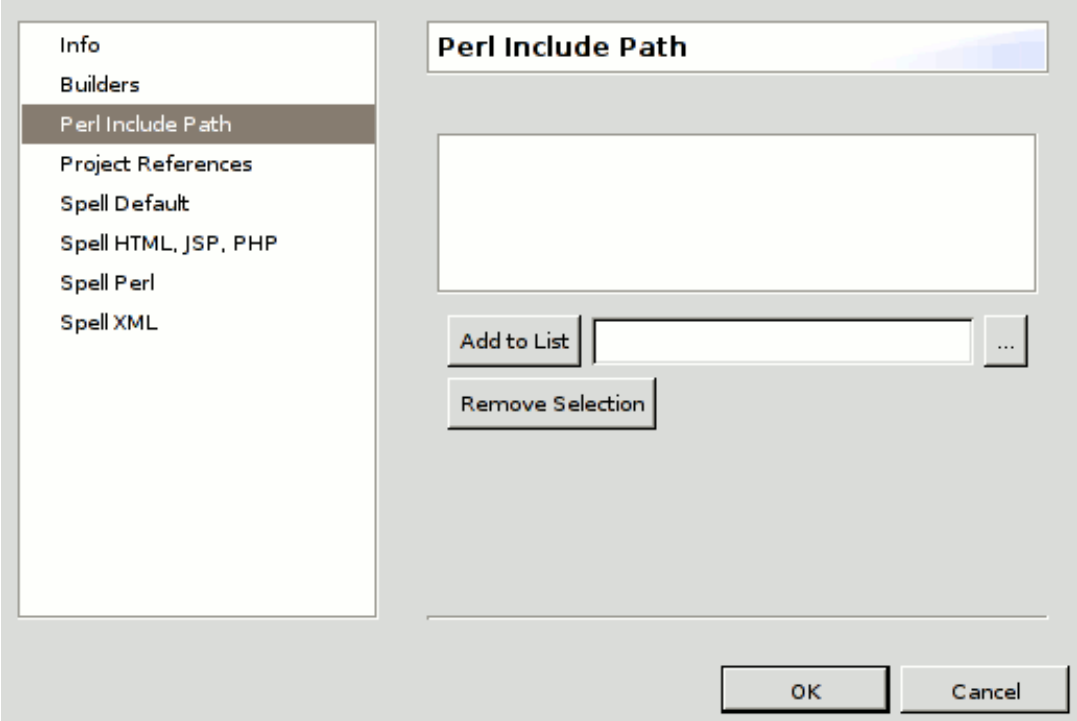


Follow the wizard's instructions to create your Perl Project. Perl Projects appear with a custom folder icon in the Navigator view:

Perl Include Path

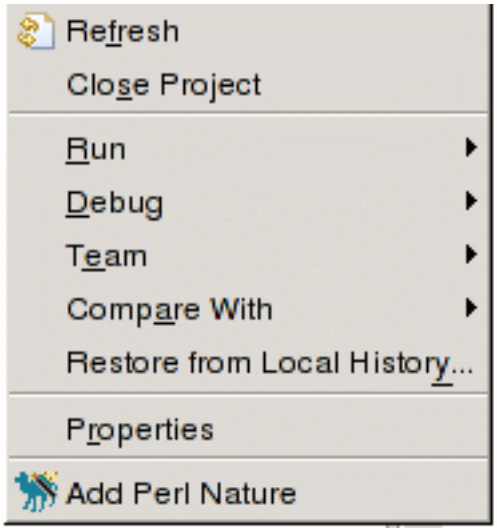
To add entries to a project's Perl Include Path (@INC), right click on the project icon and select Properties....

If non-absolute paths are entered, they are interpreted as relative to the project folder. Standard Eclipse variables (e.g. **`${project_loc}`**) can be used.



Converting an Existing Project

To add the Perl Nature to an existing project, select the project in the Navigator and select Add Perl Nature from the context menu.



To remove the Perl Nature from a project, select the project and select Remove Perl Nature from the context menu.

Eclipse Basics

Perspectives

Each Workbench window contains one or more perspectives. A perspective defines the initial set and layout of views in the Workbench window. Within the window, each perspective shares the same set of editors. Each perspective provides a set of capabilities aimed at accomplishing a specific type of task or works with specific types of resources. For example, the Java perspective combines views that you would commonly use while editing Java source files, while the Debug perspective contains the views that you would use while debugging Java programs. As you work in the Workbench, you will probably switch perspectives frequently.

Perspectives control what appears in certain menus and toolbars. They define visible *action sets*, which you can change to customize a perspective. You can save a perspective that you build in this manner, making your own custom perspective that you can open again later.

You can set your Workbench preferences to open perspectives in the same window or in a new window.

The main perspectives for developing Perl applications are:

This is the main perspective for coding Perl scripts.

Provides the main functionality for debugging and executing Perl scripts. For details see the section called “Views in the Debug Perspective”.

Views

Views support editors and provide alternative presentations as well as ways to navigate the information in your Workbench. For example, the Navigator view displays projects and other resources that you are working with.

Views also have their own menus. To open the menu for a view, click the icon at the left end of the view's title bar. Some views also have their own toolbars. The actions represented by buttons on view toolbars only affect the items within that view.

A view might appear by itself, or stacked with other views in a tabbed notebook. You can change the layout of a perspective by opening and closing views and by docking them in different positions in the Workbench window.

Using Perspectives

New Perspectives

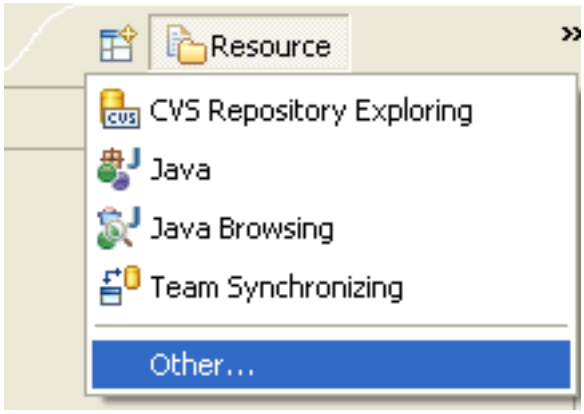
There are several ways to open a new perspective within this Workbench window:

- Using the Open Perspective button on the shortcut bar.
- Choosing a perspective from the Window->Open Perspective menu.

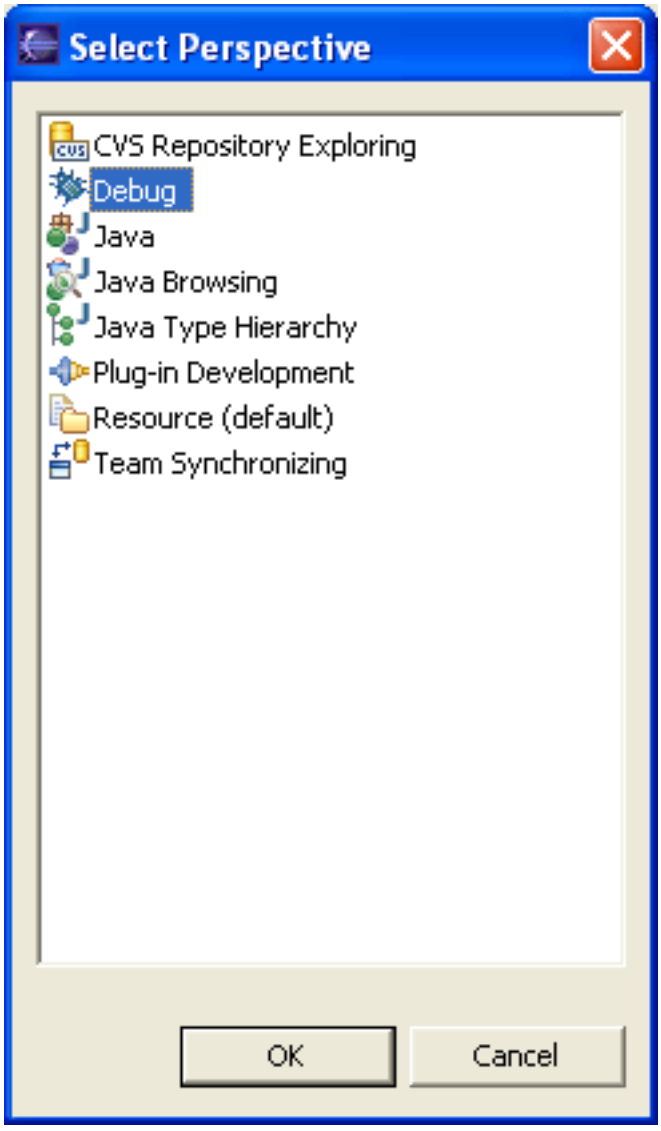
To open one by using the shortcut bar button:

1. Click on the Open Perspective button .

2. A menu appears showing the same choices as shown on the Window->Open Perspective menu. Choose Other from the menu.



3. In the Select Perspective dialog choose Debug and click OK.



The Debug perspective is displayed.

4. There are several other interesting things to take note of.

   - The title of the window now indicates that the Debug perspective is in use.
   - The shortcut bar contains several perspectives, the original Resource perspective, the new Debug perspective and a few others. The Debug perspective button is pressed in, indicating that it is the current perspective.
   - To display the full name of the perspective right click the perspective bar and check Show Text.

5. In the shortcut bar, click on the Resource perspective button. The Resource perspective is once again the current perspective. Notice that the set of views is different for each of the perspectives.

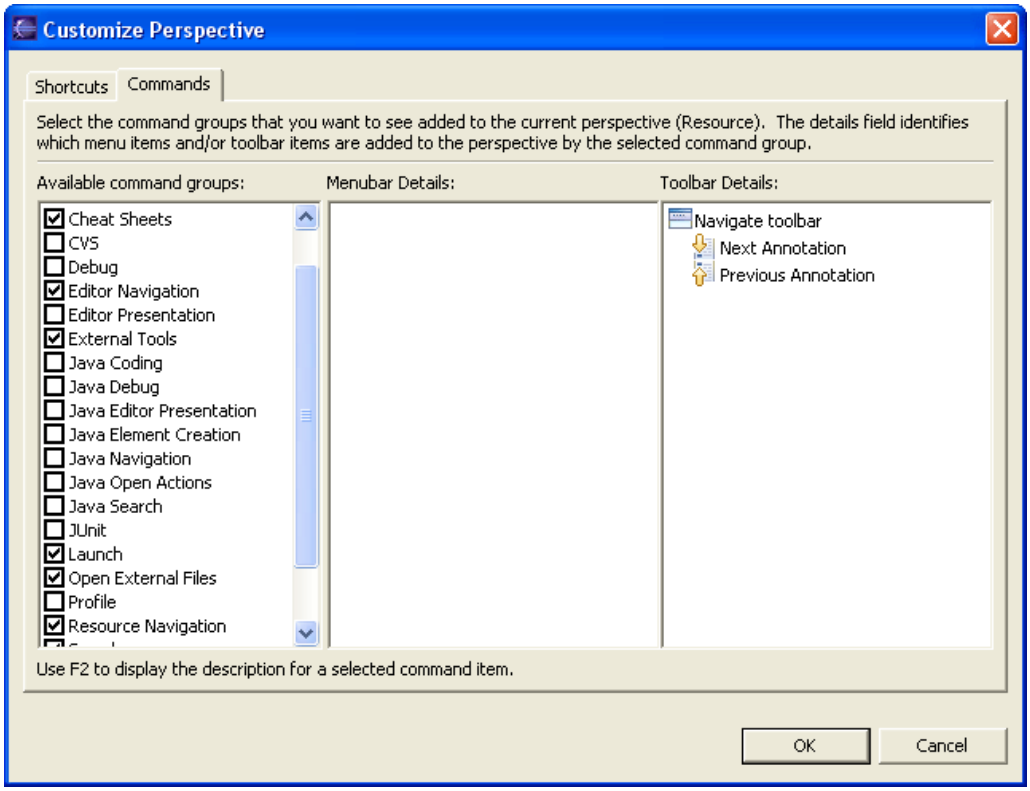
Configuring Perspectives

In addition to configuring the layout of a perspective you can also control several other key aspects of a perspective. These include:

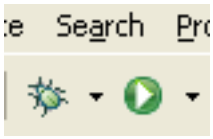
- The New menu.
- The Window->Open Perspective menu.
- The Window->Show View menu.
- Action sets that show up on the toolbar.

Try customizing one of these items.

1. In the shortcut bar click on the Resource perspective.
2. Select Window->Customize Perspective....
3. Select the Commands tab.
4. Check Launch and click OK.



5. Observe that the toolbar now includes buttons for debug/run launching.



6. After experimenting with the other options on the Customize Perspective dialog, choose Window->Reset Perspective to return the perspective to its original state.

Saving a User Defined Perspective

If you have modified a perspective by adding, deleting, or moving (docking) views, you can save your changes for future use.

1. Switch to the perspective that you want to save.
2. Click Window->Save Perspective As.
3. Type a new name for the perspective into the Name field.
4. Click OK.

Resetting Perspectives

To restore a perspective to its original layout:

1. Click Window->Preferences.
2. Expand Workbench and choose Perspectives.

3. From the Available perspectives list, select the perspective you want to restore.
4. Click Reset.
5. Click OK.

Using Views

Opening Views

Perspectives offer pre-defined combinations of views and editors. To open a view that is not included in the current perspective, select Window->Show View from the main menu bar.

You can create *fast views* to provide a shortcut to views that you use often.

After adding a view to the current perspective, you may wish to save your new layout by clicking Window->Save Perspective As.

Moving and Docking Views

To change the location of a view in the current perspective:

1. Drag the view by its title bar. Do not release the left mouse button yet.
2. As you move the view around the Workbench, the mouse pointer changes to one of the drop cursors shown in the table below. The drop cursor indicates where the view will be docked if you release the left mouse button. To see the drop cursor change, drag the view over the left, right, top, or bottom border of another view or editor.
3. When the view is in the location that you want, relative to the view or editor area underneath the drop cursor, release the left mouse button.
4. (Optional) If you want to save your changes, select Window->Save Perspective As from the main menu bar.
5. Note that a group of stacked views can be dragged using the empty space to the right of the view tabs.

Working with the Perl Editor

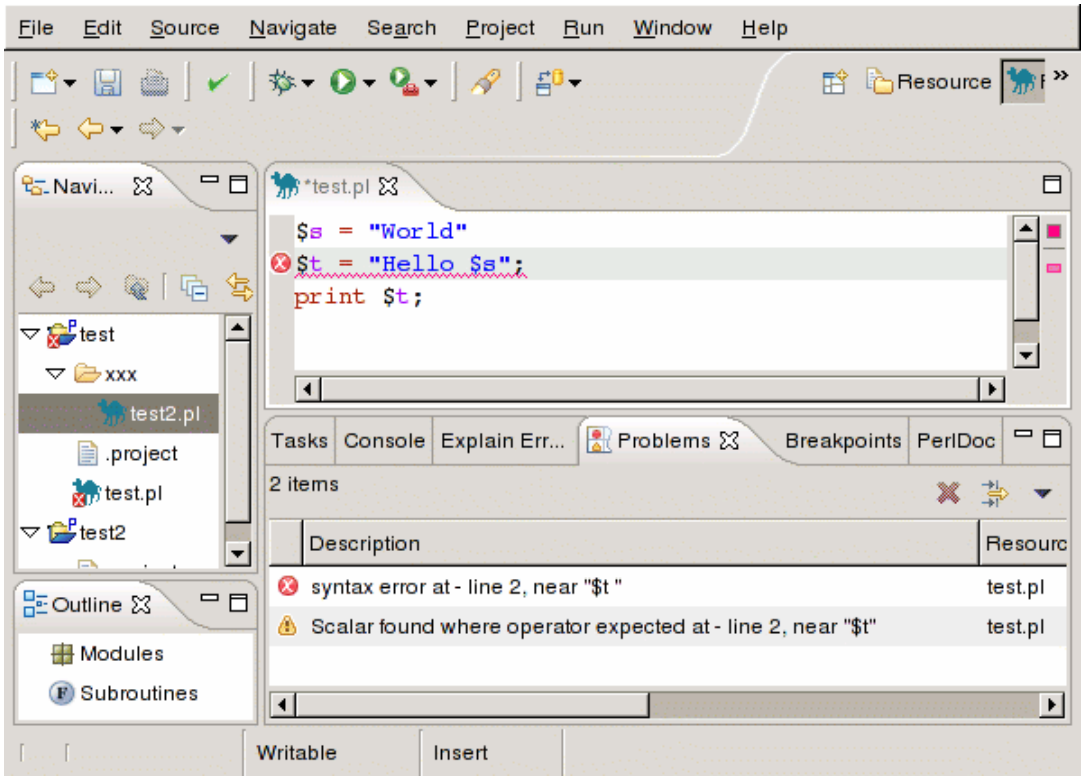
Syntax Check

EPIC performs on the fly syntax check of Perl source files. In order for the Syntax Check to work, the Perl Interpreter has to be set up correctly (see *Setting Up Preferences*).

The Syntax Check is performed after a defined idle period, after the user has stopped typing. This idle period can be configured in the preferences.

When an error/warning has been found, the editor displays the appropriate icon in the annotation ruler (the gray bar on the left side of the editor), underlines the error in the source, and inserts a marker into the Problems view.

A syntax check can be enforced by pressing **Shift-F5**. It is also triggered automatically by saving a source file.

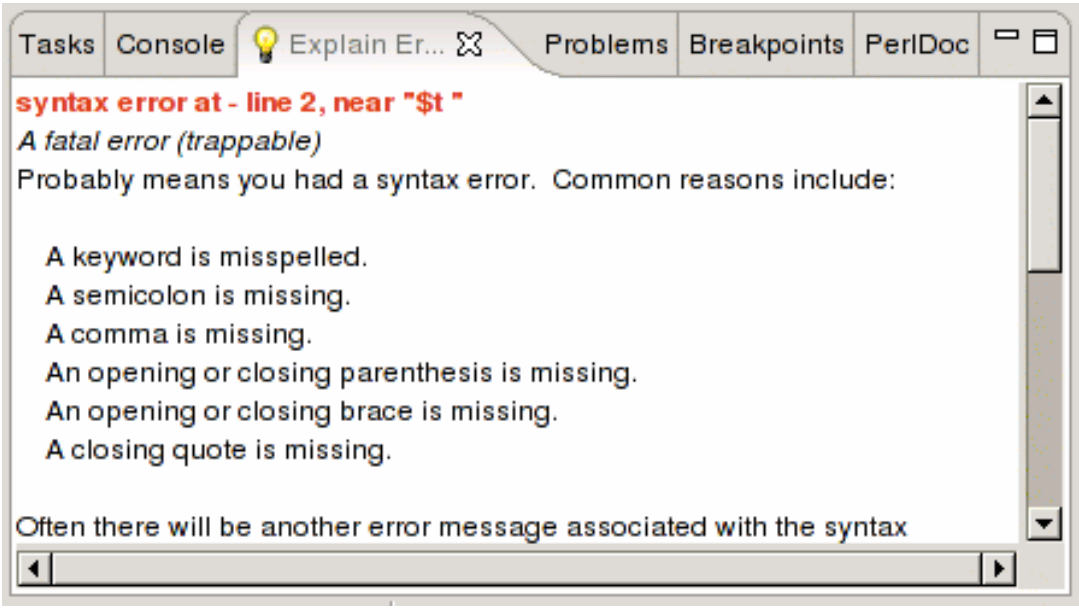


Explain Errors and Warnings

In addition to displaying warnings and errors, the editor is capable of explaining them in more detail.

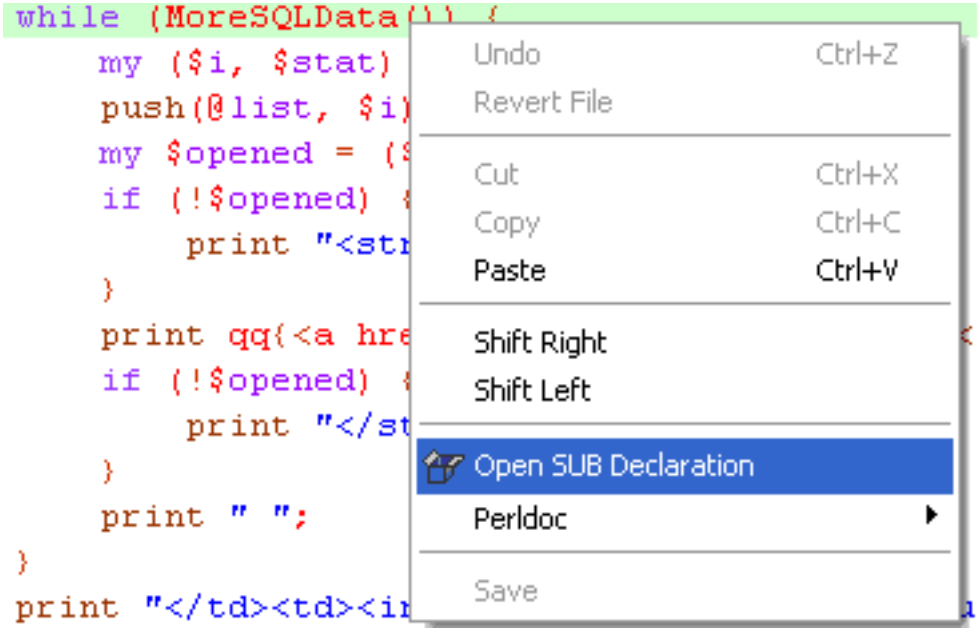
To get an Error/Warning explanation, right-click the Error/Warning icon and select Explain Errors/ Warnings from the context menu.

The explanation(s) will be displayed in the Explain Errors/Warnings view:



Open SUB Declaration

Open SUB Declaration allows the user to search for the declaration of a specific SUBroutine.



The search first determines what is selected. If no text is selected, it attempts to find a subroutine name at the current cursor position. The search will fail if no subroutine name is selected.

Due to the dynamic nature of Perl programs, the search is not entirely reliable. For subroutine names qualified by a package prefix, an attempt will be made to locate the appropriately named module file using the @INC path. For unqualified subroutine names, the search will first occur in the current editor and then extend to modules referenced by 'use' statements.

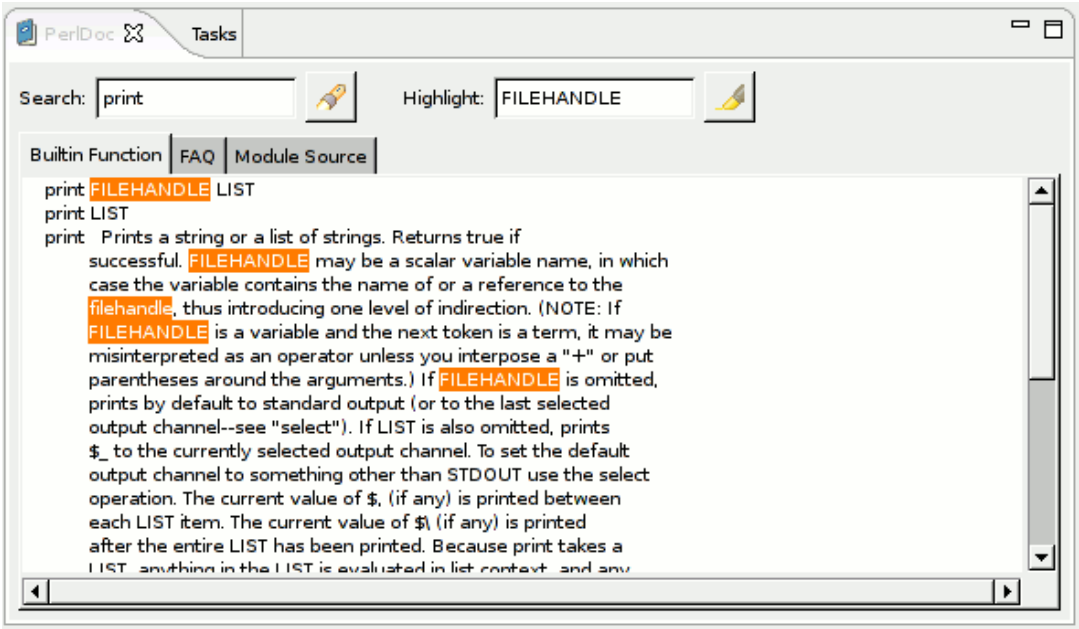
If the subroutine declaration is found in a file inside workspace, it will be highlighted in an existing or new editor.

```
sub MoreSQLData {
    if (defined @::fetchahead) {
    return 1;
    }
    if (@::fetchahead = $::currentquery->fetchrow()) {
```

Perldoc

To retrieve Perldoc information, select a keyword or text and choose Perldoc from the context menu or press **Shift-Ctrl-H**. If nothing is selected, an input dialog will appear.

If Perldoc entries are found, they are displayed inside the Perldoc view.



Note

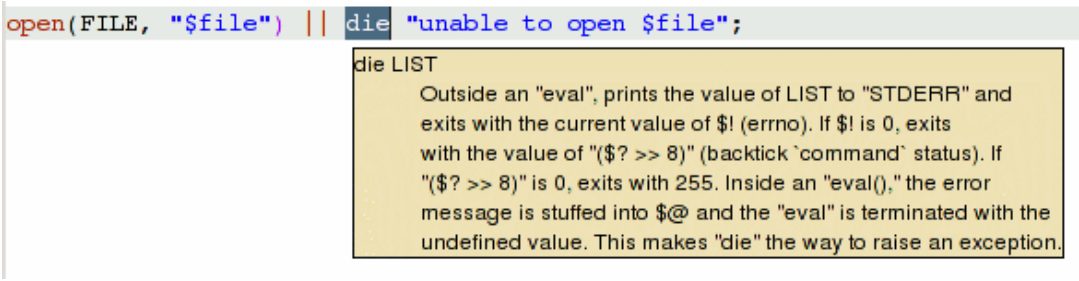
Perldoc has to be installed and available in the system PATH, otherwise this feature will not work.

Quick Reference

Apart from Perldoc support, a quick reference feature is available. This feature has the advantage that no perldoc has to be installed on the system but does not provide as much information as perldoc.

To view the Quick Reference, select a keyword and move the mouse pointer over the selection.

A tooltip with a short description of the keyword should appear.



Code Assist

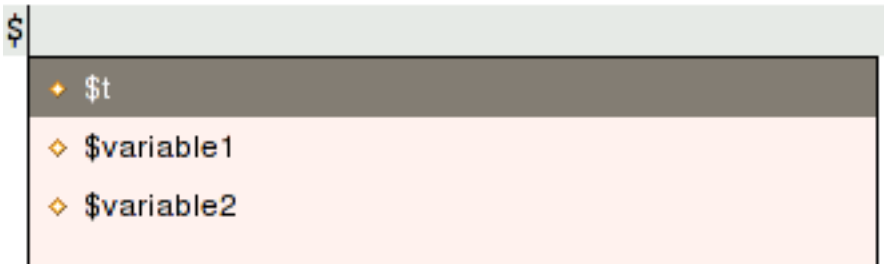
Code Assist features try to assist the user during source code editing.

Note

The features currently implemented in EPIC may not be fully functional but will be improved in the future.

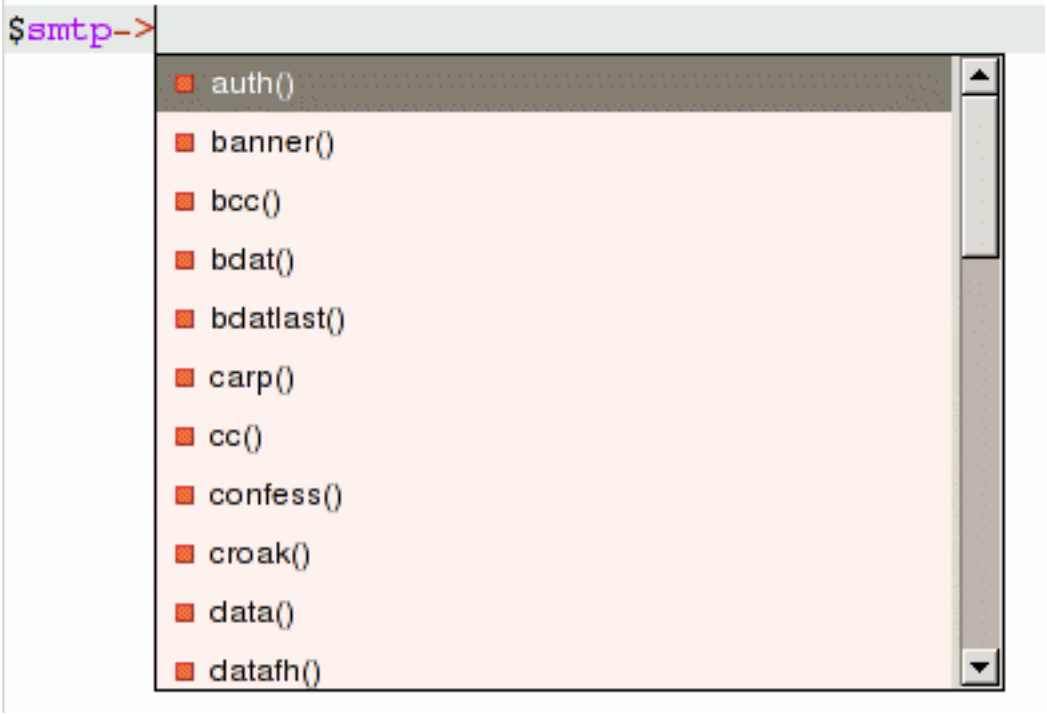
Variable Inspection

When you press one of the auto completion characters **$ @ %**, the editor displays all defined variables in a list. From the list you can select the variable that should be inserted in the source code.



Module Inspection

The editor tries to display methods available in modules when the auto completion characters **>** or **:** are entered.



Note

Currently, indirect object invocations are not recognized by code assist. This code block will not work:

```
$smtp = new Net::SMTP;
$smtp->[no content assist]
```

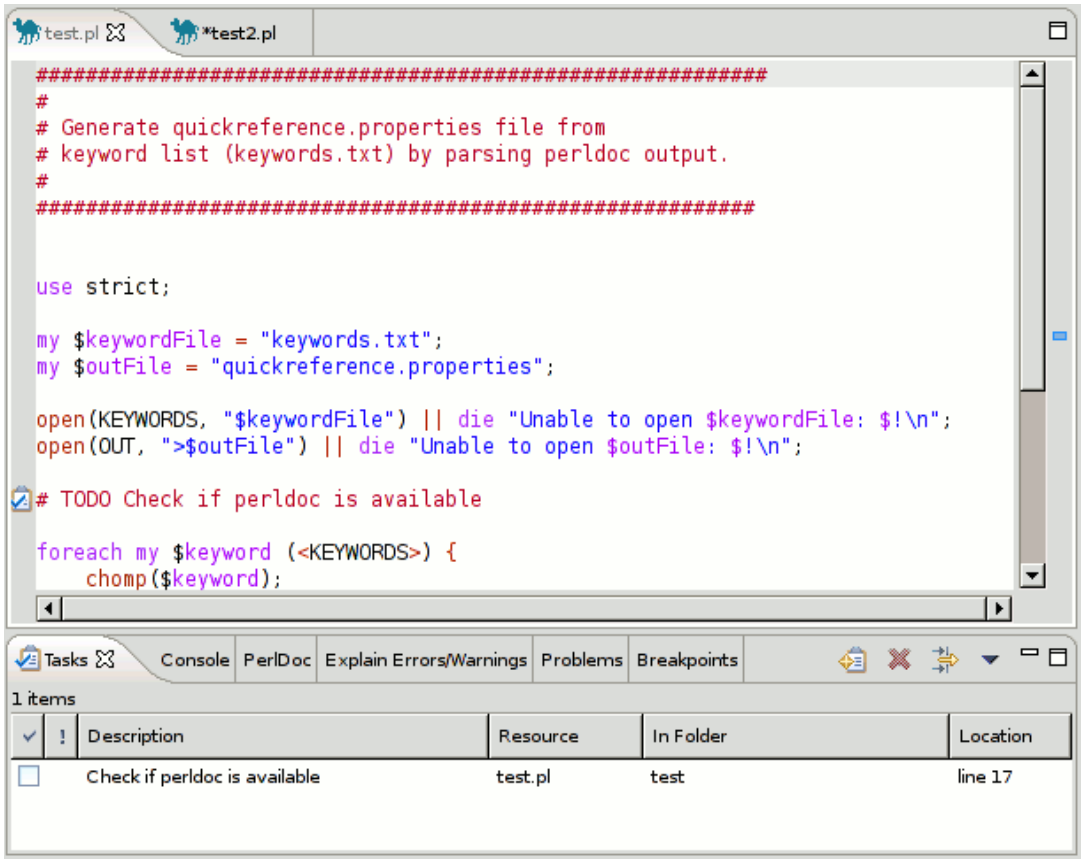
This one will work:

```
$smtp = Net::SMTP->new();
$smtp->[content assist]
```

Task Markers

Task markers are a very convenient way to add items to the Eclipse task list. A task marker is generated when a **`#TODO any text`** is found in the Perl source code. On deletion of the **`#TODO`** comment, the task marker is also deleted.

You can customize the keywords which begin task markers in the preferences (see the section called “Task Tags”).

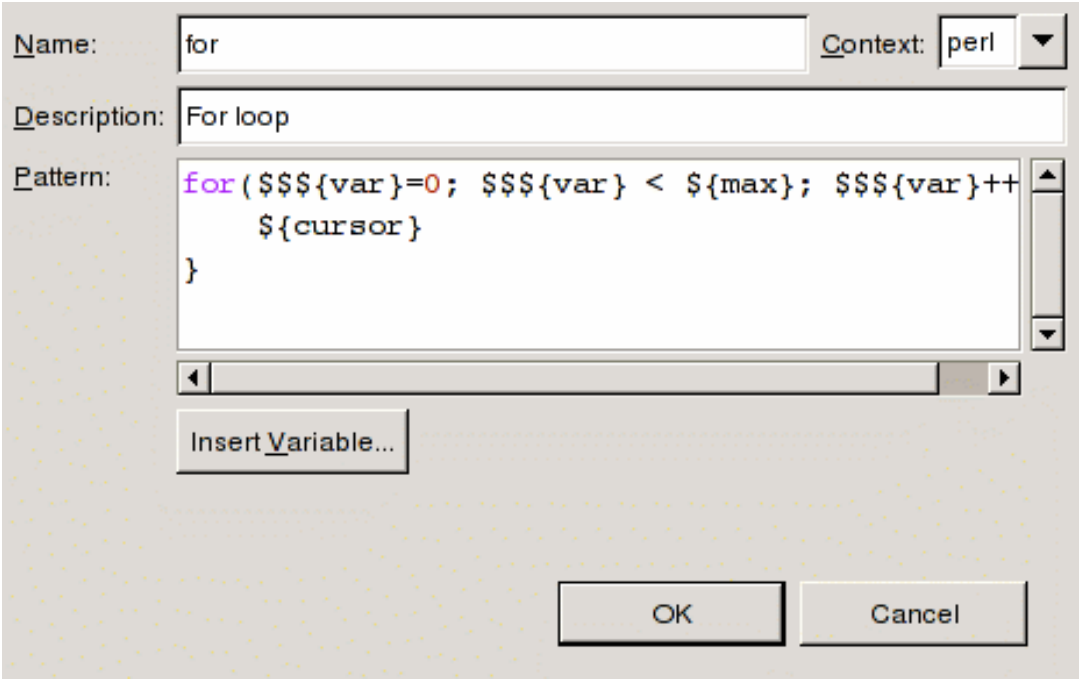


Templates

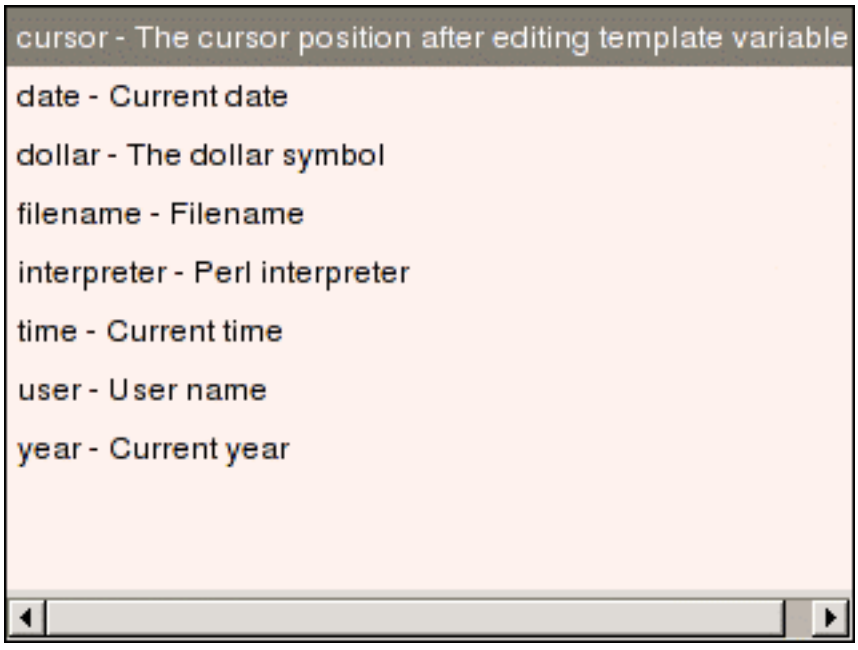
Templates allow for easy insertion of predefined text segments. In addition to normal text these segments can also include pre-defined variables that are included at runtime as well as variables that are specified by the user when the template is inserted.

Defining Templates

Templates are defined in the EPIC Preferences (Window->Preferences...). To define a new template, press the New... button.



To insert pre-defined variables, press the Insert Variable... button.

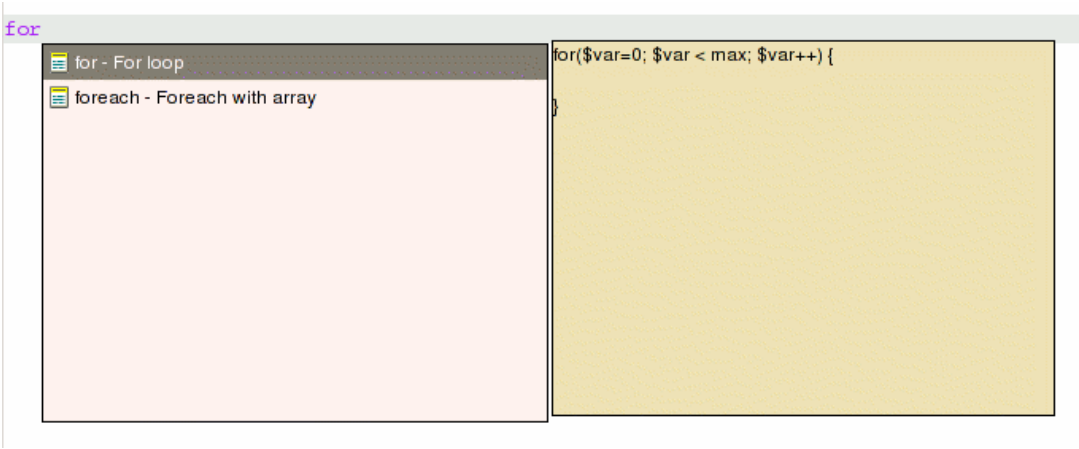


In addition to pre-defined variables, the user can specify additional variables (using the syntax **`${varname}`**) which can be edited when the template is inserted. When the first variable is inserted, variables with the same name will automatically be changed.

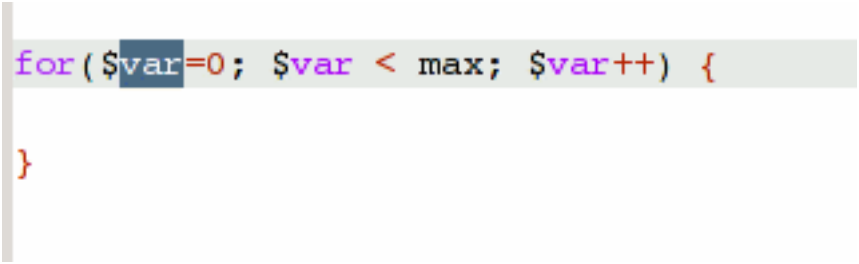
Using Templates

Templates are invoked by typing some characters and pressing **Ctrl-Space**.

Templates matching the typed characters will be displayed in a list. A preview is also available.



If the template contains user defined variables the user can press the **TAB** key to jump to the next variable after the template has been inserted.



Source Formatter

EPIC uses PerlTidy for source code formatting (PerlTidy is included in the EPIC package).

To format the source code, select Source->Format from the Eclipse menu or use **Ctrl-Shift-F**.

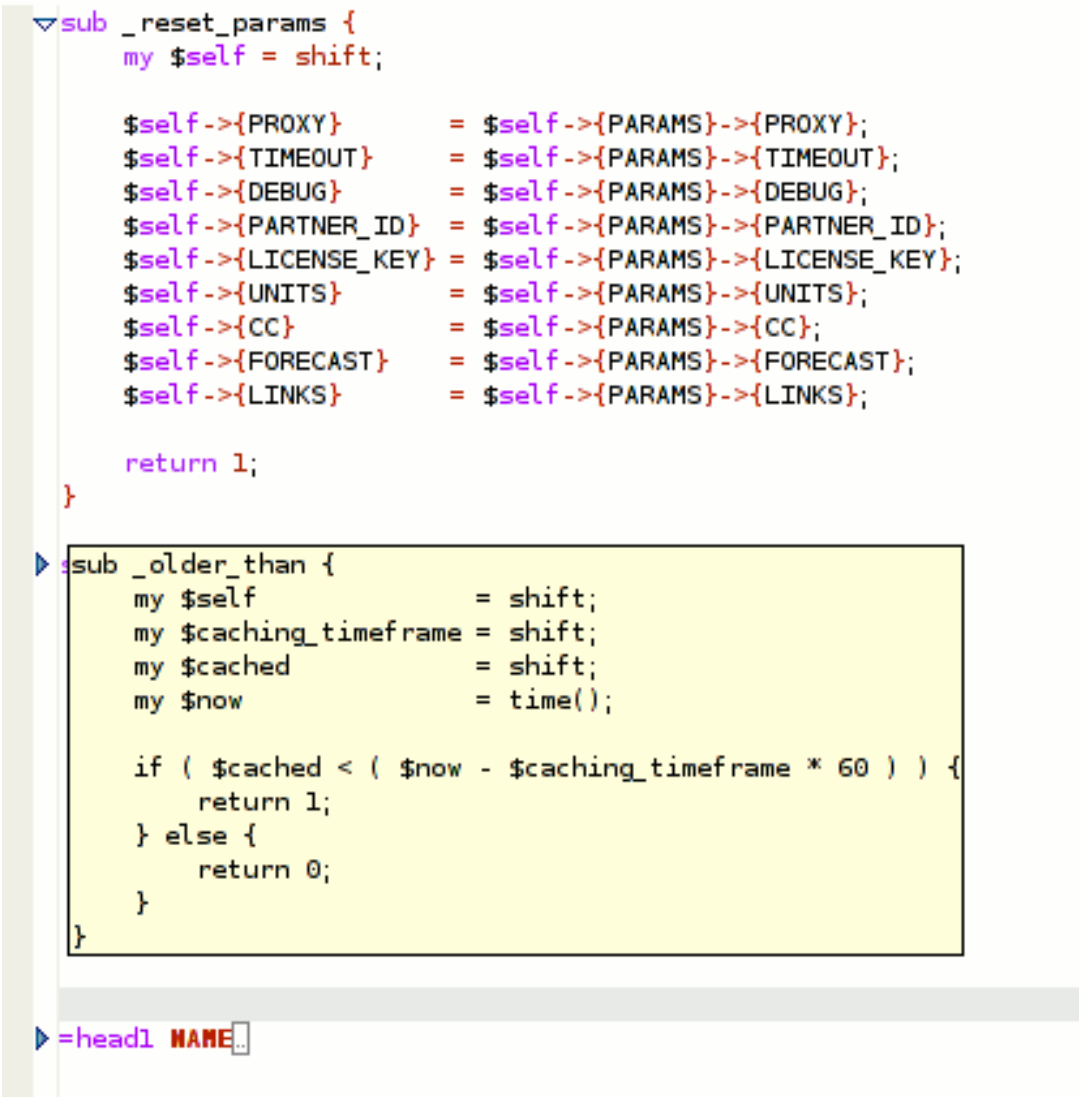
PerlTidy settings can be changed in the Source Formatter preference page.

Note

Source formatting might take a while if the source code has a lot of lines.

Source Folding

The editor supports folding of POD comments and subroutines.



Source folding can be disabled in the Editor preference page.

Note

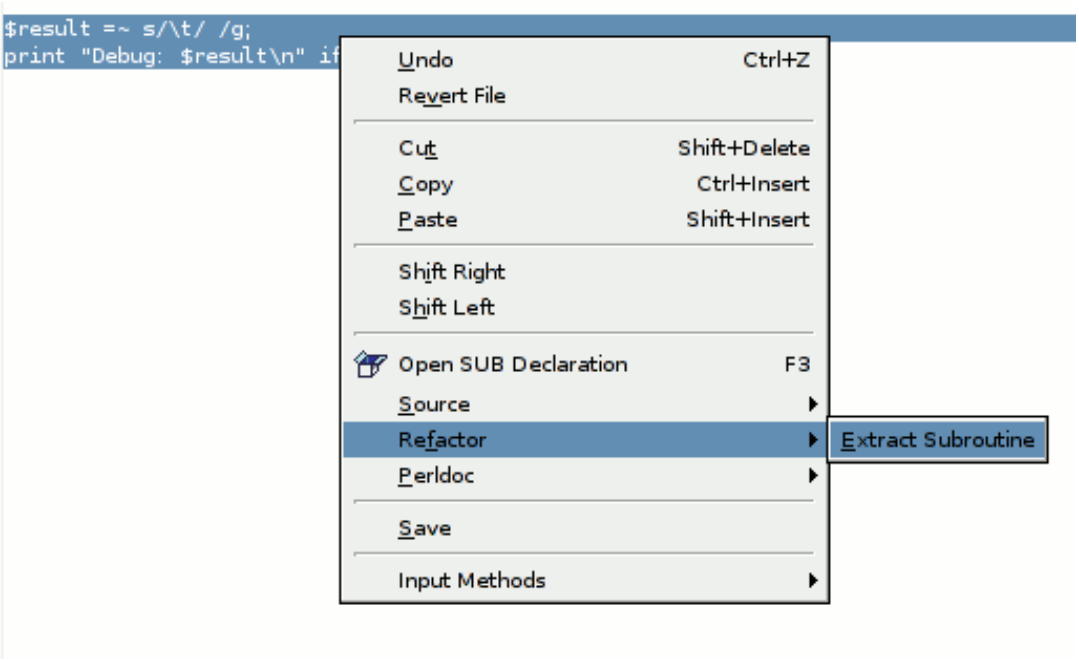
On big files source folding can decrease performance. So if you experience slowdowns, disabling source folding might help.

Refactoring

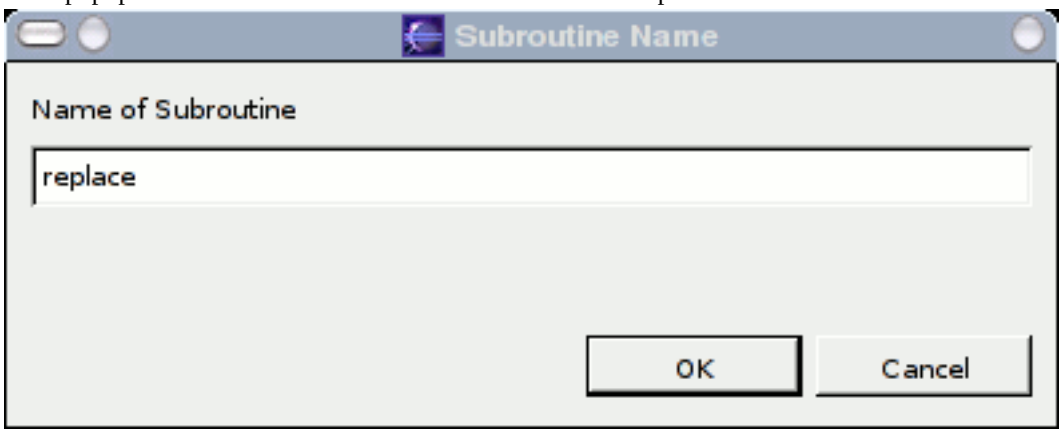
Extract Subroutine

Extraction of subroutines is supported by the use of the CPAN Devel::Refactor module.

To extract a subroutine, mark the code to extract and select Refactor->Extract Subroutine from the popup menu.



In the popup menu insert the name of the new subroutine and press **Enter**.



The new subroutine will be placed at the end of the Perl script (before __END__ section) and the selection will be replaced with the subroutine call.

```
my () = replace ($debug, $result);
sub replace {
    my $debug  = shift;
    my $result = shift;

    $result =~ s/\t/ /g;
    print "Debug: $result\n" if ($debug);
    return ();
}
```

Note

The extraction might not work properly at the moment because the Devel::Refactor module

is in an early stage of development. With upcoming versions of the module, this function should become more reliable.

HTML Export

To export, select Source->Export->HTML from the Eclipse menu and specify an output file.

HTML export settings can be changed in the Source Formatter preference page.

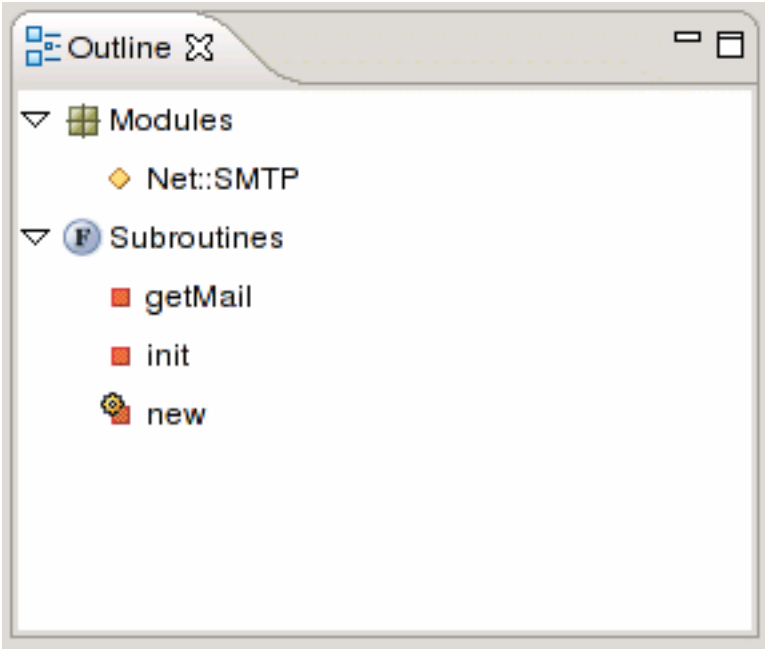
Note

For HTML export to work, a working Source Formatter is needed (see the section called “Source Formatter”).

Outline View

The Outline view displays packages and subroutines defined in the edited file. Modules referenced by 'use' statements are also shown. When you click on a module or subroutine name in the outline, the editor will jump to the appropriate position in the source code. When the cursor is moved inside of a subroutine's definition, the subroutine will become selected in the outline.

Subroutines named `new` will get a different icon.



Using the Perl Debugger

Launching Perl Programs

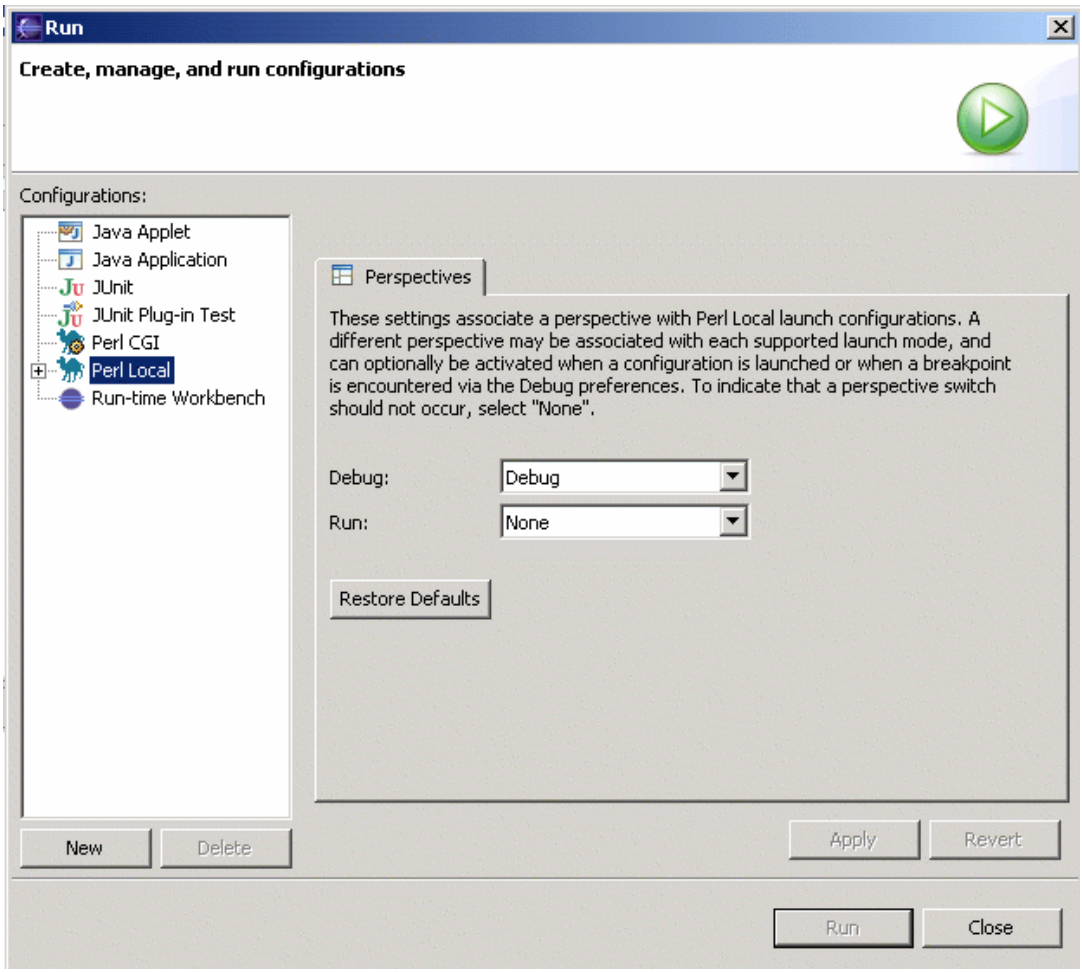
You may launch your Perl programs from the workbench. Programs may be launched in either run or debug mode.

- In run mode, the program executes, but may not be suspended or examined.
- In debug mode, execution may be suspended and resumed, variables may be inspected, and expressions may be evaluated.

The environment a Perl program is to be executed in is defined via "Launch Configurations". A launch configuration defines

- if the program is to be executed in a CGI or normal Perl environment
- the host the program is to be executed on
- the program to execute
- execution parameters to pass
- environment variables (for CGI environment)
- configuration data for the web server used to provide the CGI framework

Launching Perl Programs in Run Mode



1. Select Run->Run... from the Eclipse menu.
2. Within the appearing dialog, select the configuration type:

   - Perl Local: Run a Perl script on the local machine
   - Perl CGI: Run Perl programs in a CGI environment on the local machine
   - Perl Remote: Run a Perl script on a remote machine

   and press the New button to create a new launch configuration.
3. Adjust launch configuration attributes. For details see the section called “Creating Launch Configurations”.
4. Press the Run button.

This executes the program. The program's console output will be shown in the console window. For "Perl Local" and "Perl Remote" configurations, the console window also accepts keyboard input to be passed to the program.

If you switch to the debug view, you have additional control over the execution of the program. For details see the section called “Debug View”.

Re-launching a Perl Program

The workbench keeps a history of each launched and debugged program. To relaunch a program, do one of the following:

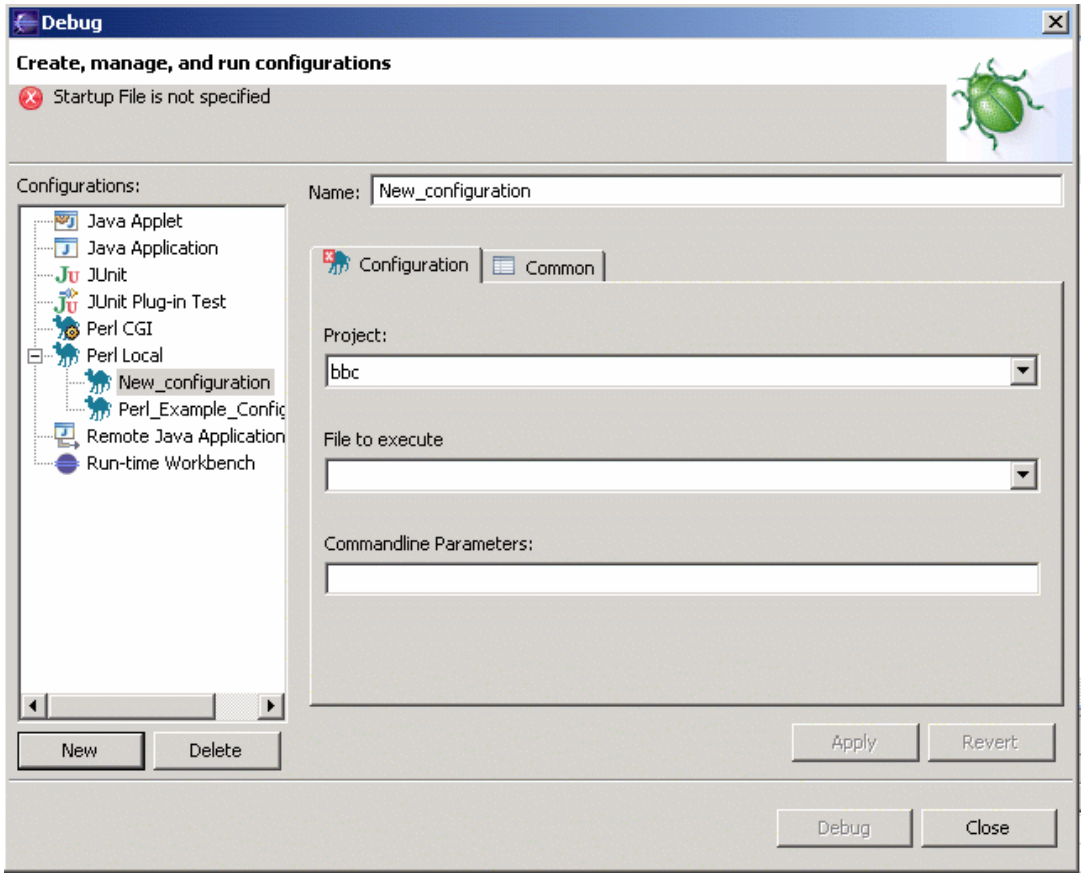
- Select a previous launch from Run or Debug button pull-down menus.
- From the menu bar, select Run->Run History or Run->Debug History and select a previous launch from these sub-menus.
- In the Debug view, select a process that you want to relaunch, and select Relaunch from the process's pop-up menu.

To relaunch the most recent launch, do one of the following:

- Click the Run or Debug buttons (without using the button pull-down menu).
- Select Run->Run Last Launched (**Ctrl-F11**), or Run->Debug Last Launched (**F11**) from the workbench menu bar.

Creating Launch Configurations

Perl Local: Running a Perl Script on the Local Machine



1. Enter the name for the launch configuration in the Name field.
2. In the Configuration tab

- Project field: select the project to debug

Note

Only Perl projects (projects associated with a Perl nature) will be shown. If the project you require is not shown, see the section called “Converting an Existing Project” for adding a Perl nature to your project.

- File to execute field: select the Perl file to execute

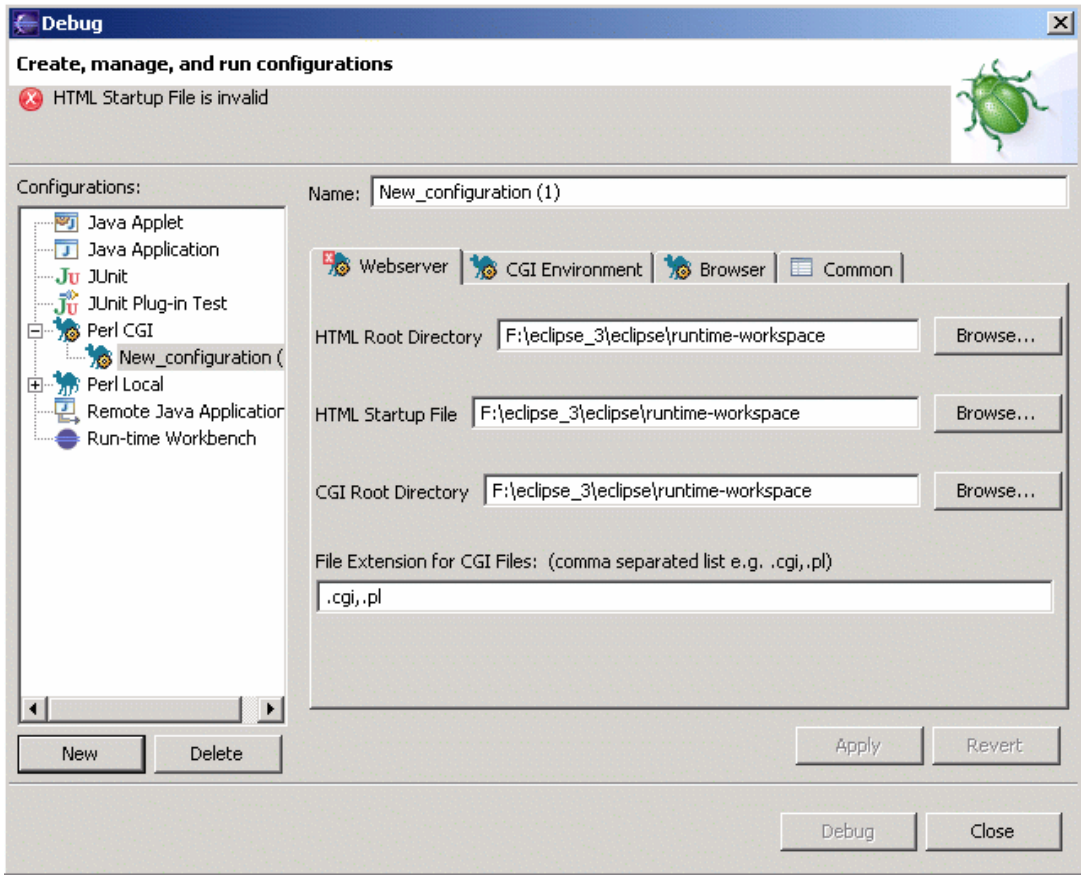
Note

Only files associated with the Perl editor will be shown. See the section called “Associating Files with the Perl Editor” for details.

- Commadline Parameters field: enter command line parameters to pass to the selected Perl script

3. Press the Apply button

Perl CGI: Run Perl Programs in a CGI Environment



1. Enter the name for the launch configuration in the Name field.

2. In the Web Server tab

- HTML Root Directory field: enter the base directory that contains all HTML files or use

the Browse button to select the appropriate directory.

- HTML Startup File field: enter the file name to be shown in the browser after startup or use the Browse button to select this file.
- CGI Root Directory field: enter the base directory that contains all CGI files or use the Browse button to select the appropriate directory.
- Extension for CGI files field: this is a comma separated list of file extensions (each starting with a ".") used for CGI files in the project.

3. In the CGI Environment tab

   - CGI-Script Environment Variables field: sorted list of environemt variables to be set for CGI scripts to be executed. Use the following syntax to specify environment variables: **`name=value`**.

4. In the Browser tab, there are two possible settings:

   - Select Custom Browser: specify the path to the browser executable and add the required command line parameters. Use `%1` as a place holder for the HTML file to be opened.
   - Select Default System Browser to use the browser defined as default browser by your OS preferences.

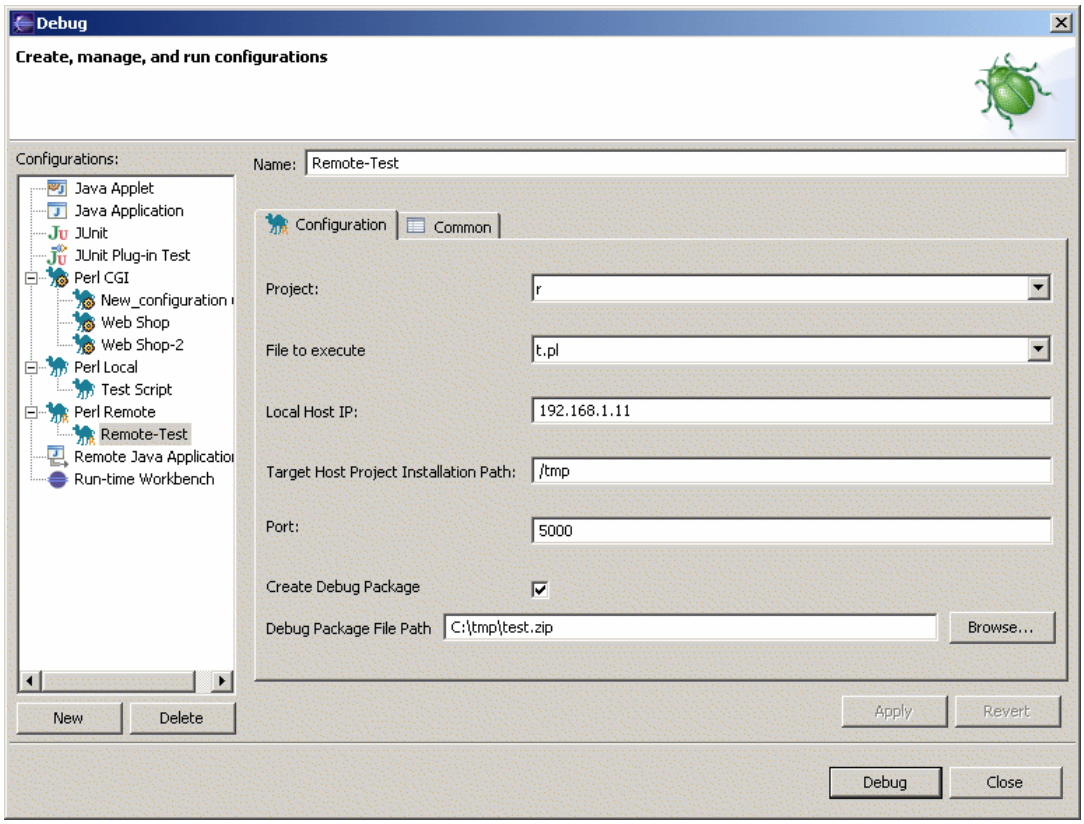
5. Press the Apply button.

Perl Remote: Debug a Perl Script on a Remote Machine

Some points to consider:

- You need a Perl interpreter installed on your target machine.
- The EPIC debugger opens a network connection between local and remote host (port number can be specified in the corresponding launch configuration). So have a look at your firewall configuration.
- Scripts executed and code shown are from different sources: EPIC displays the code present on your local machine and executes a copy of this code on the remote host. So if you make changes within your project please make sure to transfer these changes to the remote host (see below for further details).

  This problem gets more significant for code/modules provided by your Perl installation. If modules on your local and remote machine differ, debugging results may be quite meaningless. So try to have the same Perl version installed on both machines and make sure all modules you require (and are not part of your Eclipse project) are identical.
- @INC path handling: if the include path references directories within your project or directories that are linked into your projects, EPIC will include these modules in the list of files to be copied to the remote host and adjust the @INC-path for your remote machine accordingly.



1. Enter the name for the launch configuration in the Name field.

2. In the Configuration tab

   - Project field: select the project to debug.

   ## Note

   Only Perl projects (projects associated with a Perl nature) will be shown. If the project you require is not shown, see the section called “Converting an Existing Project” for adding a Perl nature to your project.

   - File to execute field: select the Perl file to execute.

   ## Note

   Only files associated with the Perl editor will be shown. See the section called “Associating Files with the Perl Editor” for details.

3. Local Host IP: this is the IP-address used by the remote host to connect to your local machine. In most cases the default value is appropriate.

4. Target Host Project Installation Path: the location on the remote machine, where your Eclipse project should be copied to.

5. Port: Port used to transfer data between local and remote host.

6. Create Debug Package: if checked, a ZIP file containing all data to be transfered to the remote machine is created. This file is stored at the location indicated in Debug Package File Path.

7. Press the Debug button.

8. Extract the ZIP file to the location on your remote machine indicated in Target Host Project Installation Path.

9. Start the script **start_epicDB.pl** on your remote machine (it is located in the directory indicated by Target Host Project Installation Path).

10. Enjoy debugging...

Breakpoints

The Perl debugger supports *line breakpoints* and *regular expression breakpoints*. Both types of breakpoints are set on an executable line of a program. If enabled, they suspend thread execution before the corresponding line of code is executed. Regular expression breakpoints additionally extract the regular expression contained in the line of code they are associated with and enable you to debug the regular expression within the RegExp-Plugin.

The following symbols are used to indicate breakpoints:

Status	**Line Breakpoint**	**Regular Expression Breakpoint**
Enabled		
Disabled		
Registered with debugger		

Note

Regular Expressions Breakpoints are still in an experimental state and will at the moment only work for expressions of the type: `Expr1 =~ <delim>regexp<delim>modifiers;` Modifiers are ignored.

Setting Breakpoints

1. In the editor area, open the file where you want to add the breakpoint with the Perl editor.

2. Directly to the left of the line where you want to add the breakpoint, open the marker bar (vertical ruler) pop-up menu and select Add Breakpoint or Add RegExp Breakpoint.

While the breakpoint is enabled, thread execution suspends before that line of code is executed.

Enabling or Disabling Breakpoints

Open the debug view, open the Breakpoints view and use the check box in front of the break point to enable or disable the break point.

Removing Breakpoints

There are two possible ways for removing a breakpoint:

1. Right click on the breakpoint symbol in marker the bar (vertical ruler) of the editor pop-up menu and select Remove Breakpoint.

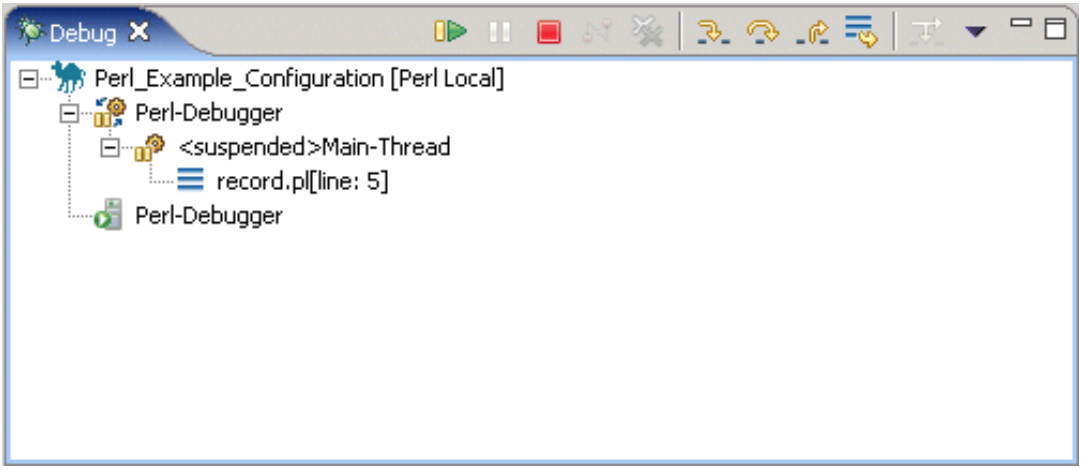
2. Open the debug perspective, open the Breakpoints view, right-click the breakpoint you want to

remove and select Remove.

Views in the Debug Perspective

If any of the views described below is not visible, you can open it using the Window->Show View menu.

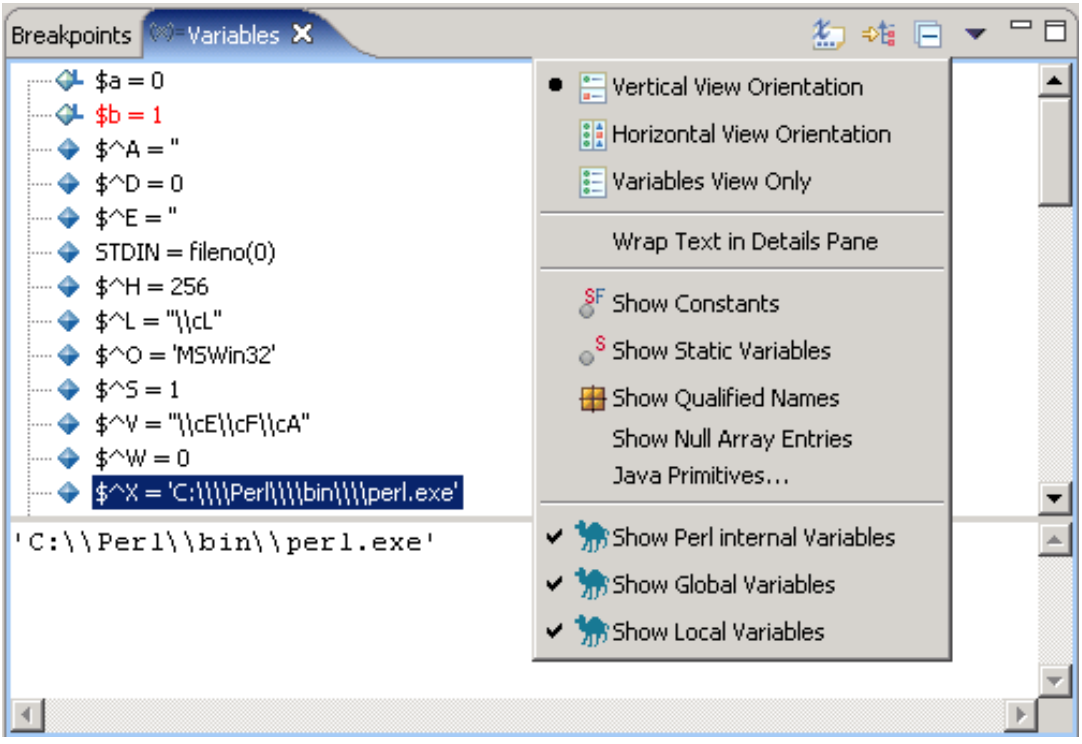
Debug View



This view allows you to manage the debugging or running of programs in the workbench. It displays the stack frame for the suspended threads for each target you are debugging. Each thread in your program appears as a node in the tree. It also displays the process for each target you are running.

If the thread is suspended, its stack frames are shown as child elements. Clicking on a stack frame takes you to the corresponding line in the Perl editor. If necessary, a new editor is opened automatically.

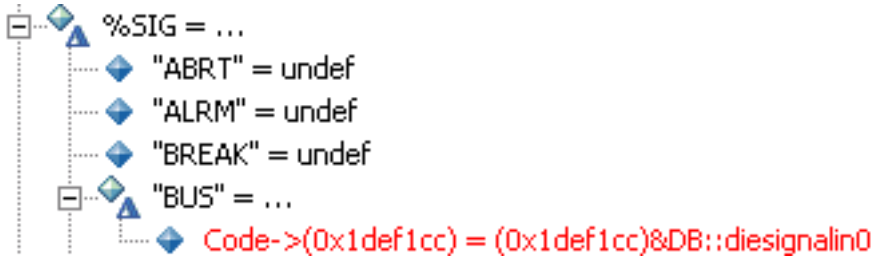
Variables View



When a stack frame is selected, you can see the visible variables in that stack frame in the Variables view. The view shows the value of primitive types. Complex variables can be examined by expanding them to show their members.

Global variables (including Perl internal variables) are marked with an icon, local variables with an icon.

If the value of a variable has changed since the last execution step, it is displayed in red. If changes in inner levels of complex variables have occurred, the path to the changed variables is indicated via delta symbols:



This makes it possible to see these kinds of changes even if variables are collapsed.

Customizing the Variables View

You can customize the presentation with the configuration menu:

Show Details Pane

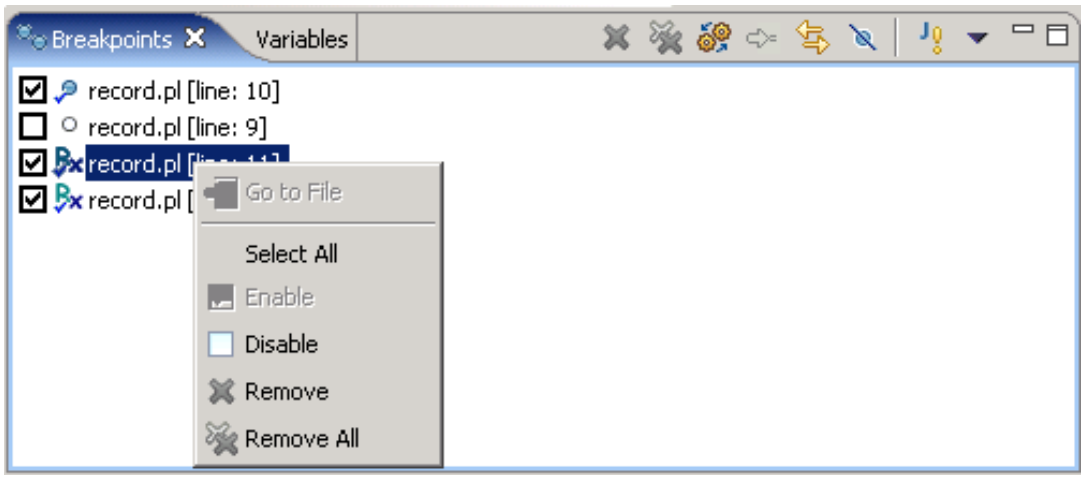
To show the detail pane select Vertical View Orientation or Horizontal View Orientation. Select Variables View Only to disable it. The details pane shows the value of primitive variables (especially useful for string variables).

Select Variables to Display

The variables view allows to select the following types of variables for displaying by checking the corresponding menu entry:

- Perl Internal Variables: these are variables provided by the Perl interpreter like `$_`, `@INC` etc.
- Global Variables: variables visible from everywhere in your program
- Local Variables: variables declared with the keyword `my`, more correctly called "lexical" variables in Perl. To show lexical variables, you need to install the Perl PadWalker module. The PadWalker module has some problems which influence viewing of local variables.

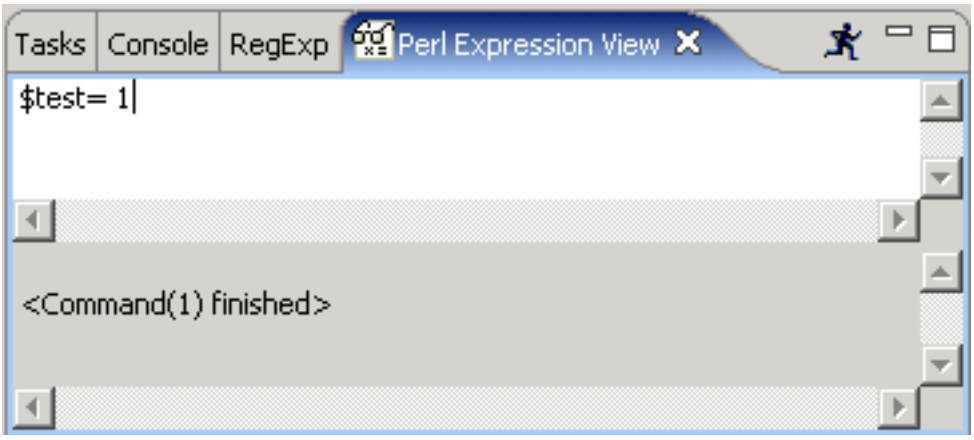
Breakpoints View



The Breakpoints view shows all breakpoints (see the section called “Breakpoints”), their state and location.

By selecting one or more breakpoints and invoking the context menu, you can enable, disable or remove these breakpoints.

Perl Expression View



The Perl Expression View allows you to execute any valid Perl code within the current context of the program executed in debug mode.

1. Open the view (Window->Open View->Other, then EPIC->Perl Expression View).
2. Enter the code to execute.
3. Press the symbol.

Stepping Through the Execution of a Perl Program

When a thread is suspended, the step controls can be used to step through the execution of the program line-by-line. If a breakpoint is encountered while performing a step operation, the execution will suspend at the breakpoint and the step operation is ended.

Step Over

1. Select a stack frame in the Debug view. The current line of execution in that stack frame is highlighted in the editor in the Debug perspective.
2. Click the Step Over button ( ) in the Debug view toolbar, or press the **F6** key. The currently selected line is executed and suspends on the next executable line.

Step Into

1. Select a stack frame in the Debug view. The current line of execution in that stack frame is highlighted in the editor in the Debug perspective.
2. Click the Step Into button ( ) in the Debug view toolbar, or press the **F5** key. The next expression on the currently selected line to be executed is invoked, and execution suspends at the next executable line in the method that is invoked.

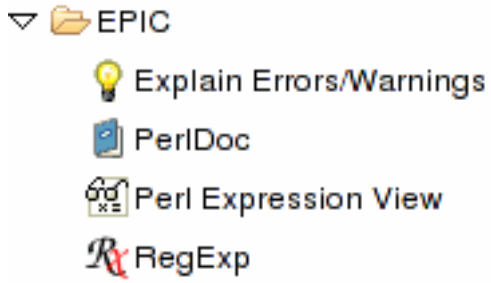
Run to Return

1. Select a stack frame in the Debug view. The current line of execution in that stack frame is highlighted in the editor in the Debug perspective.
2. Click the Run To Return button ( ) in the Debug view toolbar, or press the **F7** key. Execution resumes until the next return statement in the current subroutine is executed, and execution suspends on the next executable line.

RegExp Plug-in

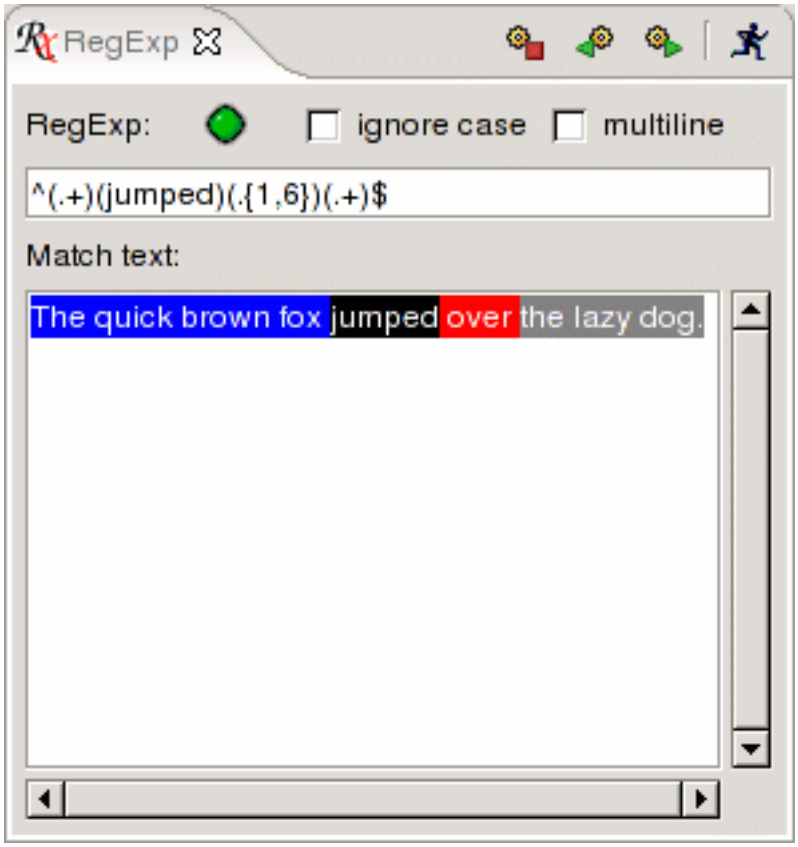
Enabling the RegExp View

To display the RegExp view, select Window->Show View->Other... from the Eclipse menu and select the EPIC->RegExp view from the list.



Using the RegExp Plug-in

The RegExp plug-in is a small tool to debug regular expressions.



To check if a regular expression is valid, press the Run icon. If the regular expression matches the text, it will be signaled by a green icon. If the regular expression contains brackets, the matching character groups in text will be colored.

Regular expression shortcuts are available via the context menu.

Debugging Regular Expressions

The Single Step feature allows for a step by step inspection of the regular expression.

If no groups `(...)` are defined by the user, the RegExp Plug-in tries to use logical blocks for matching, otherwise the already existing groups are used.

The following buttons are provided:

- Reset (clears all color markers)
- Step forward
- Step backward

Known Bugs & Problems

Debugger

Problems Showing Local Variables

The EPIC debugger uses the PadWalker module to show local (or lexical) variables. PadWalker 0.10 does not display the values for lexical variables declared on top-level (outside of functions and blocks) correctly if used in debugger mode. To get around this problem, put your code within a function:

```
dummy();

sub dummy
{
    <YOUR CODE>
}
```

References

Part of this document is taken from the official Eclipse documentation provided by the Eclipse project and IBM.

EPIC uses the public domain ANTLR 2 library developed by Terence Parr and others in the ANTLR project [http://www.antlr.org].