Build Perl applications with Eclipse

Use EPIC to build, edit, and develop

Skill Level: Intermediate

Martin Brown (mcb@mcslp.com)
Freelance Writer
Consultant

17 Jan 2006

The EPIC project lets developers build, edit, and develop Perl-based applications using the Eclipse IDE. In this tutorial, we look at the EPIC plug-in, how it can be used to develop Perl applications, and how it can be integrated into existing development processes.

Section 1. Before you start

For many Perl programmers, the typical development environment is probably an editor like Emacs coupled with a command-line environment. The problem is you tend to spend most of your day switching between Emacs and the command line, and it gets worse if you are doing Web development, as you have to switch between Emacs, your command line, and your Web browser as you write, execute, and check logs for information. Surprisingly, there are few IDEs that have really captured the Perl programmer's imagination over the years, which is why EPIC and Eclipse fills such a void.

EPIC is a complete plug-in suite that supports a new "nature" within Eclipse. The EPIC plug-in incorporates extensions to the editor so that it understands Perl structure and layout. The plug-in also adds additional views and interfaces to your code, and related information enables you to view documentation, execute your Perl applications, and debug them.

About this tutorial

This tutorial will look at the basics of the EPIC plug-in before moving on to an examination of the EPIC system using a real-world example, developing a small module and script entirely within Eclipse that supports RSS parsing. You'll use this as an opportunity to examine other areas, such as the integration with Perldoc, code folding and refactoring -- all of which can make the application development process run more smoothly. By the end, you will have a good understanding of how the EPIC plug-in can be used to develop your Perl applications within Eclipse.

Prerequisites

You will need the following tools before you can make good use of EPIC:

- Eclipse V3.0 or 3.1
- Java technology V1.3, 1.4, or higher
- Perl V5.8.6 or higher. A version of Perl is included with most UNIX® and Linux® installations, and Mac OS X. On Windows®, use ActivePerl.
-
    1. PadWalker Perl module at CPAN.
    2. EPIC (Eclipse Perl Integration), an open source Perl IDE for the Eclipse platform at SourceForge.

Section 2. Why use an IDE?

This section will examine the reasons behind using an Integrated Development Environment (IDE) over more traditional methods.

Tasks during development

Before looking at the reasons behind using an IDE over more traditional methods, it is worth considering all the tasks you tend to perform when developing with a scripted language. There are differences from the typical compiled language. You generally don't need to compile the source into the final application, but some of the tasks remain constant:

- **Writing the code** -- This includes getting the format right so it is readable.
- **Checking the validity** -- Although you won't compile the code, there is still a formal structure, and you can still introduce bugs and problems into the code that can be identified by running some simple checks on the code.
- **Access documentation** -- No matter how good a programmer you are, it is almost inevitable that you will need to look up some aspect of documentation.
- **Write comments/documentation** -- Adding commentary to your code makes it readable, and adding documentation as you go helps to make it portable.
- **Executing the code** -- Often, perhaps more so with scripted languages, you tend to try out the code you are writing.
- **Debugging** -- Any problems during execution will normally need to be investigated through a determined period of debugging.

How you perform each of these tasks will depend on what environment you use. Let's look at the typical non-IDE based environment first.

Existing environments

Ask a typical Perl programmer what he uses for editing and working with Perl scripts, and it's likely that he will simply return the name of his favorite editor -- perhaps vi, maybe even Notepad. At a push, he might be using a more extensive and intelligent editor like Emacs or oXygen that provides built-in markup, highlighting, and intelligent formatting.

The ability to use a standard editor and execute the program directly through the command line is one of the major benefits and advantages of scripting languages like Perl and other scripted languages like Python, PHP, and Ruby.

There are some obvious benefits to the editor approach. For example, you can easily edit and create the scripts pretty much everywhere, with or without a specific editor, so there are no limits on when and where you can program.

Some aspects, though, are less than perfect. Looking up documentation, for example, often needs to be handled in another application or terminal window. Execution of the application will also require dropping to a shell or terminal to execute. Also there is no management of the project as a whole. A typical Perl project will consist of Perl scripts, modules, and probably other data, as well, like XML files or other data sources. They may all exist in the same folder, but their relationship to each other and their significance might be less clear.

IDE benefits

The key element to any IDE is directly related to that first word: integrated. All the steps outlined in Tasks can generally be performed within an IDE without ever having to leave or switch from the application.

For example, code can be written and automatically formatted. Errors and typos in your code can be highlighted as you type, and you can use the hot links to documentation to verify the functions or modules you need to use, without separately looking up that information.

Usually, you can also execute -- and monitor the execution -- of your application so you determine whether it works correctly, and you can debug its operation in the process.

You can use the information generated, and output during the execution and debugging process in your application directly. For example, generated errors and warnings will provide a link that will take you directly back to the appropriate line of your source code.

Overall, the main benefit of the integrated system is to save you time. You no longer have to spend time switching between applications, or finding and locating the code that generated problems. All of the information is not only highlighted but linked and accessible, making it easier to work within the code structure.

Many of these abilities are unfamiliar to the typical script programmer who is used to the simple editor approach. But it's time to move on to a more coherent environment.

Section 3. Installation and setup

Let's take a look at how to install the EPIC plug-in so you can use the IDE features to write Perl applications.

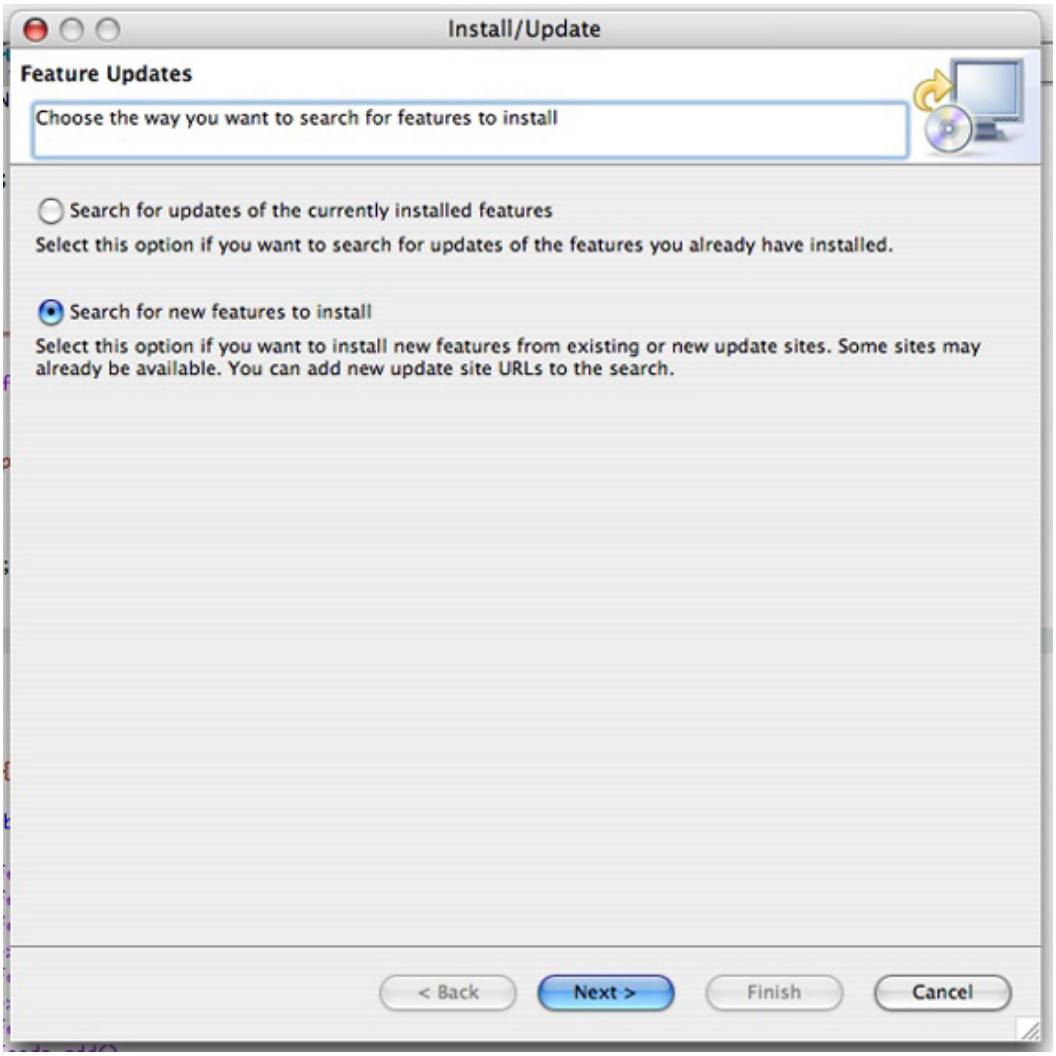
Installing the EPIC plug-in

Before looking at EPIC, you need to install the plug-in. Before you get there, you will obviously need a Perl interpreter. I'm using Mac OS X, which, being based on BSD, comes with the UNIX-based Perl interpreter that you might otherwise have access to on any other UNIX/Linux host. On Windows, you can use the ActivePerl interpreter (from ActiveState) or the Perl interpreter provided as part of the Cygwin system. I

prefer ActivePerl, but the choice is yours.

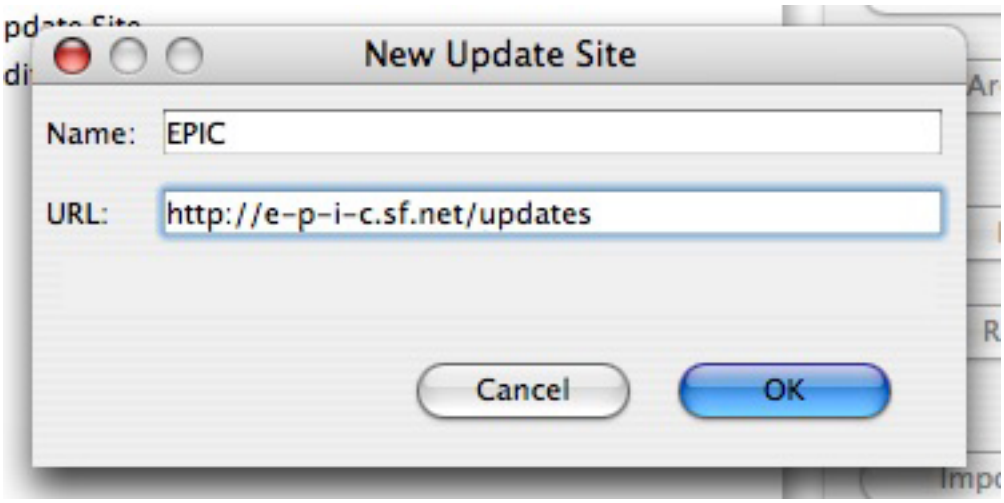
Once you have a Perl interpreter handy, use the Software Update component of Eclipse to install it:

1. Choose **Help > Software Update > Find and Install**, and you'll be presented with a window for configuring downloads, as shown in Figure 1.
Figure 1. The Software Update window



2. Click Next.

3. Click New Remote Site. Give the new site a name and enter the source URL (`http://e-p-I-c.sf.net/updates`"), as shown in Figure 2.
Figure 2. Creating a new site



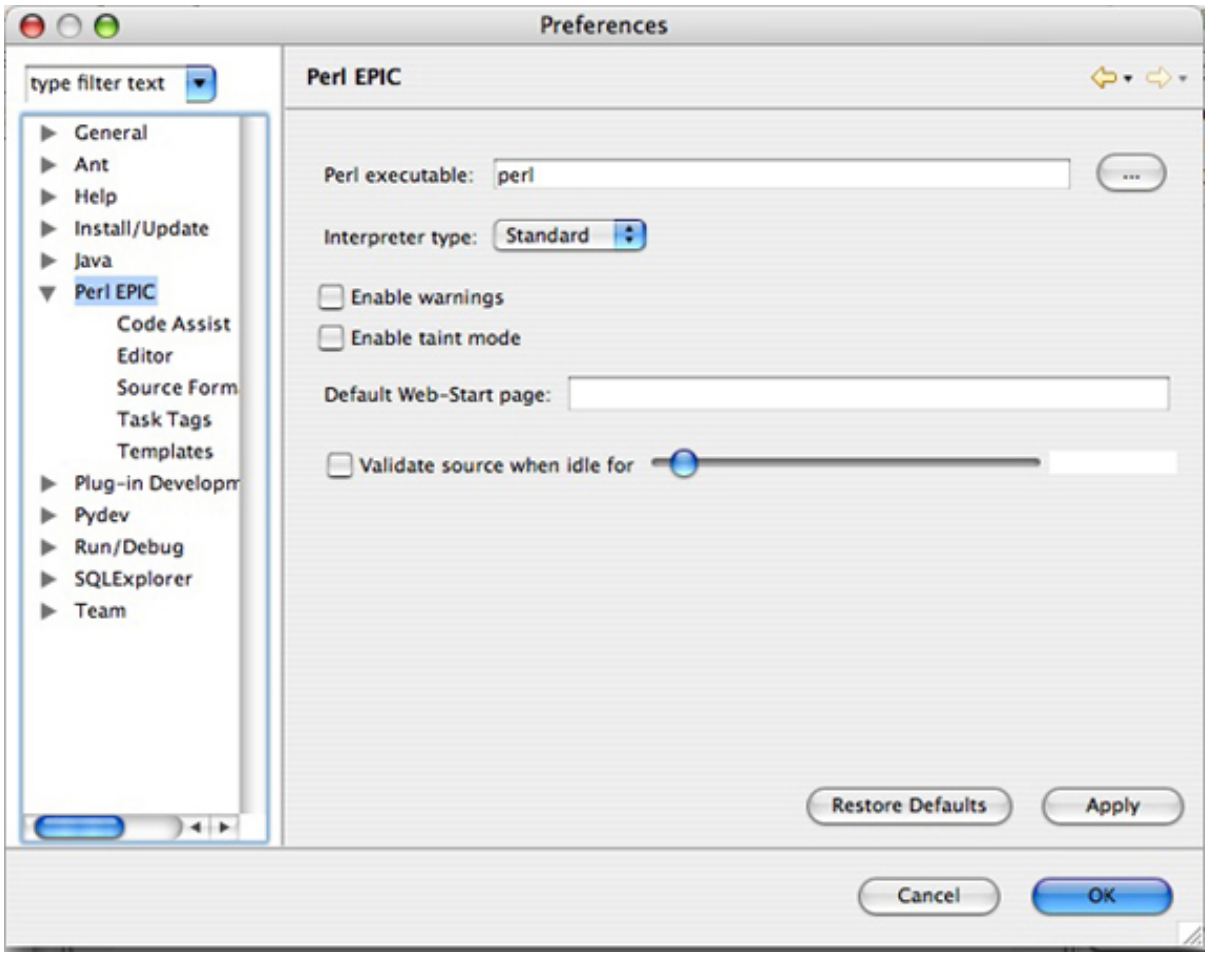
4. Then follow the on-screen process to find, select, and install the plug-in.

Quick configuration

You'll take a closer look at the preferences and their effect on how you use and work with the EPIC plug-in later, but you can benefit from a brief look at the preferences panel to get an idea of the sort of facilities that are available when using the plug-in.

To access the preferences for EPIC, open the standard Eclipse Preferences Window and choose the Perl EPIC folder from the navigation panel on the left, as shown here in Figure 3.

Figure 3. EPIC Preferences



The preferences are split into sections, starting with the general preferences for the plug-in:

- **General Preferences** -- Sets the location of the Perl executable, interpreter, execution model, and the period to wait before the code is checked in the background.
- **Code Assist** -- Sets the characters that trigger auto-completion.
- **Editor** -- Sets editor preferences, including the colors used for highlighting different components, annotation formats, and so on.
- **Source Formatter** -- Sets formatting preferences.
- **Task Tags** -- Sets task tags, which are quick notes that take you back to a specific location.
- **Templates** -- Sets up templates of code that can be inserted directly into your code to speed development time.

When these options affect the way you work, I'll mention how to adjust the action in

this tutorial. We'll also look at some specific elements, such as task tags and templates, in their own sections later.

Windows notes

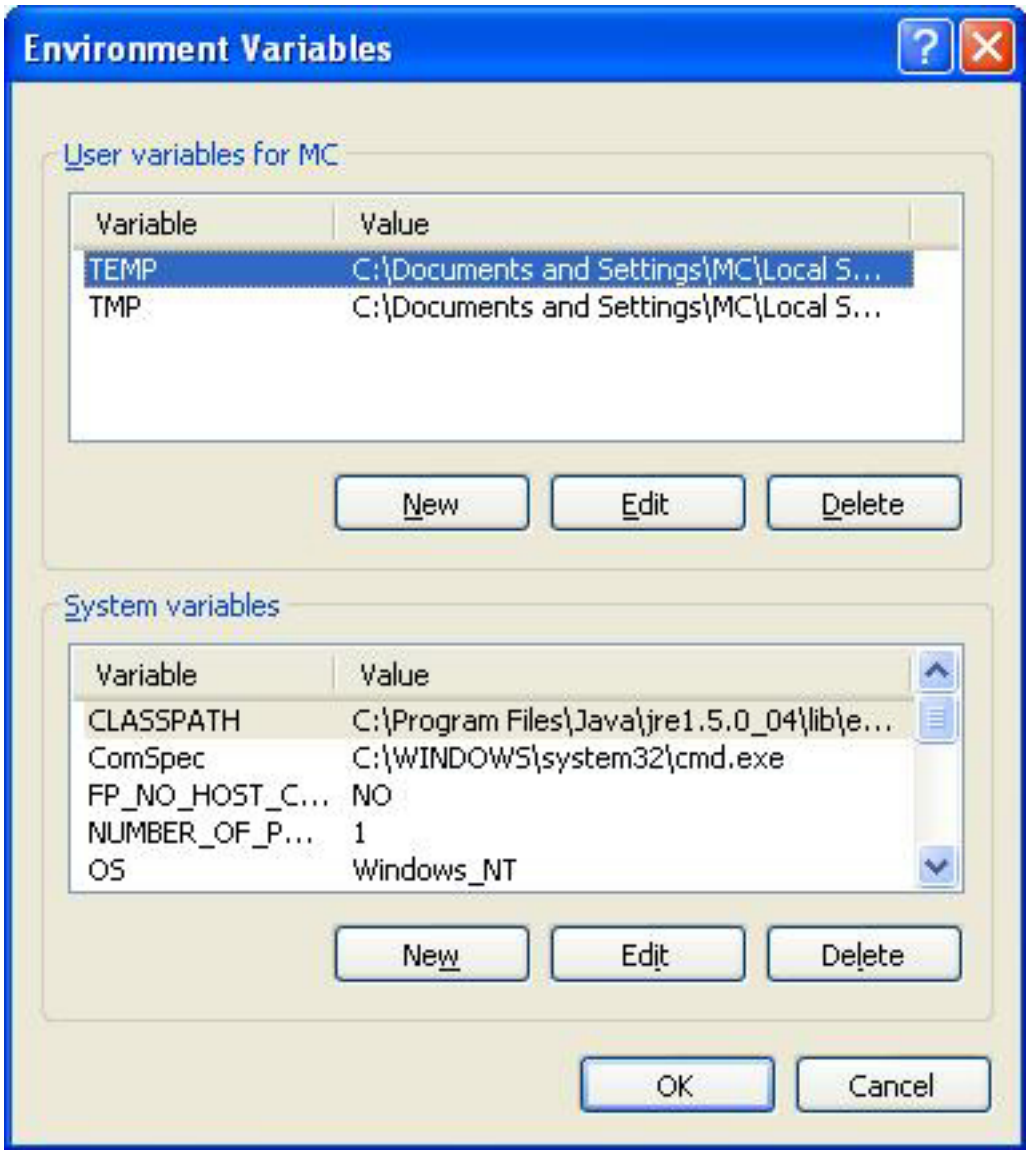
When using the EPIC plug-in within Eclipse under Windows, there are some tricks that will improve your interaction between components.

If you are using ActiveState's ActivePerl distribution, change the Perl executable (as set in the General Preference panel) to the wperl.exe executable. This will prevent a command prompt window being displayed each time the code is being checked. It is also a good idea (but not essential) to add the Perl binary directory to your path. It should have been added automatically when ActivePerl was installed.

If you are using the Cygwin version of Perl, ensure that the mount command, part of the standard Cygwin installation, is available through your system path. You can verify this by checking the values of environment variables. To do this:

1. Open the System Control Panel (usually in **Start > Control Panels > System**, or right-click on My Computer and select Properties).
2. Switch to the Advanced panel.
3. Click Environment Variables. You should be presented with a window like that shown in Figure 4.

Figure 4. System and user environment variables in Windows



Check the value of the PATH variable. If the Perl or Cygwin binary directories are not listed, add them to the path value. Individual directories are separated by a semicolon.

Section 4. Creating projects and files

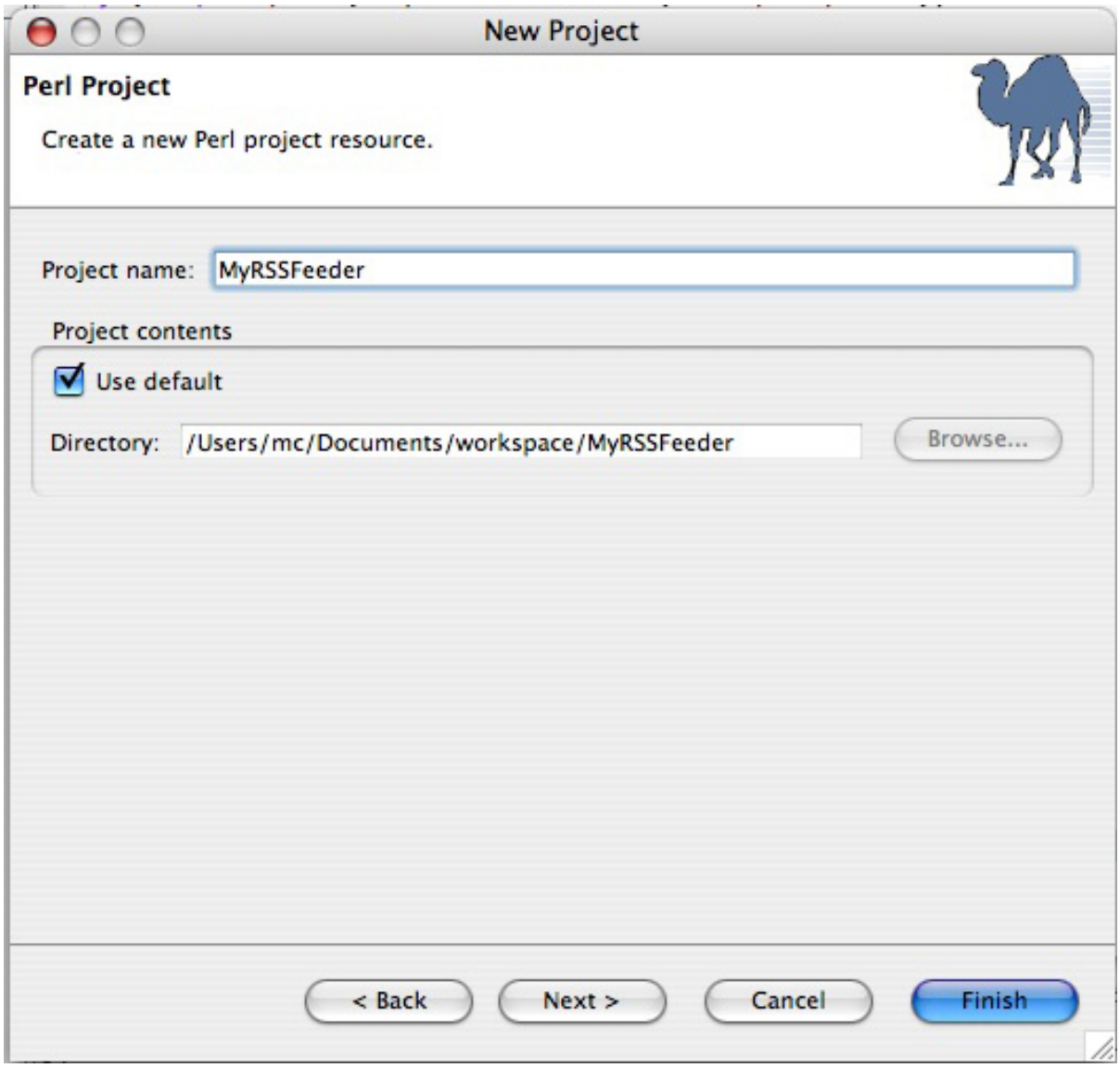
To write Perl applications within Eclipse using the EPIC plug-in, you need to understand the roles of the Perl project and the Perl file.

Creating a new project

Let's create a new Perl project. Because EPIC provides a new nature, you can create a new project to build your Perl application. For this demonstration, you'll be building a Really Simple Syndication (RSS) application that will download an RSS file from the Internet, parse it, and dump a summary of the information. You'll then extend this basic functionality.

You can do all of this by creating a new project to contain your RSS project files. Create a new project by selecting it from the list of available project types. Choose **New > Perl Project**, or **New > Other** and select Perl Project from the list. You can see the resulting window in Figure 5.

Figure 5. Creating a new Perl project



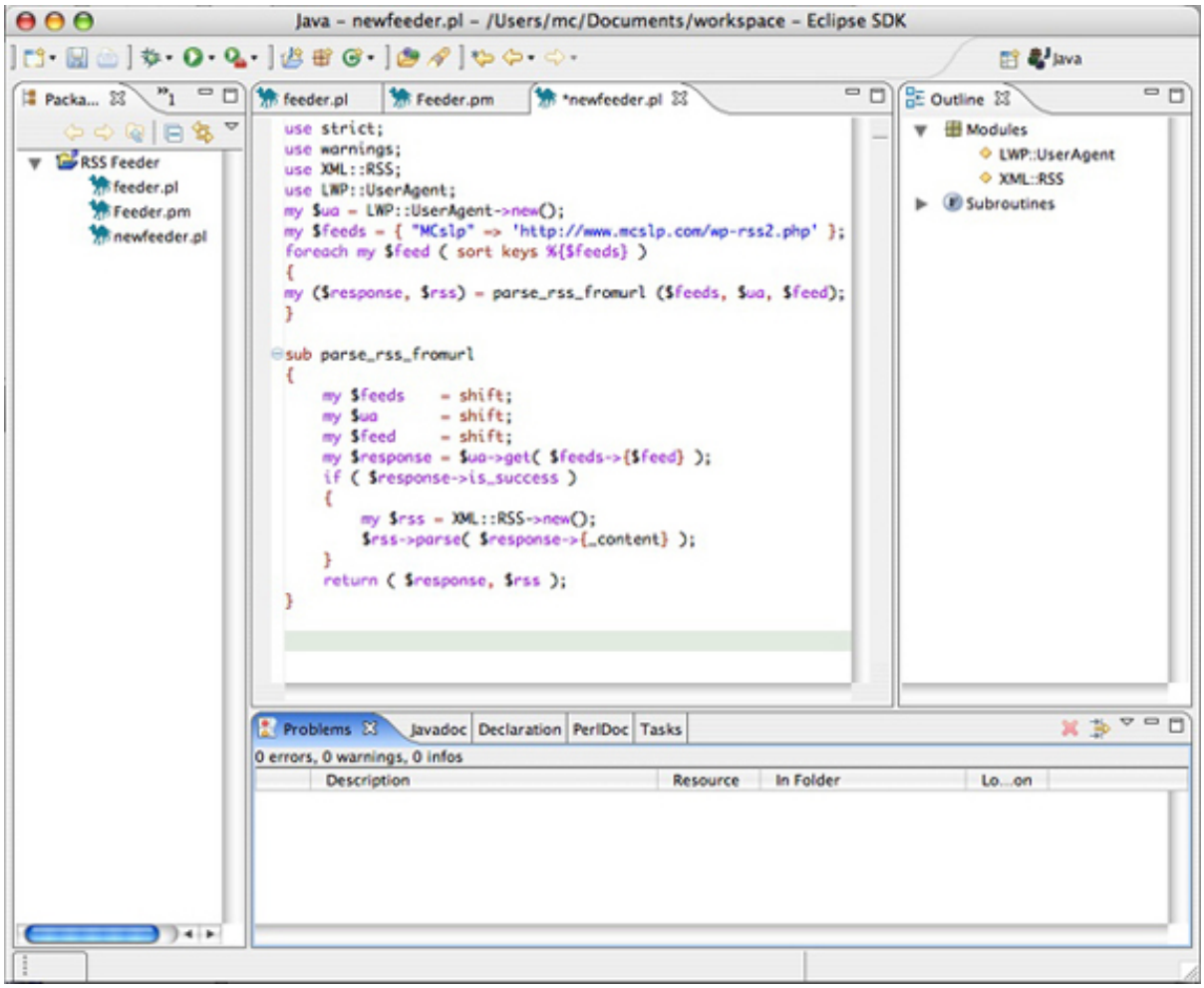
Give the project a name (RSS Feeder) and specify the workspace for the project, or simply use the default workspace.

Eclipse should change to the Perl perspective automatically when you create a new Perl project. The Perl perspective includes a number of specific panels that will help you as you start to write Perl script.

The Perl perspective

If Eclipse does not automatically switch to the Perl perspective, you can switch to it using **Window > Open Perspective** and selecting Perl from the list of perspectives. You can see an example of this perspective in Figure 6, here with some open and active files and views.

Figure 6. The Perl perspective



You can see from Figure 6 that the perspective includes many different panels (called Views in Eclipse), including:

- **Package explorer view** -- This shows the layout of your project (files, modules, and scripts).
- **Outline view for the current file** -- This shows the list of modules imported and the list of functions defined within the current file.
- **Standard editor/file interface** -- This will show the source individual files in the project.
- **Tasks view** -- This shows a list of registered tasks.
- **Console view** -- This is the standard output from your application.
- **Problems** -- This view highlights and provides links to errors in your code within the current project.

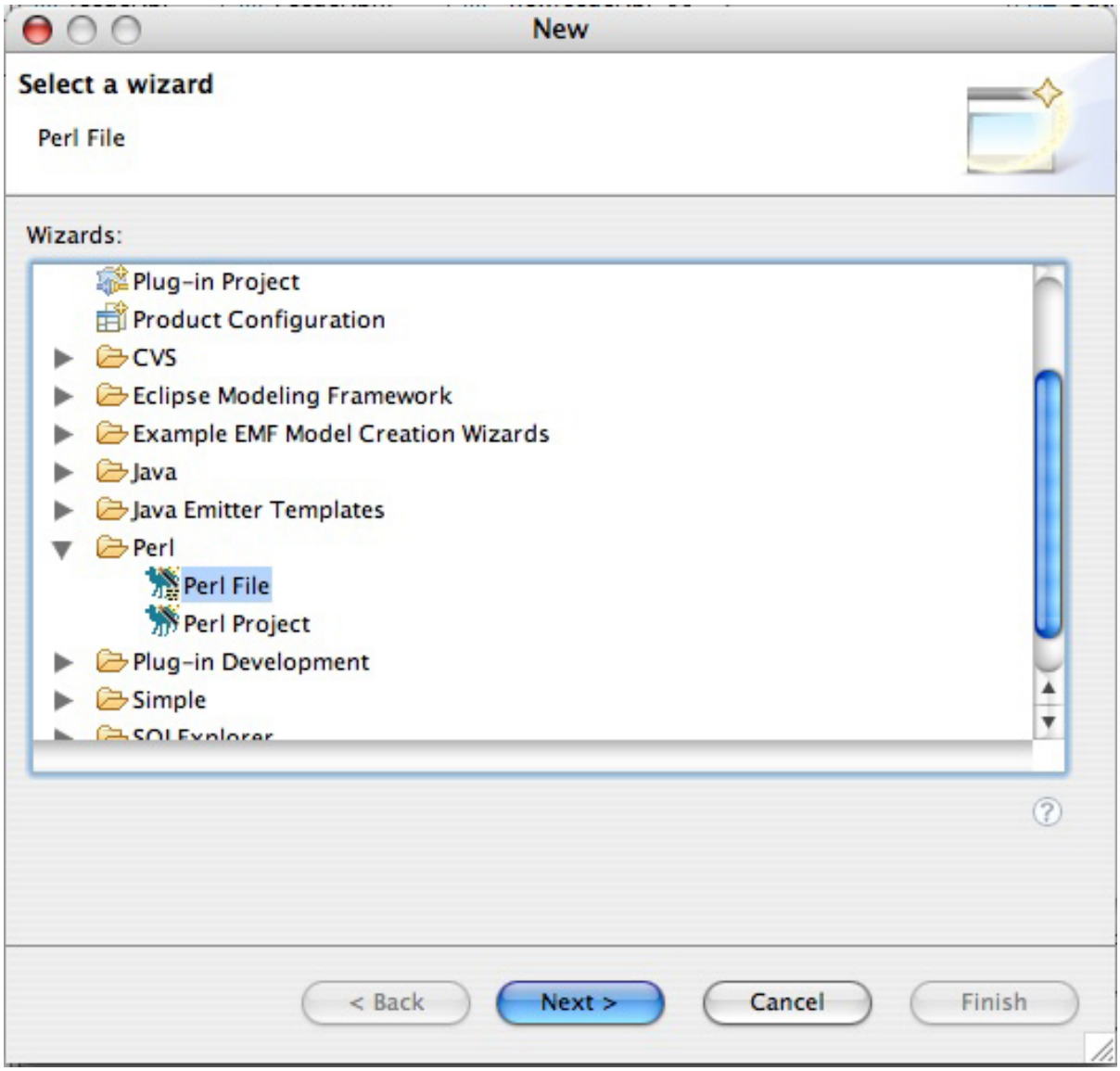
You'll be using the information contained in most of these views within this tutorial. Remember that you can add and remove views at any time by selecting Show View from the Window menu, and either selecting a view directly or choosing the Other menu item and choosing from a more extensive list.

Creating a new file

Once you've created a new project, the first job is to create a new file. Files within EPIC can be either scripts or Perl modules. As far as EPIC is concerned, there is no difference between the two, although of course Perl treats them differently. Because EPIC doesn't specifically differentiate between the two, you have to rely on the file extension to differentiate between the files in a given project.

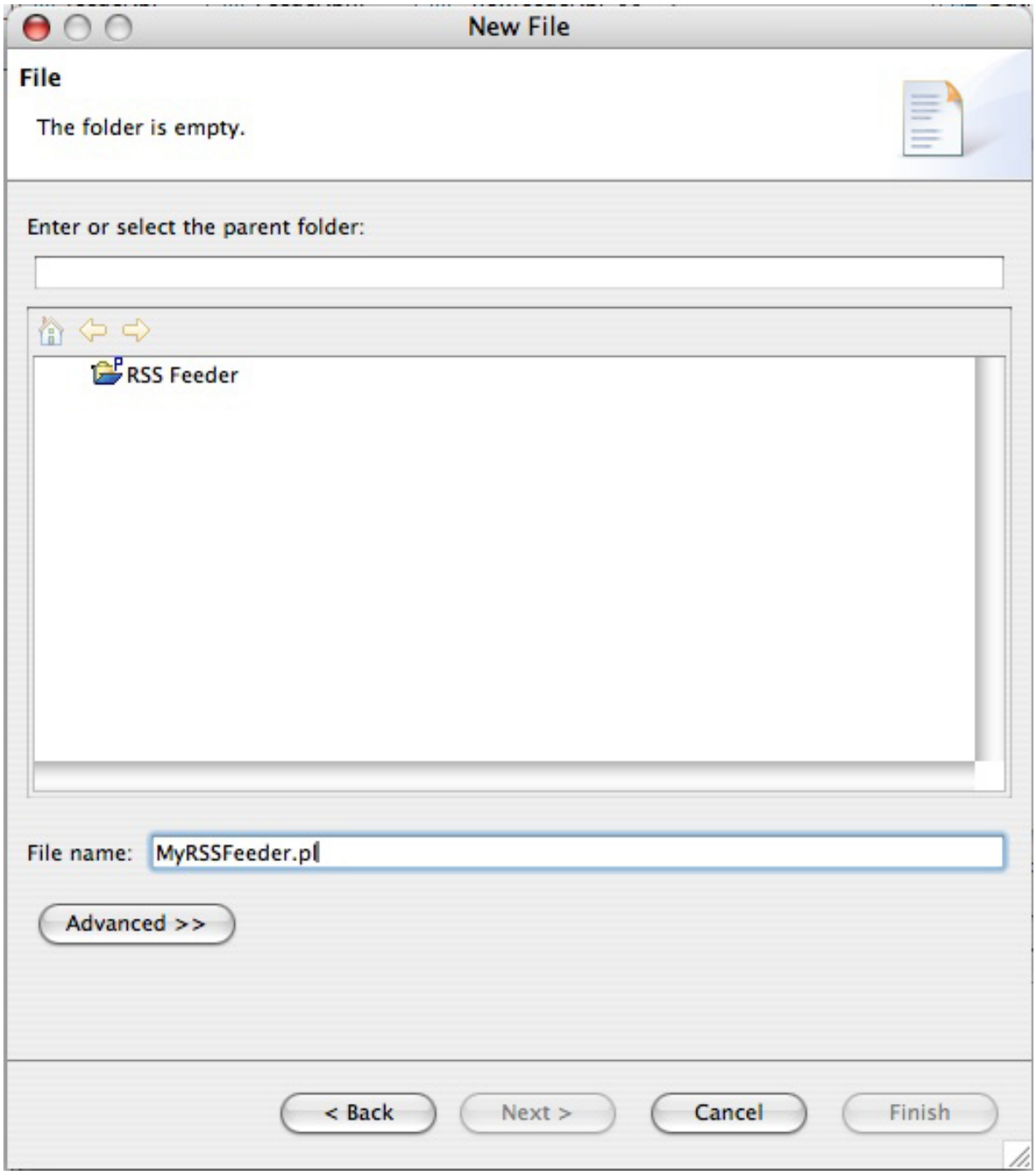
You can do all of this by creating a new project to contain your RSS project files. Create a new project by selecting it from the list of available project types. Choose **New > File > Other** and select Perl File from the list, shown in Figure 7.

Figure 7. Selecting a Perl file



You can see the resulting window Perl file properties window in Figure 8.

Figure 8. Setting Perl file properties



You'll need to specify the folder (or project) where the file should be created and the file name. You can optionally associate the file with an existing file on the filesystem by clicking the Advanced button and setting preferences.

Click Finish to create the new file. You'll then be presented with an editor window for the file.

Section 5. Editor features and Perldoc integration

The bulk of any programming endeavor is actually writing the code, so it is no surprise that the bulk of the EPIC functionality is related to improving the environment of the editor.

The Perl editor

Actually generating code is basically a case of typing the code you want to write into the editor. There's very little difference here from any other editor. The primary difference is in the additional functionality that you gain as you use it.

The first feature you'll look at is the syntax coloring. Basically, this colors different elements of the source code (according to the settings) to make it easier to identify components in the code.

For example, if you type the following:

```
use strict;
use warnings;
```

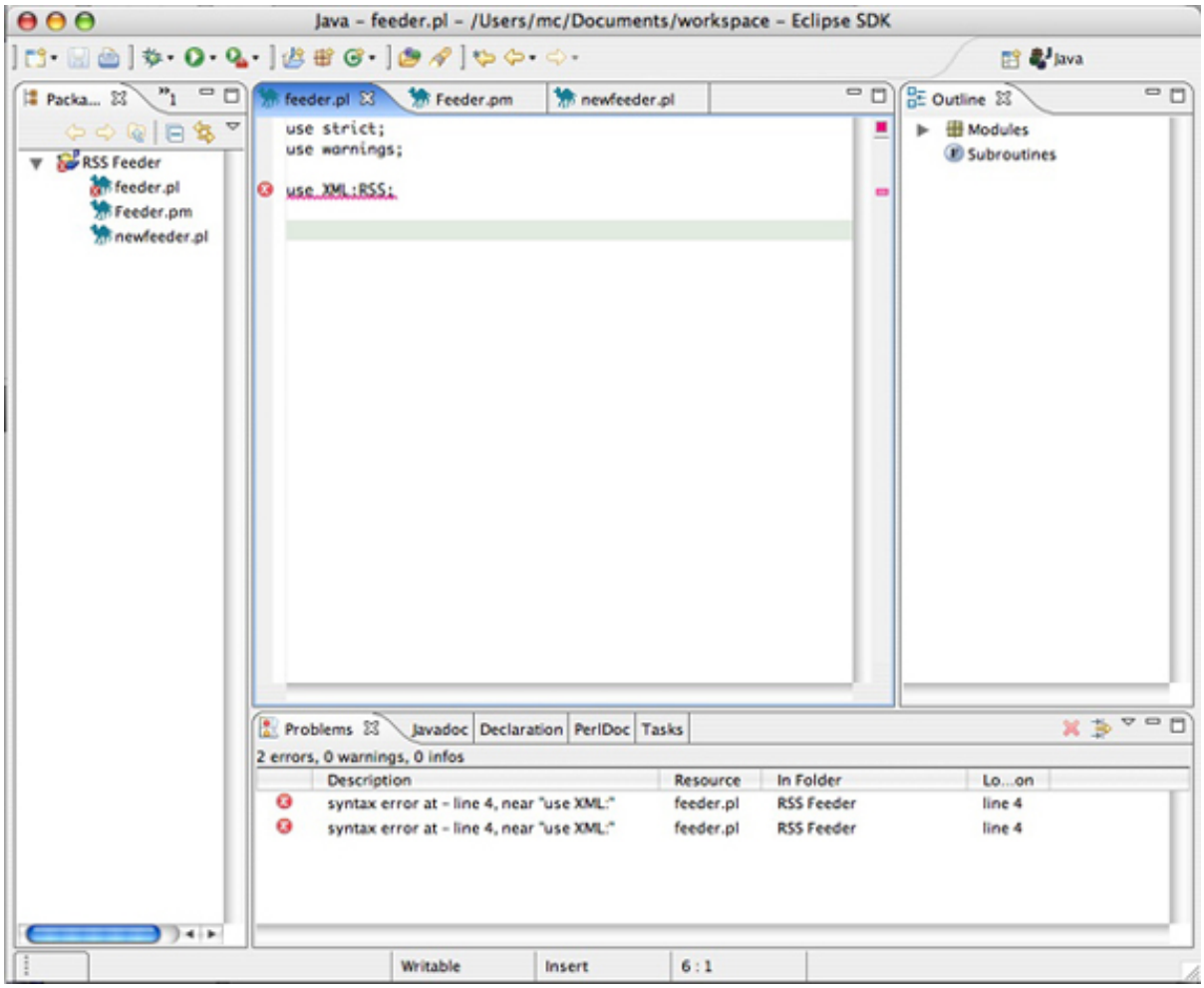
The `use` keyword will be highlighted in a different color to the names of the modules you are loading. Syntax highlighting applies to a wide range of elements, and each has its own color. You can change these by modifying the preferences. Sample elements include functions, arguments, static strings and variables, and key terms like those seen here.

The second feature that is directly obvious during editing is the highlighting of errors in your code that can be identified during the standard checks performed before execution. For example, type the following, exactly as it is here:

```
use XML:RSS;
```

When you pause (the amount of time you have to pause is configurable), the EPIC plug-in checks the format and syntax of your code and reports any errors it finds. The offending error lines are highlighted, and if you switch to the Problems view, you can see a description of the error, and the file and line in which the error occurs. You can see an example of this in Figure 9.

Figure 9. Highlighted errors in the code



In this example, the code is highlighted automatically because that code is wrong. It needs a double colon for Perl to identify the module. To get more detailed help, right-click on the error line and choose Explain Errors/Warnings for a more complete description of the problem and possible resolution.

Note that it is not EPIC checking the validity of the code. It checks the code through Perl, then parses the output. This ensures that the code is valid Perl and also means that pragmas in the code (such as the warnings and strict pragmas defined earlier) will also be applied.

Simple code completion

Although useful, syntax and error highlighting don't speed up the rate at which you can create code, although they do help reduce the amount of errors and typos you introduce.

For improving the speed at which you generate code and its quality, EPIC will

complete common elements for you automatically. For example, if you type:

```
my $feeds = {"
What you actually get is:
my $feeds = {""}
```

EPIC has automatically completed the closing brace and quote.

Add a semicolon to the previous line and type: $ on a new line.

After a short pause, EPIC will bring up a list of possible variables. You've only defined one, but the system can automatically suggest possible variable completions with any of the variable types, scalar ($), array (@) or hash (%).

To continue building your RSS parser, let's populate the hash with some information about some feeds. For the purposes of the demonstration, you'll just define one:

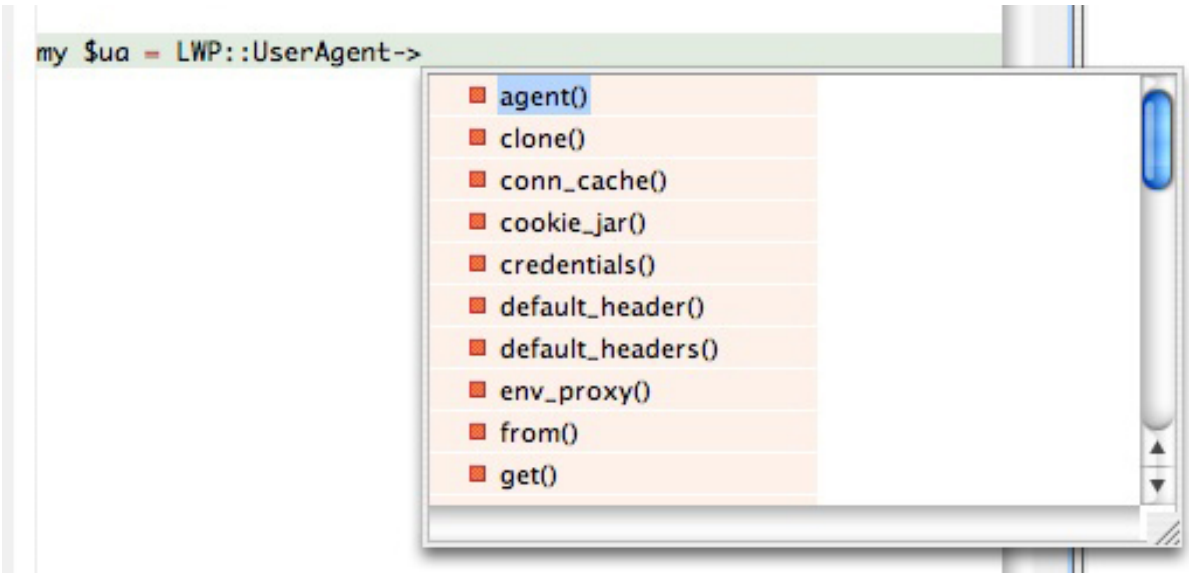
```
my $feeds = {"MCslp" => "http://mcslp.com/?feed=rss2"};
```

To download an RSS feed, you need to use the Lib WWW Perl library (LWP) that handles all of the download for you by whatever method is specified in the URL. You need to create a new UserAgent object. Add the `LWP::UserAgent` module to the start of the script. Then type:

```
$ua = LWP::UserAgent->
```

As you can see in Figure 10, EPIC will provide a list of possible functions you can use to complete the line. This interactive completion works as a sort of combination of documentation lookup and code completion, reminding you about the functions or methods applicable to an object or class and allowing you to select it.

Figure 10. Interactive method completion



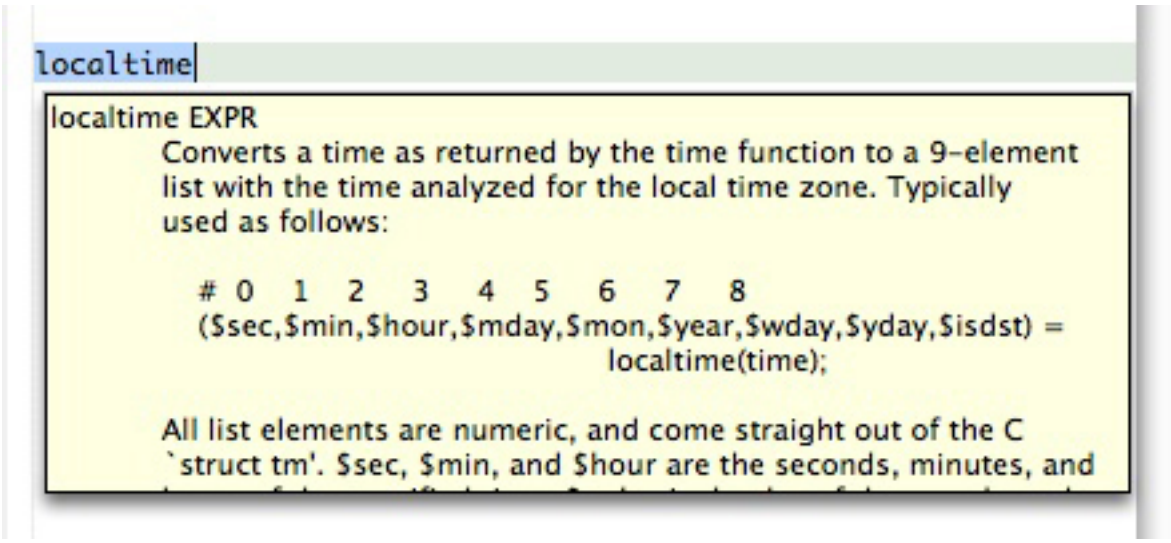
Note that for this system to work correctly on objects, you must create the object using `$ua = LWP::UserAgent->new();`, rather than `$ua = new LWP::UserAgent;`. This is because of the way EPIC determines this information.

Basic Perldoc integration

Let's return to the RSS feeder example. When you parse a feed, you want to make a note of the date and time the feed was processed. The built-in Perl `localtime()` functions returns a list of variables containing the time information. However, despite programming in Perl for 10 years, I can never remember the order of the information.

What I can do with Eclipse and EPIC is type the name of the function, double-click to select it, then hover over it to get a quick view of the contents. Again this information is pulled from Perldoc. I can now see what the return values are. You can see an example of this in Figure 11.

Figure 11. Getting inline documentation help



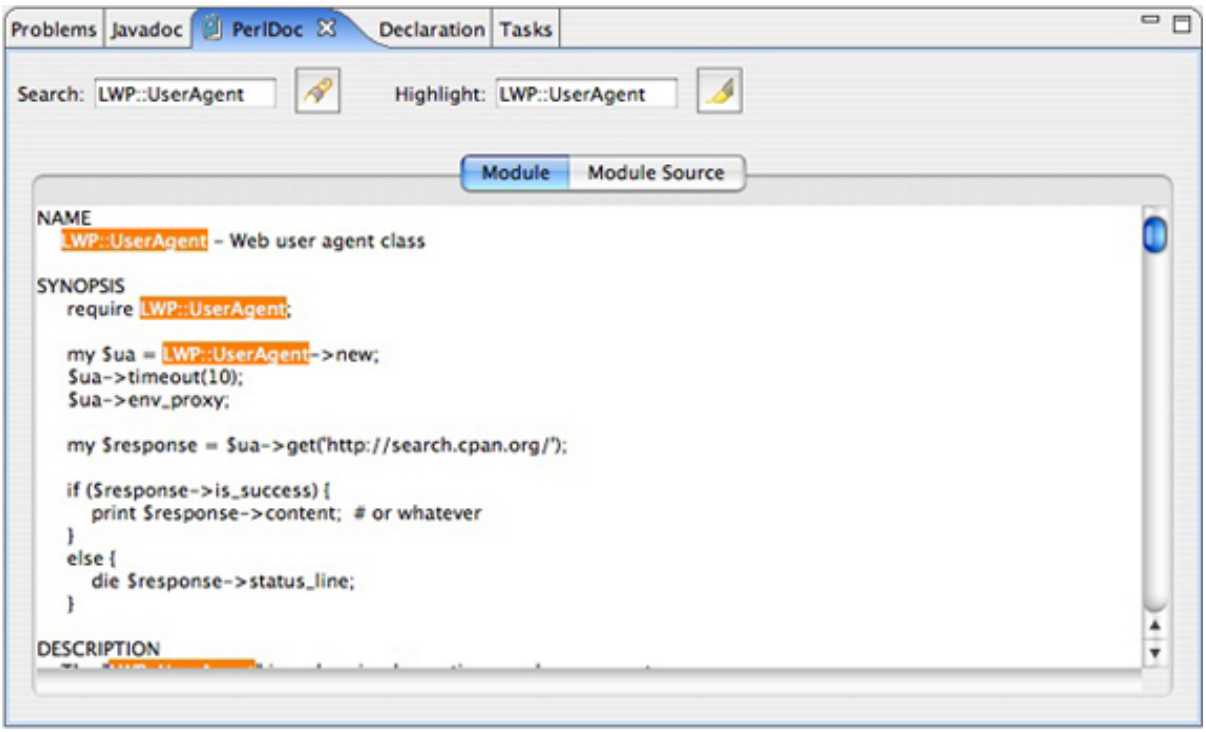
Let's try it by typing `localtime`. Now highlight the word and hover over it.

The information is pulled directly from the Perl documentation (Perldoc). For those not aware, Perldoc is a command-line interface that extracts the documentation embedded into Perl scripts and modules.

Looking up full definitions in Perldoc

You can also use, browse, and search for information from the Perldoc library. If you press Control-Shift-H or right-click within your editor, you'll be prompted for a Perldoc fragment to look for (module, built-in function, or FAQ search). You can also highlight an item and right-click or use the keyboard shortcut. In both cases, the information is displayed in a special Perldoc view. You can see the Perldoc view, here showing the `LWP::UserAgent` module, in Figure 12.

Figure 12. Full Perldoc integration



Again, like the syntax checking, the information is being accessed through the standard Perl tools. Rather than EPIC replacing their functionality, it is using it as a way of obtaining the information you need and which is most useful to you. That means that the same information (and formatting) that are available to you through Emacs or the command line are also available through EPIC and Eclipse.

In terms of Perldoc, that means you have access to the entire range of standard Perl pages (such as perlre or perlfunc), individual access to specific areas (for example, you can get the definition of a single function, as demonstrated with localtime), and you can get information on all of the installed modules within a given Perl installation.

Section 6. Power-user functionality

There are a few additional areas of functionality useful to people working on large projects or who spend a frequent amount of time in Eclipse developing Perl applications.

Task tags

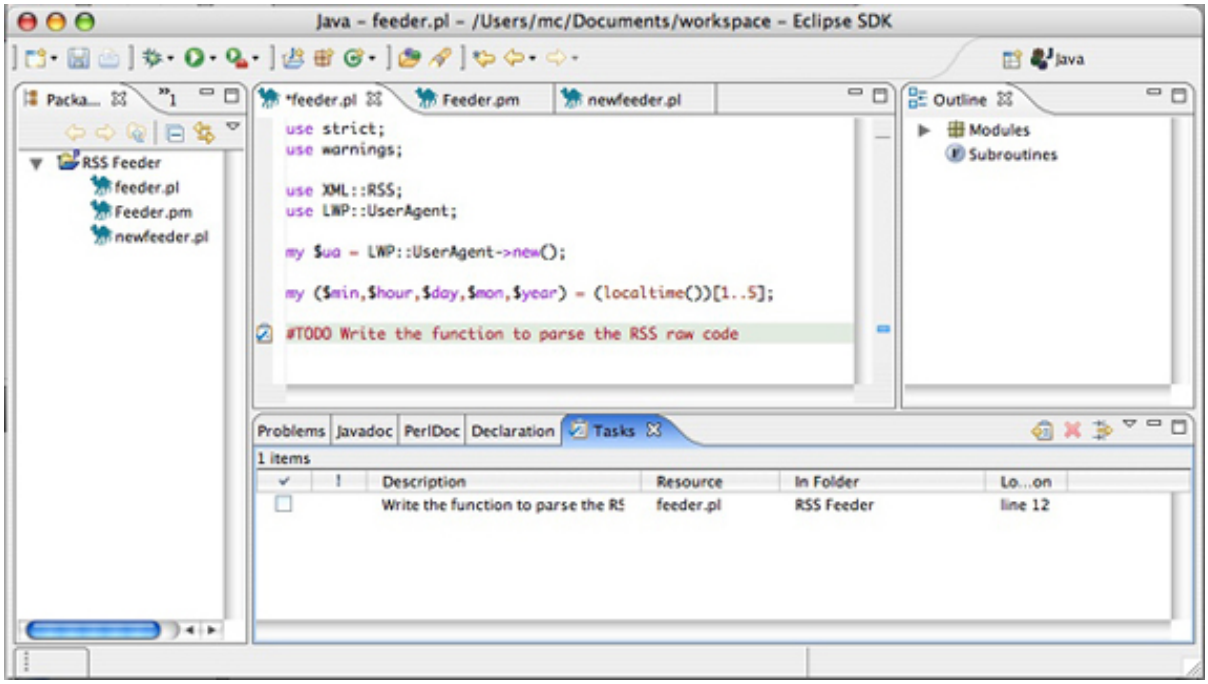
Let's say you need to parse the RSS just downloaded, but you want to write a separate function for that. You don't want to worry about actually writing that particular piece of code yet. Within a standard editor and environment, you might do this in a number of ways -- using a special comment in the source code that you can search and locate during a later session, for example.

EPIC (and actually Eclipse) supports this. You can search for an individual item or piece of code just like any other. However, EPIC and Eclipse extend this functionality by supporting specific task tags. These are comments with a specific format. For example, type the following into your source code:

```
#TODO Write the function to parse
the RSS raw code
```

If you then open the Tasks view (select it or add it through the **Window > Show View** menu item) you can see that you've just added a task, with a description, into a list of things to do. You can see this, both the source and the resulting view, in Figure 13.

Figure 13. Task tags in action

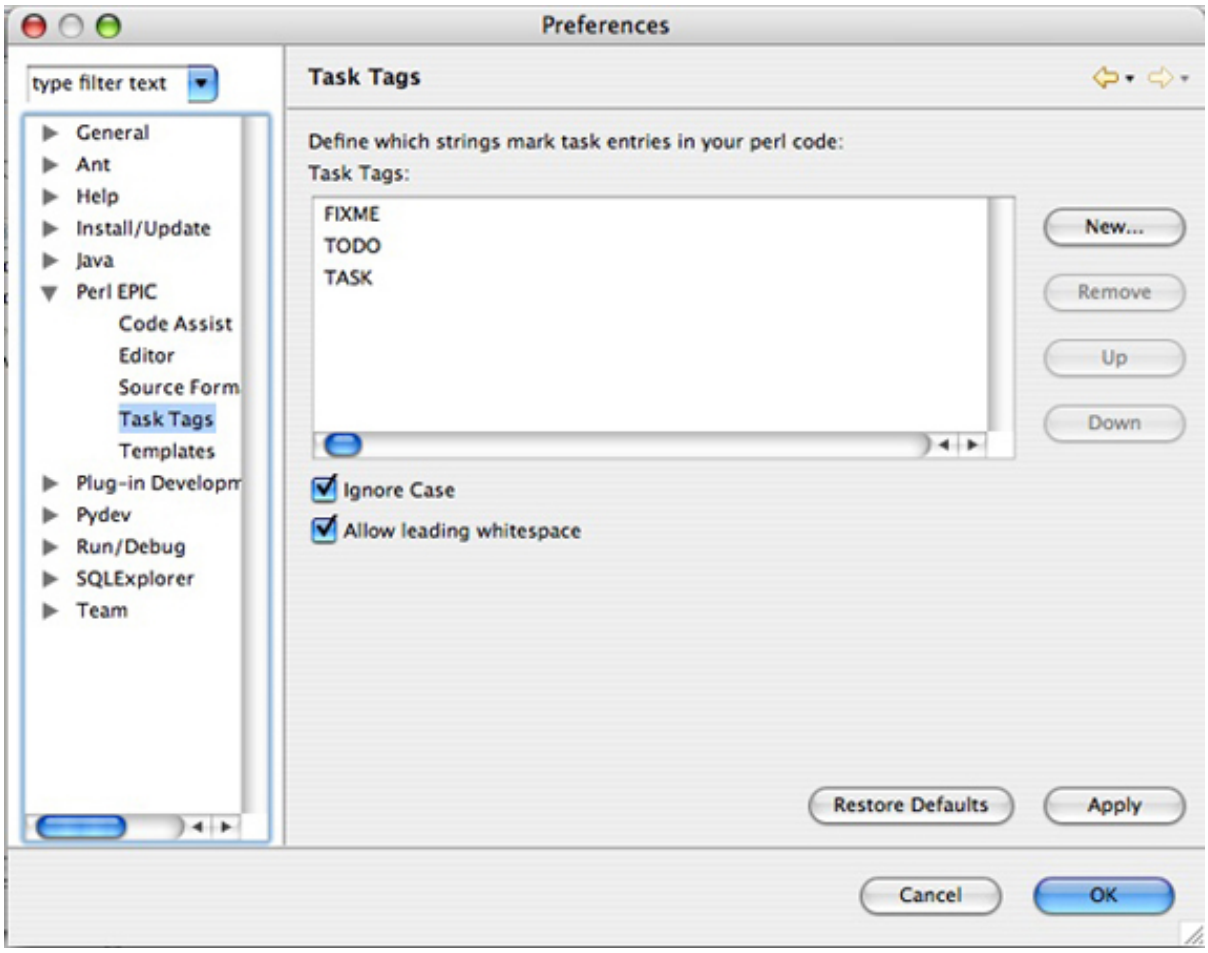


The Tasks view is project-specific, which means that you will be shown such tags and their descriptions across all of the files in your project, not just the current file. This makes it a great way of highlighting tasks and to-dos in your code right across your project.

The standard installation comes with a two recognized tags: TODO and TASK. You

can configure additional task tags using the Task Tags preferences for EPIC. You can see the preferences window in Figure 14.

Figure 14. Setting task tag preferences



To create a new task tag: click New and specify the text to be identified as the tag. Incidentally, since tags work through comments, you should know that `Control-/` (Control slash) comments the current line and `Control-\` uncomments a particular line. If you've selected multiple lines, the operation comments or uncomments each individual line.

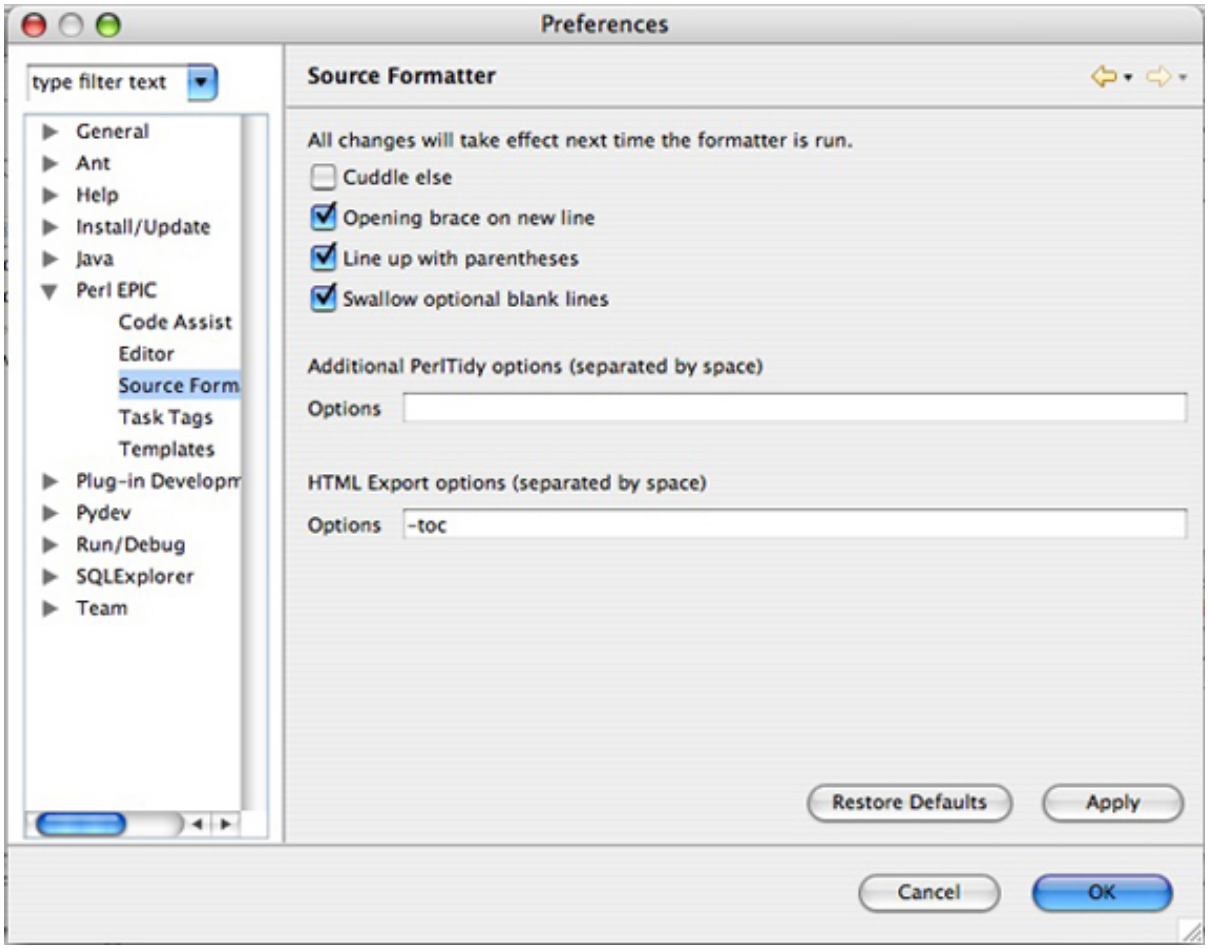
Code formatting

There are lots of ways in which you can format Perl code, and getting the format correct makes a big difference to the readability of the code you are working with. EPIC will automatically format certain elements of your code. You've seen some evidence of this already with the way in which EPIC automatically lays out different components during auto-completion using parentheses and quote marks. You can also force EPIC to reformat the code into a readable format by pressing

Control-Shift-F or by choosing Format from the Source menu.

The exact format will depend on your preferences, set through the Source Formatting pane in the Eclipse preferences (see Figure 15). The formatting is performed by Perltidy, which reads and reformats code according to some agreed-upon standards (mostly those detailed in *Programming Perl* from O'Reilly).

Figure 15. Setting source formatting preferences



The main options affect the formatting as follows. Cuddle else changes the behavior for an `else` statement, shown in Listing 1:

Listing 1. else statement

```
}
else {
to
} else {
```

The opening brace on the new line forces Perltidy to place the opening brace to a block on a new line. See Listing 2 for an example.

Listing 2. Forcing Perltidy to place brace to a block on a new line

```
if ($a == 1) {
becomes
if ($a == 1)
{
```

Lining up parentheses ensures that parentheses line up when split across lines. Finally, Swallow optional blank lines deletes any blank lines that serve no purpose. For example, blank lines before comments are kept, but blank lines after are not. Blank lines are retained between major blocks (for example functions, loops and tests).

Any additional options can be configured by adding arguments to the command line called when Perltidy is executed. See Resources for more information on Perltidy.

Refactoring

Refactoring code adapts the source code without changing its behavior. In the case of EPIC, refactoring can convert a block of code into a function, making the functionality accessible to any part of the code. EPIC will extract the variables required for input to the block of code, determine the outputs, and convert them into arguments and return values respectively. To use refactoring, let's consider the code fragment you use to download and parse an RSS feed URL into an RSS object (see Listing 3).

Listing 3. Code fragment used to download and parse an RSS feed URL into an RSS object

```
foreach my $feed ( sort keys %{$feeds} )
{
    my $response = $ua->get( $feeds->{$feed} );
    if ( $response->is_success )
    {
        my $rss = XML::RSS->new();
        $rss->parse( $response->{_content} );
    }
}
```

If you select the contents of this `foreach` loop (the contents of the entire block, not including the `foreach` statement and the parentheses), right-click and choose the **Refactor > Extract** subroutine, EPIC will convert the code into the following fragment, instead (see Listing 4):

Listing 4. Extract subroutine

```
foreach my $feed ( sort keys %{$feeds} )
{
    my ($response, $rss) = parse_rss_fromurl ($feeds, $ua, $feed);
}

sub parse_rss_fromurl
{
    my $feeds    = shift;
    my $ua       = shift;
    my $feed     = shift;
    my $response = $ua->get( $feeds->{$feed} );
    if ( $response->is_success )
    {
        my $rss = XML::RSS->new();
        $rss->parse( $response->{_content} );
    }
    return ( $response, $rss );
}
```

You can see that EPIC has determined the variables it needs in the new subroutine and the response values. The actual code remains the same, but the entire block has been rebuilt into a new subroutine. Also note that the `foreach` loop now contains the required subroutine call and arguments.

Section 7. Templates and modules

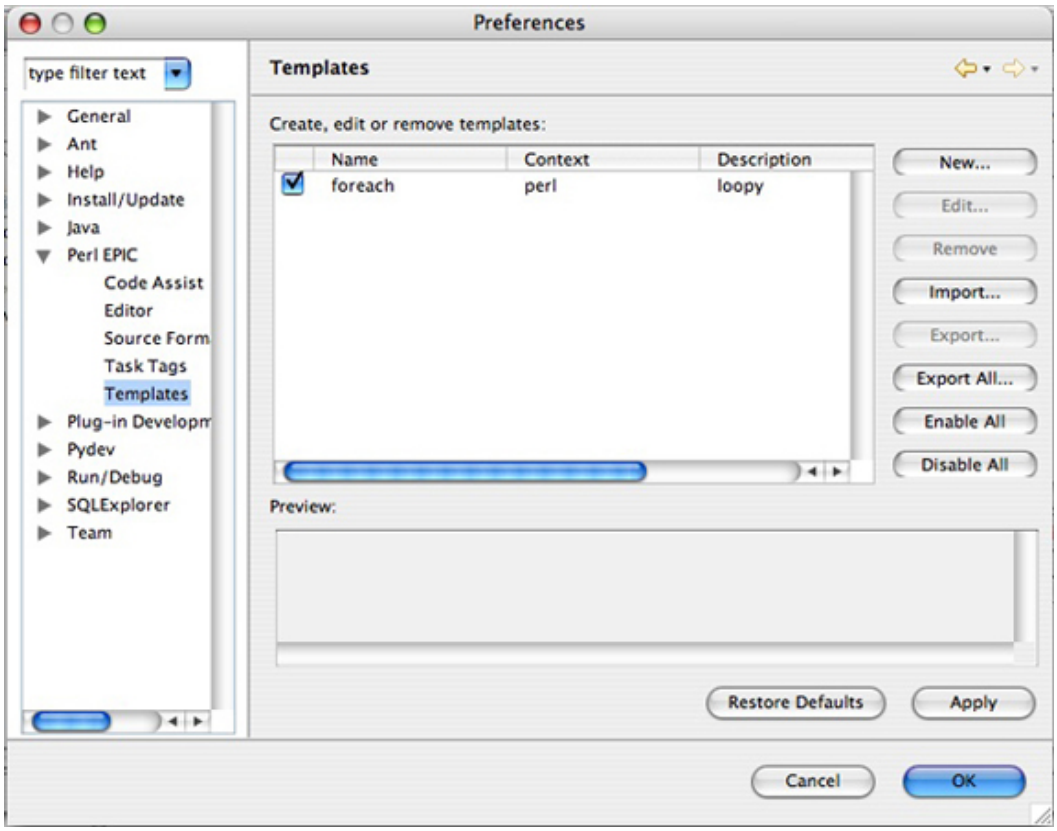
A lot of source code is based on similar elements. For example, although subroutines may be different, the same basic content remains the same: You need the same keywords, parentheses, and the same structure for extracting subroutine arguments. Templates enable you to quickly insert this into your code.

Creating templates

Templates are defined through the Templates pane in the Eclipse Preferences. At face value, a template is exactly that: a fragment of code that can be inserted quickly and easily into your source code.

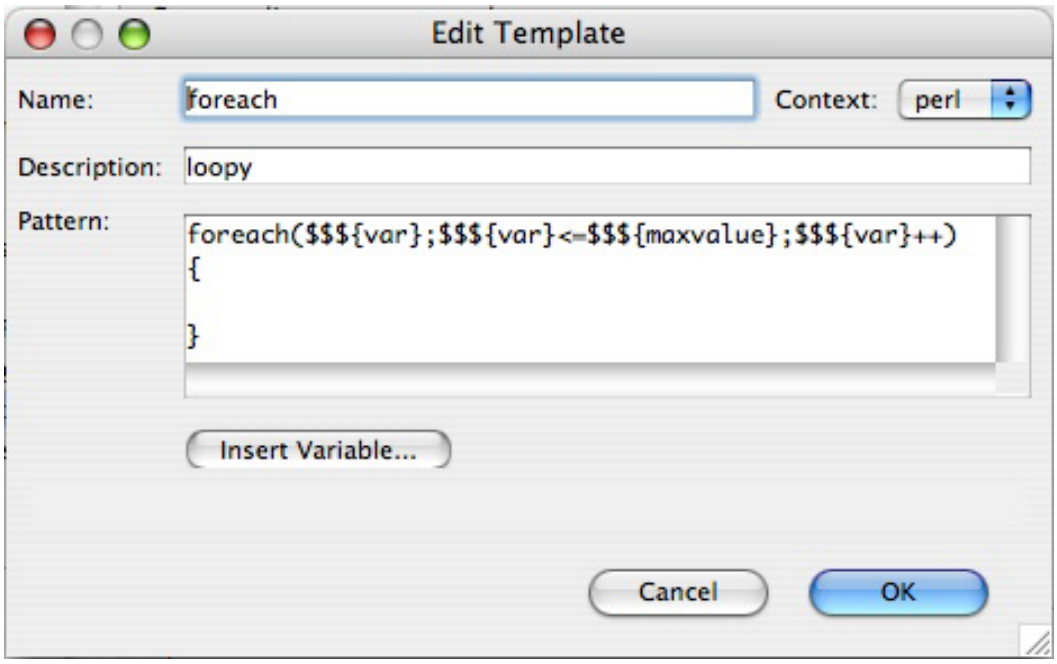
To create a new template:

1. Open the Preferences window and choose Templates from the Perl EPIC section. You should get a preference panel like the one shown here in Figure 16.
   Figure 16. Template preferences



2. Click the New button and you'll be prompted with the window in Figure 17.

Figure 17. Creating a new template



3. Enter a new name for the template. Try to keep it to a single word, as it will be used when you want to insert the code.
4. Enter a description for the code.
5. Enter the code for the template. The code you enter here will be inserted exactly as you enter it, as if you had cut and pasted the content.
6. Click OK to close the template definition.
7. Click OK to close the Preferences window.

To actually use the template, type the name of the template into your code and press Control-Space. The code you entered for your template will be inserted at the current location, replacing the template name you had specified.

While templates like this are useful, you still have to go back and set certain elements. For example, in a `for` loop, you still have to edit and enter the name of the loop variable, the test value, and the increment. You can simplify this process even further by using dynamic templates.

Creating dynamic templates

Dynamic templates are created just like standard templates. The difference is the code entered into the definition. EPIC supports the notion of *variables* in a template. These have nothing to do with variables in your code. Instead, they provide an easy way for you to customize the inserted code with simple elements.

For example, in the `for` loop example just given, you might enter code like this into the template:

```
for($i;$i<10;$i++)
{
}
```

Even though `$i` may not be the variable you want to use, or 10 the value you want to compare against. Using a dynamic template, define a variable element that can easily be replaced when the template is inserted. To do this, specify the name of a template variable using the form `${varname}`. If you use the same name multiple times, you only have to type the variable once, and you can create multiple template variables in a single template. Use a double dollar sign ($$) to insert the dollar symbol into your template.

For example, you can change the `for` loop definition to:

```
for($${var};$${var}<=${maxvalue};$${var}++)
{
}
```

Now, when you insert the template, the cursor will automatically by highlighted across the first instance of `${var}`. Type the name of the variable, and every instance of `${var}` will be replaced with the variable name. Press Tab and the highlight will move to `${maxvalue}`.

This is a piece of functionality that probably makes more sense in the flesh, so try out the above example to get a feel for how flexible the template system is with variable substitution.

Section 8. Summary

There you have it. The EPIC plug-in provides a complete environment for writing and developing Perl code. EPIC does this in a variety of ways, mostly related to how you edit, create, and format the code for it to be used within your applications. Although EPIC doesn't help actually generate the code (although templates almost provide that functionality), it does make it easier for you to navigate your source code, and you can format and auto-complete elements to reduce the amount of typing you have to do.

Resources

Learn

- Perl.com provides information and tutorials on using the Perl language.
- For help on the basics of using Eclipse, read Get started now with Eclipse.
- For more details about Eclipse, be sure to visit Eclipse.org.
- There are some interesting technical articles at the Eclipse Corner.
- To find out more about events, resources and projects happening in the Eclipse community, check out the Eclipse Corner Developer Community Resources.
- For insight into the world of Eclipse hackers, check out Planet Eclipse.
- Visit the developerWorks Open source zone for extensive how-to information, tools, and project updates to help you develop with open source technologies and use them with IBM's products.

Get products and technologies

- The EPIC plug-in is hosted on Sourceforge. You can find downloads for the EPIC system, supporting extensions and software, tutorials, documentation, and the FAQ for the system.
- Perltidy is used by EPIC to format the code to make it more readable.
- Eclipse can be obtained at Eclipse.org.
- Get ActivePerl, an interpreter for Windows (and also now for Linux and Mac OS X).
- Get PyDev, a plug-in for developing Python applications within Eclipse, which provides similar functionality to EPIC.
- Innovate your next open source development project with IBM trial software, available for download or on DVD.

Discuss

- Ask technical questions about Eclipse on the Eclipse mailing lists.
- If you are new to Eclipse, it is worth taking a look at the Eclipse newsgroups.
- Get involved in the developerWorks community by participating in developerWorks blogs.

About the author

Martin Brown

Martin Brown has been a professional writer for more than eight years. He is the author of numerous books and articles across a range of topics. His expertise spans myriad development languages and platforms -- Perl, Python, Java, JavaScript, Basic, Pascal, Modula-2, C, C++, Rebol, Gawk, Shellscript, Windows, Solaris, Linux, BeOS, Mac OS/X and more -- as well as Web programming, systems management and integration. Martin is a regular contributor to ServerWatch.com, LinuxToday.com and IBM developerWorks, and a regular blogger at Computerworld, The Apple Blog and other sites, as well as a Subject Matter Expert (SME) for Microsoft.