Pedeto Getting Started Guide

This guide is intended as a reference for those working with
Perl Development Tools for the first time.

It is recommended that you step through the
material in a sequential fashion.

It is assumed at this point that you have downloaded
Perl Development Tools and installed the plugins in Eclipse.

Last updated: 2006-06-04

Preparing Eclipse

In this section, we will verify that Eclipse is properly set up for Perl development.

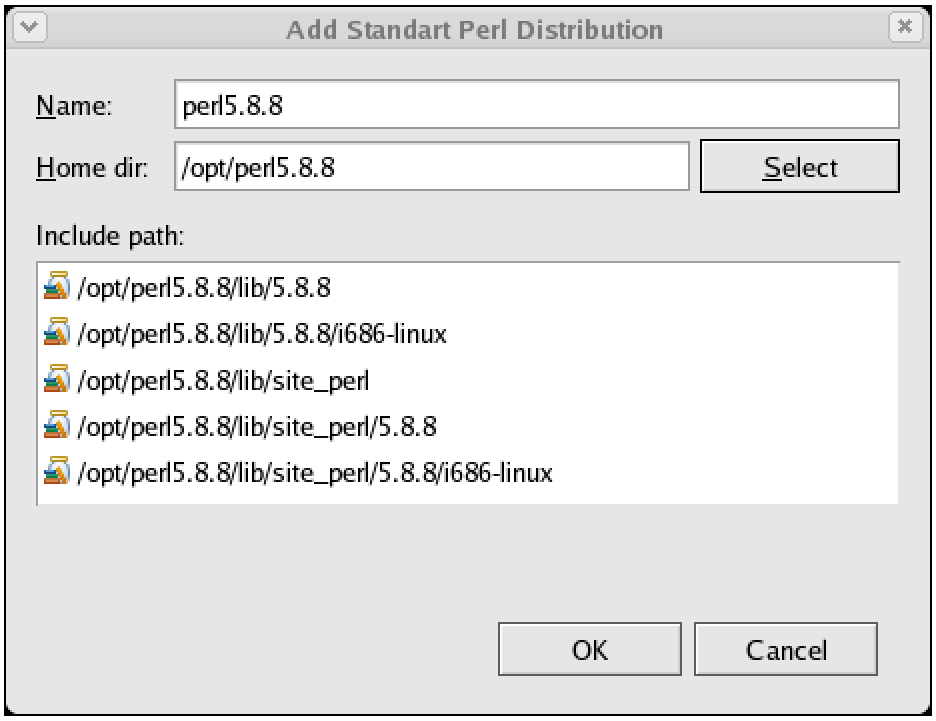
We assume the following:

- You are starting with a new Eclipse installation with default settings.
- You are familiar with the basic Eclipse workbench mechanisms, such as views and perspectives.

If you're not familiar with the basic workbench mechanisms, please see the **Getting Started** chapter of the Workbench User Guide.

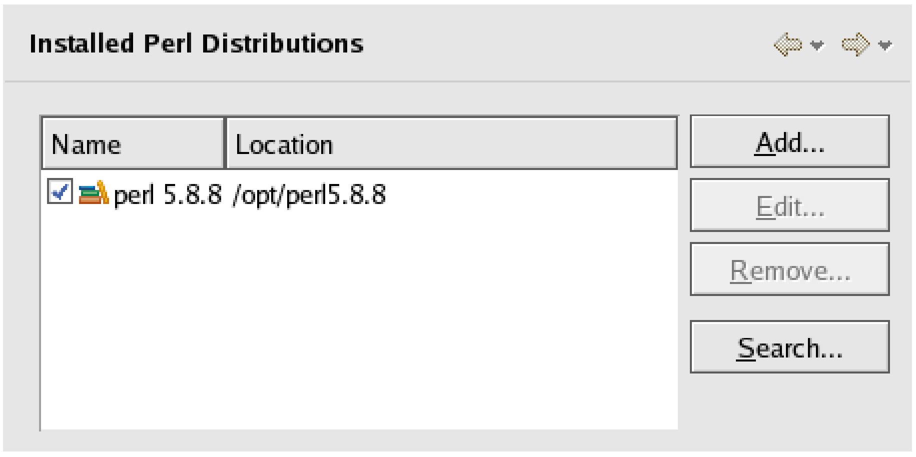
Verifying your Perl installation

1. If you still see the Eclipse Welcome page, click the arrow icon to begin using Eclipse.
2. Select the menu item **Window > Preferences** to open the workbench preferences.
3. Select **Perl > Installed Perl Distributions** in the tree pane on the left to display the **Perl Distributions Configuration** preference page. Click on the **Add...** button to select a Standard Perl Distribution. Enter the name and home directory of the Perl distribution into the dialog which opens.



Name: perl5.8.8
Home dir: /opt/perl5.8.8
Include path:
/opt/perl5.8.8/lib/5.8.8
/opt/perl5.8.8/lib/5.8.8/i686-linux
/opt/perl5.8.8/lib/site_perl
/opt/perl5.8.8/lib/site_perl/5.8.8
/opt/perl5.8.8/lib/site_perl/5.8.8/i686-linux

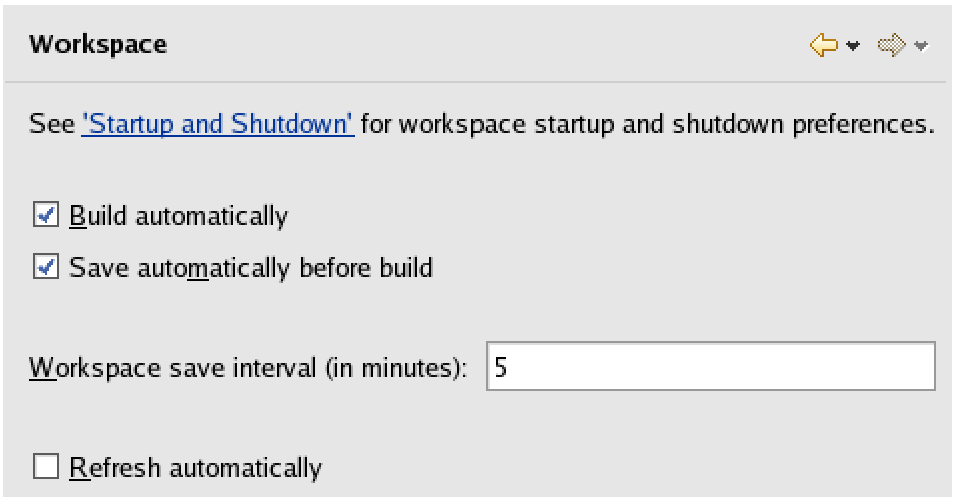
Close the dialog and check the check box in order to define the just configured Perl Distribution as your **Default Perl Distribution**. The Default Perl Distribution is used for all Perl projects unless defined otherwise at the project level. Repeat the above steps to add additional Perl Distributions.



Name	Location
✓ perl 5.8.8	/opt/perl5.8.8

If you're not sure about the location of a Perl distribution click on the **Search ...** button. Specify the root directory for the search and click on **OK**. All Perl Distributions which are found are added automatically to your list of Perl Distributions.

4. Select **General > Workspace** in the tree pane to display the **Workspace** preference page. Confirm the **Build automatically** option is checked.



Workspace

See 'Startup and Shutdown' for workspace startup and shutdown preferences.

- [✓] Build automatically
- [✓] Save automatically before build

Workspace save interval (in minutes): 5

- [ ] Refresh automatically

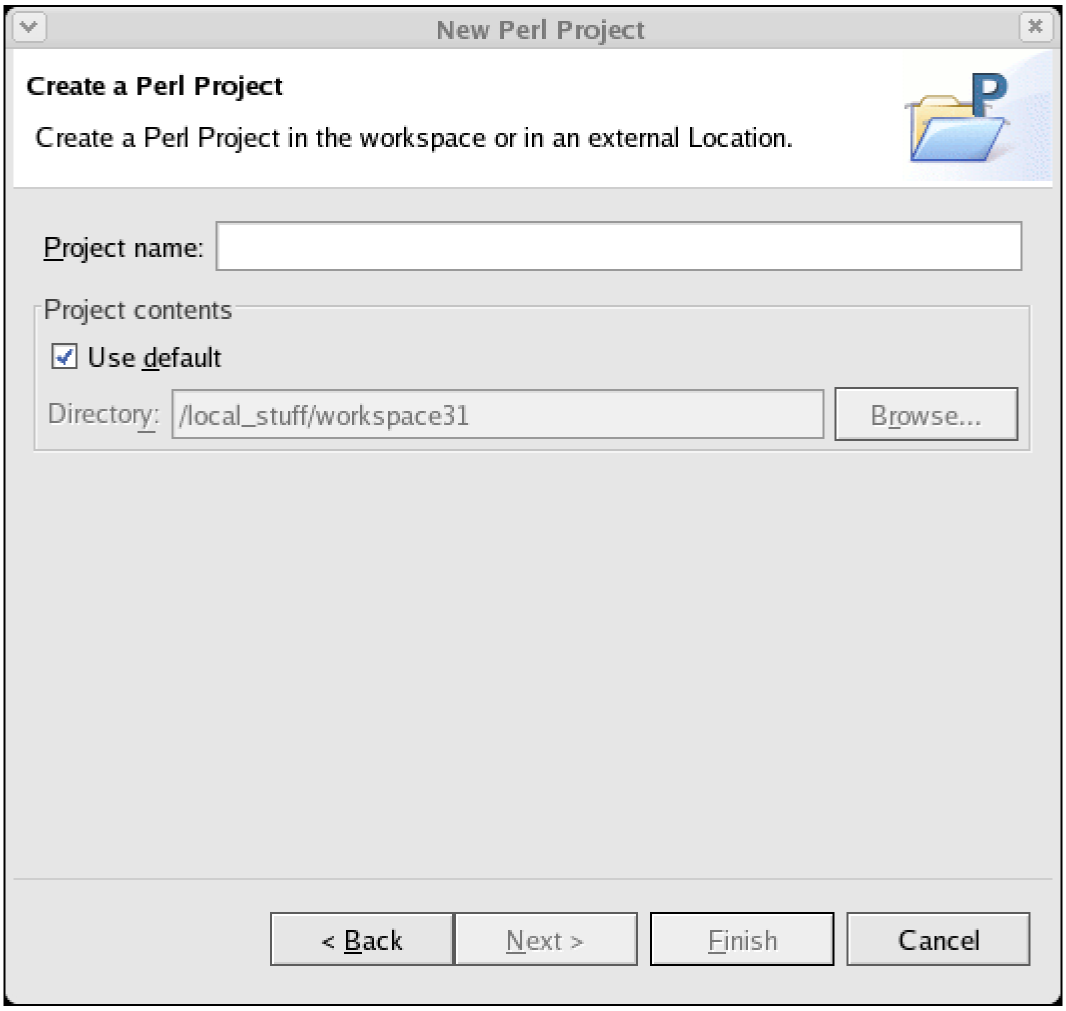
5. Click on **OK** to save the preferences.

Creating your first Perl project

In this section, we will create a new Perl project and add the famous "Hello World! program.

Creating the project

1. Open the Perl perspective.
2. Inside Eclipse select the menu item **File > New > Project...** to open the **New Project** wizard
3. Select **Perl > Perl Project**. Click **Next >** to start the **New Perl Project** wizard:



New Perl Project

Create a Perl Project

Create a Perl Project in the workspace or in an external Location.

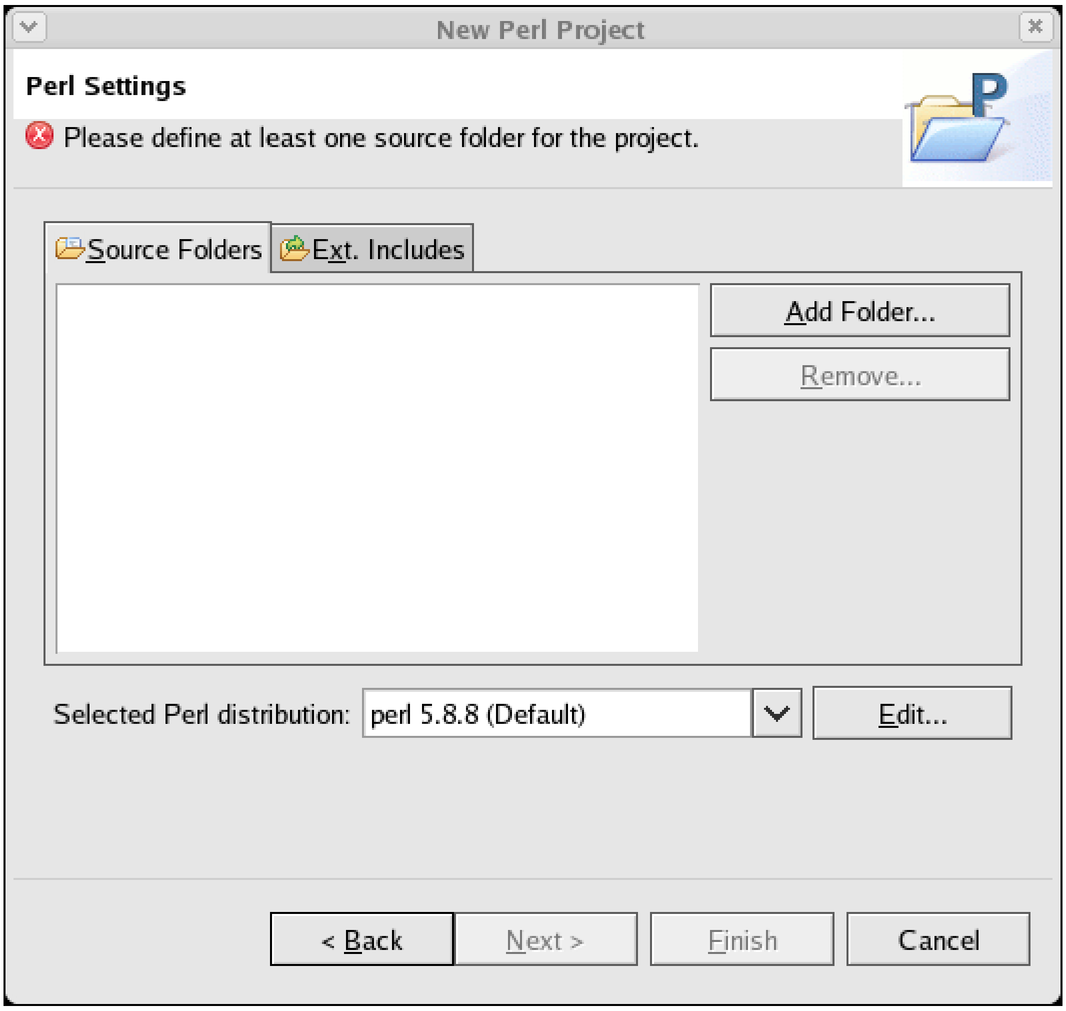
Project name:

Project contents

☑ Use default

Directory: /local_stuff/workspace31

Enter "SampleProject" into the **Project name** field and click **Next >**.

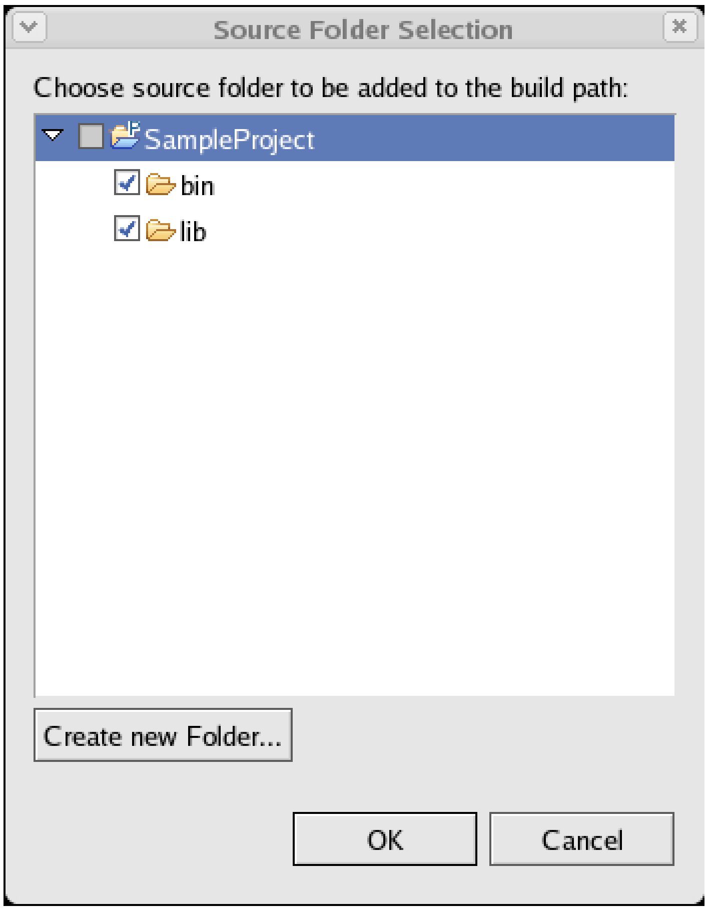

New Perl Project

Perl Settings

Please define at least one source folder for the project.

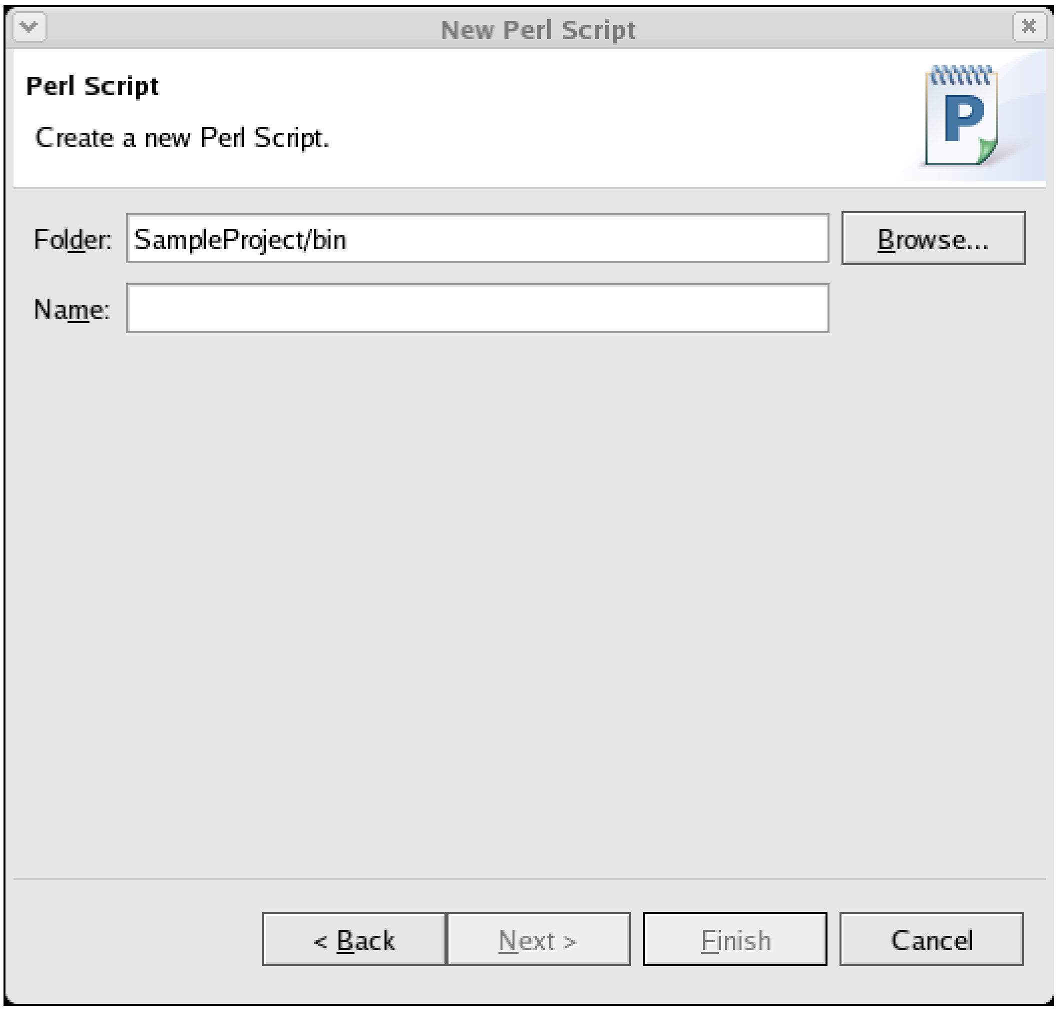
Selected Perl distribution: perl 5.8.8 (Default)

Click on **Add Folder...** and create a folder named "bin". We will place all "scripts" into the bin folder Create an additional folder "lib". All module files will be placed into this directory. Ensure both directories are selected in the dialog as shown below.



Close the dialog and click on **Finish**. Your Perl Project will be created.

4. Select the bin folder of SampleProject in the Perl Navigator.
5. Select the menu item **File > New > Other...** to open the **New Module** wizard
6. Select **Perl > Script** then click **Next** to start the **New Script** wizard:

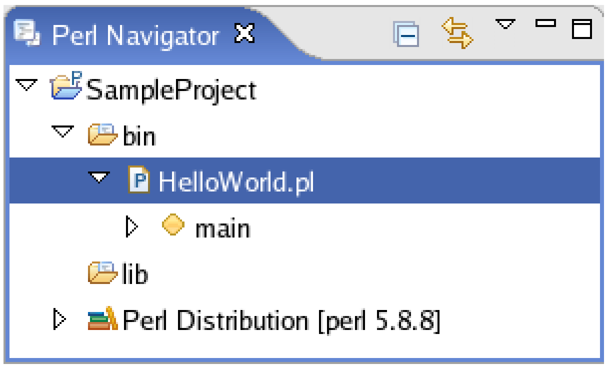


7. Type "HelloWorld" in the **Name** field and slick **Finish**. The file will now be created and the Perl editor is opened with the new file.
8. Type
   ```
   use strict;
   use warnings;
   print "Hello World!";
   ```
   in the Perl editor.
9. Congratulations. You've created the famous Hello World script in Perl.

Browsing Perl elements in the Perl Navigator

In this section, you will browse Perl elements within the SampleProject project.

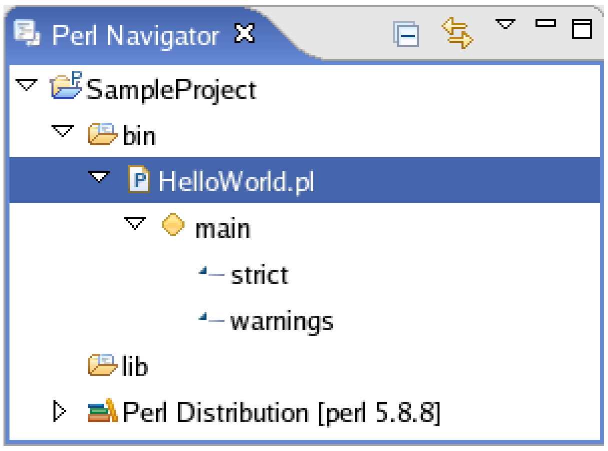
1. Make sure you're in the Perl perspective. Open the Sample Project, the bin folder and click on the expand icon left to *HelloWorld.pl*.

   Perl Navigator

   ```
   SampleProject
     bin
       HelloWorld.pl
         main
     lib
     Perl Distribution [perl 5.8.8]
   ```

   Note the yellow icon and the text main beneath *HelloWorld.pl*. The Navigator shows subelements of a source file in the tree; e.g. the declared packages in a file. Since we didn't define a package in our *HelloWorld.pl* script Perl places everything automatically into the main package.
2. If you open the main folder you will see **strict** and **warnings**. These are the two "modules" which were included.

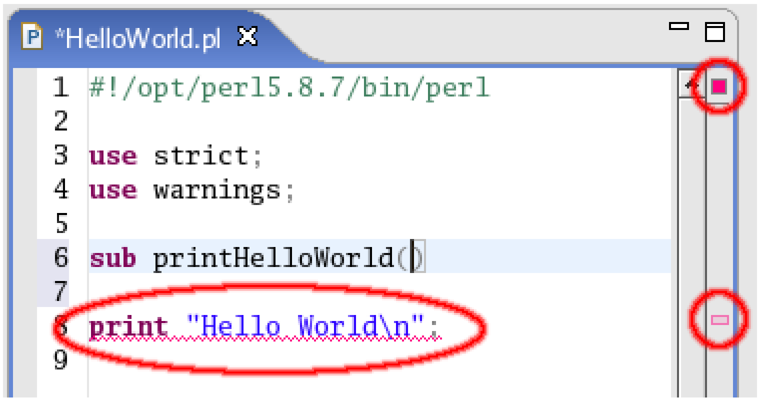
   Perl Navigator

   ```
   SampleProject
     bin
       HelloWorld.pl
         main
           strict
           warnings
     lib
     Perl Distribution [perl 5.8.8]
   ```

Adding a Method to the Script

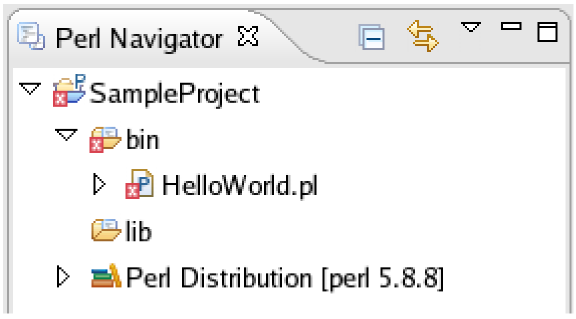
In this chapter we'll add a method to the Hello World script and take a first look at the background syntax check.

1. Open the *HelloWorld.pl* script by double clicking on in it in the Perl Navigator. Now start adding a method by typing the following before the print command `sub printHelloWorld()`

   ```
   *HelloWorld.pl
   #!/opt/perl5.8.7/bin/perl

   use strict;
   use warnings;

   sub printHelloWorld()

   print "Hello World\n";
   ```

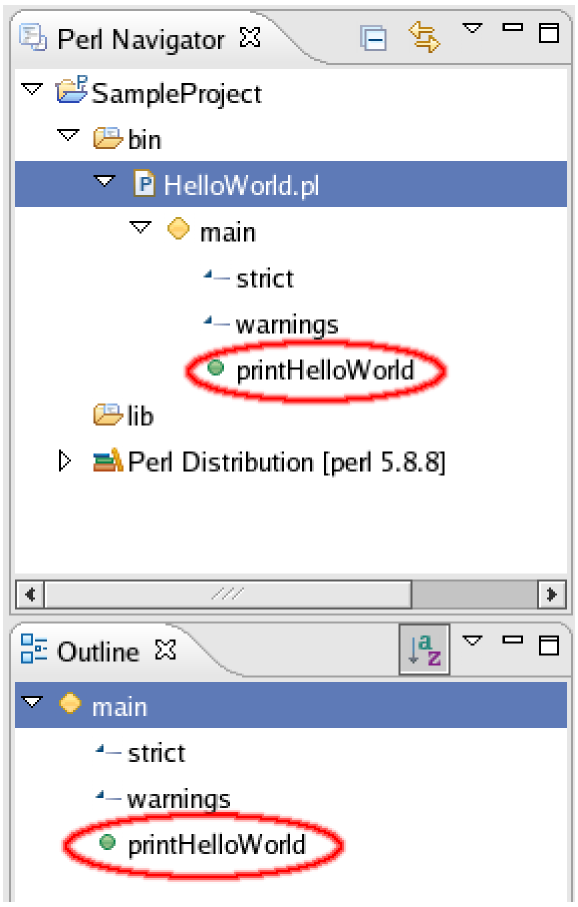
   Since we did't finish the method declaration the code cannot be compiled successfully. This is indicated by error annotations (red boxes) which appear in the overview ruler, positioned on the right hand side of the editor. If you hover over the second red box, a tool tip appears: *Illegal declaration of subroutine main::printHelloWorld*. Note that error annotations in the editor's rulers are updated as you type.
2. Click the **Save** button. The compilation unit is compiled automatically and errors appear in the Perl Navigator view, in the Problems view and on the vertical ruler (left hand side of the editor). In the Perl Navigator view, the errors are propagated up to the project of the compilation unit containing the error.

3. Complete the new method by typing the following:

   ```
   {
    print "Hello World";
   ```

   The closing curly bracket was inserted automatically.
4. Note that the new method appears at the bottom of the Outline view. It also appears in the Perl Navigator.
5. Save the file. Notice that the error indicators disappear since the script can now be compiled successflly.

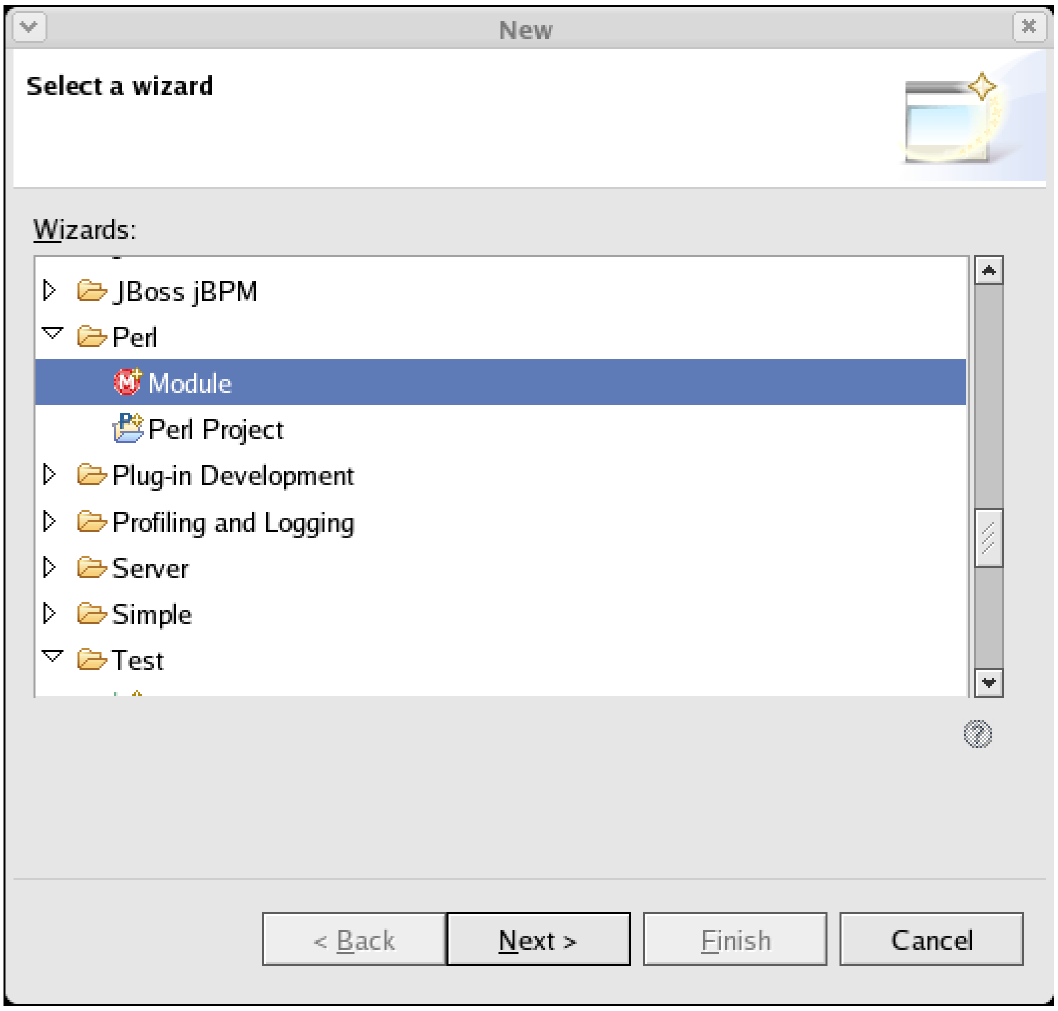


6. Finally remove the `print "Hello World!";` line outside the method.

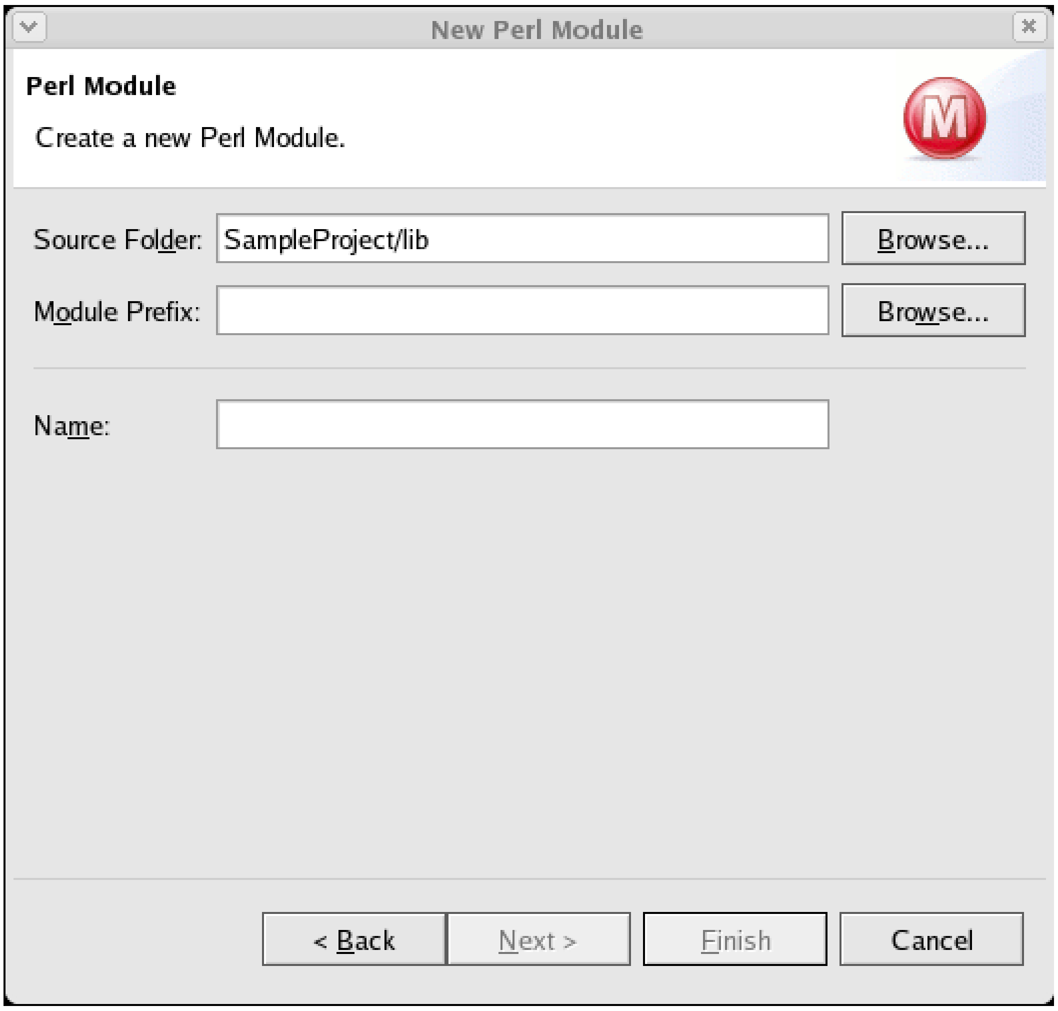
Adding new modules

In this chapter we'll add a new module to the project.

1. Select the *lib* folder in the Perl Navigator. Right click on *lib* and choose **New > Other ...**. Select **Perl > Module**



Now click on **Next >** and the new **Perl Module Wizard** opens.



Source Folder: SamplePoject/lib

Module Prefix:

Name:

Don't enter anyhting into **Module Prefix** and enter *HelloWorld* into the **Name** entry field. Click on **Finish**. This will close the dialog and create the *HelloWorld.pm* file in your lib folder. The newly created file will be opened.

2. Now let's take a look at the file. The file is empty but of the `package HelloWorld;` line at the top of the file.
3. Now we're adding the recommended additional compile checks a constructor (a simple function we call new) and a method which will print "Hello World from a Module". Do so by pasting the following code into the editor:

```
use strict;
use warnings;

=head2 new

The constructor of the class.

=cut

sub new ($) {
  my $that  = shift;
  my $class = ref($that) || $that;
  my $self  = {};
  $self->{'FooBar'} = 1;
  bless($self, $class);
  return $self;
}


=head2 printHelloWorld

A method which will print the Hello World from a Module message.

=cut

sub printHelloWorld ($) {
  my($self) = @_;
  print "Hello World from a Module.\n";
}

1;
```

4. Take you're time and look at th Perl Navigator and the outline. The new module and methods are added to both of them.
5. This is all which is needed to add a new Perl module to your project.

Using content assist

In this section you will use *content assist* to write a method call. Open the *HelloWorld.pl* file in the Perl editor if it is not already open.

1. First of all, we include the module which we've created in the previous chapter. This is done by adding

   ```
   use HelloWorld;
   ```

   below the `use warnings;` statement but before the method declaration.
   If you're familar with Perl you might now start thinking about why the `use HelloWorld;` statement doesn't trigger a compilation error. The reason for this is, the *lib* folder is set as being a source folder in the Project Preferences. So it is added automatically to the include path when compiling *HelloWorld.pl*. The script would however now fail at the command line if we would invoke it directly. So we add the following lines of code to the *HelloWorld.pl* script (before the `use HelloWorld;` statement).

   ```
   use File::Spec;
   use FindBin;
   BEGIN {
     my $libPath = File::Spec->catdir($FindBin::Bin, File::Spec->updir(), 'lib');
     push(@INC, $libPath);
   }
   ```

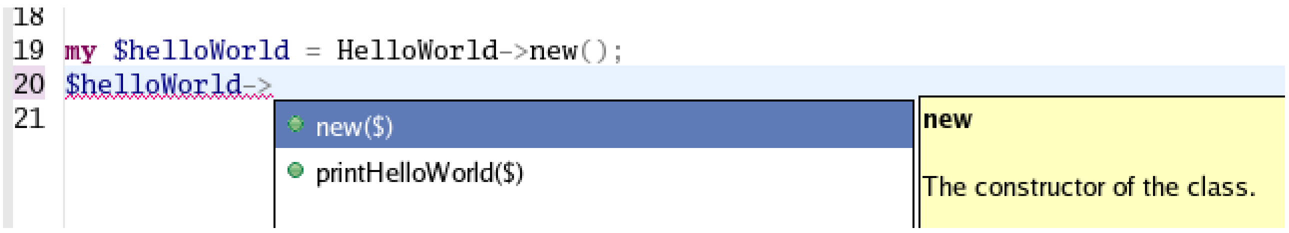
   The script now compiles without a hitch; both in Pedeto and at the command line.
2. After we're now done with the prepration work, let's start using content assist. First we create a new instance of the HelloWorld class by adding

   ```
   my $helloWorld = HelloWorld->new();
   ```

   at the end of the file. This is important so the IDE knows which methods may be called for $helloWorld. Now type

   ```
   $helloWorld->
   ```

   at the end of the file. The content assist dialog will now pop up and provides you a list of proposals.

   ```
   my $helloWorld = HelloWorld->new();
   $helloWorld->
   ```

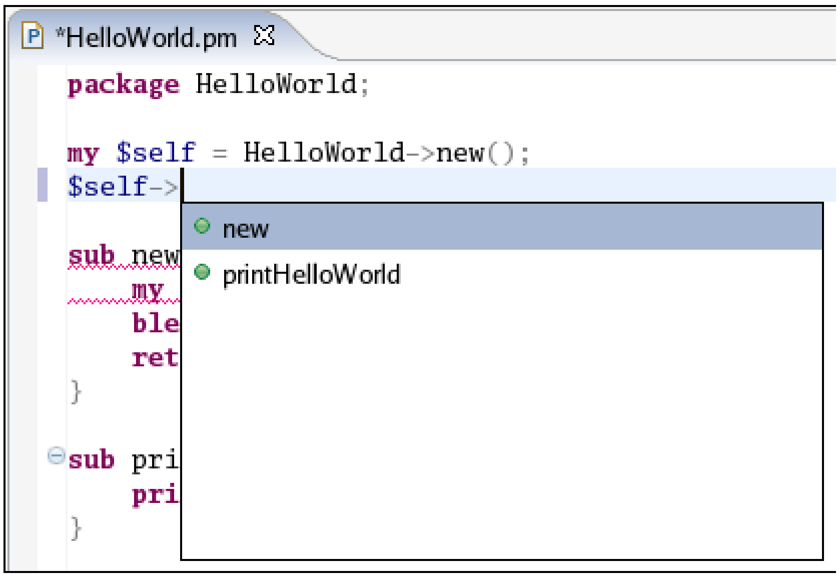
   Also note the pod documentation, which was inserted above the currently selected method, is also shown.
3. Type the letter 'p' just behind `$helloWorld->`. The list of proposals is narrowed and now only shows the "printHelloWorld()" method. Press enter to insert the method into the code. You can also open content assist manually by pressing `Ctrl+Space` after `$helloWorld->`.
4. Complete the line by typing `;`.
5. Save the file.

Important: Since Perl variables are not type safe, the IDE cannot know which proposals it shall make for a variable. So you have to give the IDE some hints about the type of a variable. If you create a new instance via the **new** method like
`my $testVariable = Red::Green::Blue->new();`
or
`my $testVariable = new Red::Green::Blue();`
the IDE is able to determine automatically the class of $testVariable. And that's also the reason why content assist did work in the above example. If the variable already holds an instance (e.g. a variable which was passed to a method call) you should add a comment before the first use of the variable like this:
`#@var $testvariable Red::Green::Blue`
Another option for placing this comment is in the pod documentation above a method or in the pod documentation at the top of the module (so you have to declare e.g. the type of $self only once per file). Let's revisit the *HelloWorld.pm* module. Open the module and add

```
#@var $self HelloWorld
```

into the pod documentation of the *printHelloWorld* method. Now go into the *printHelloWorld* method and type

`$self->`. Content assist will now also pop up as shown below.



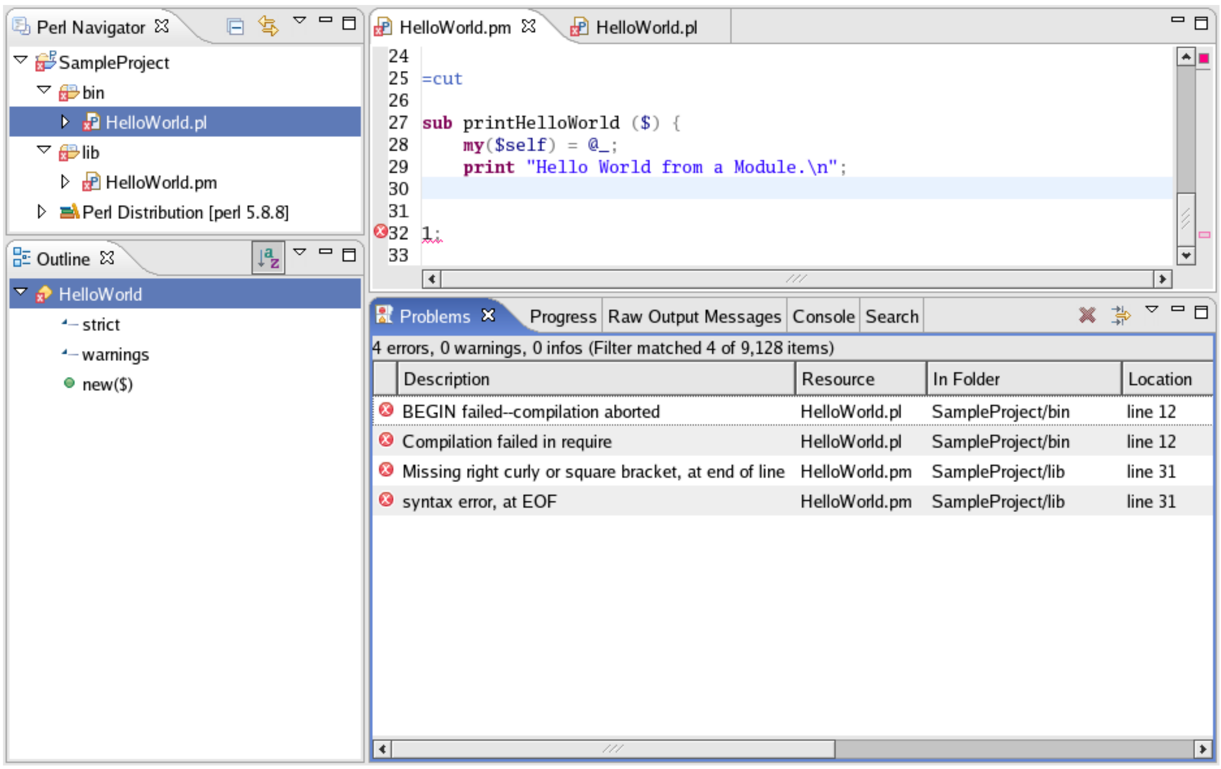
```
package HelloWorld;

my $self = HelloWorld->new();
$self->

sub new
    my
    ble
    ret
}

sub pri
    pri
}
```

Identifying problems in your code

In this section, we will review the different indicators for identifying problems in your code.

Build problems are displayed in the Problems view and annotated in the vertical ruler of your source code.

1. Open *HelloWorld.pm* in the editor from the Perl Navigator view.
2. Add a syntax error by deleting the closing curly bracket of the *printHelloWorld* method.
3. Click the **Save** button. The project is rebuilt and the problem is indicated in several ways:
   - In the Problems view, the problems are listed,
   - In the Perl Navigator view and the Outline view, problem ticks appear on the affected Perl elements and their parent elements,
   - In the editor's vertical ruler, a problem marker is displayed near the affected line,
   - Squiggly lines appear under the word which might have caused the error, and
   - The editor tab is annotated with a problem marker.



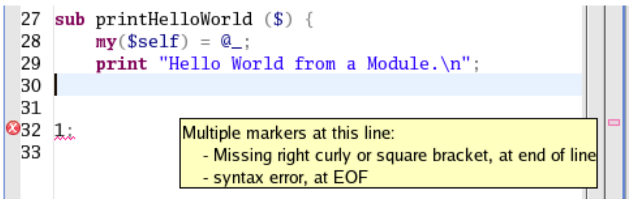
```
24
25 =cut
26
27 sub printHelloWorld ($) {
28     my($self) = @_;
29     print "Hello World from a Module.\n";
30
31
32 1;
33
```

4 errors, 0 warnings, 0 infos (Filter matched 4 of 9,128 items)

Description	Resource	In Folder	Location
BEGIN failed--compilation aborted	HelloWorld.pl	SampleProject/bin	line 12
Compilation failed in require	HelloWorld.pl	SampleProject/bin	line 12
Missing right curly or square bracket, at end of line	HelloWorld.pm	SampleProject/lib	line 31
syntax error, at EOF	HelloWorld.pm	SampleProject/lib	line 31

An important thing to note are the error messages in *HelloWorld.pl*. Since *HelloWorld.pm* is included by *HelloWorld.pl* it also cannot be compiled successfully any more. Thus error markers appear for *HelloWorld.pl*, too.

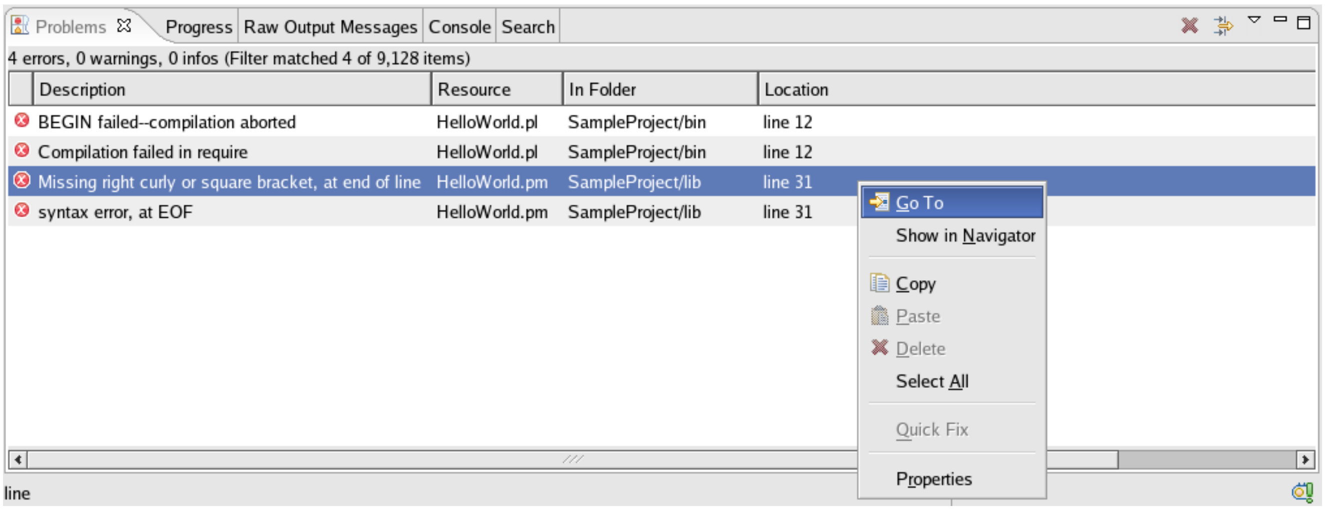
4. You can hover over the problem marker in the vertical ruler to view a description of the problem.



```
27 sub printHelloWorld ($) {
28     my($self) = @_;
29     print "Hello World from a Module.\n";
30
31
32 1;
33
```

Multiple markers at this line:
- Missing right curly or square bracket, at end of line
- syntax error, at EOF

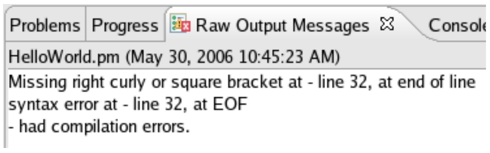
5. Click the **Close** ("X") button on the editor's tab to close the editor.
6. In the Problems view, select a problem in the list. Open its context menu and select Go To. The file is opened in the editor at the location of the problem.



4 errors, 0 warnings, 0 infos (Filter matched 4 of 9,128 items)

Description	Resource	In Folder	Location
BEGIN failed--compilation aborted	HelloWorld.pl	SampleProject/bin	line 12
Compilation failed in require	HelloWorld.pl	SampleProject/bin	line 12
Missing right curly or square bracket, at end of line	HelloWorld.pm	SampleProject/lib	line 31
syntax error, at EOF	HelloWorld.pm	SampleProject/lib	line 31

7. If you want to get access to the **Raw Output Message** which is generated by the Perl interpreter open the corresponding view via **Window > Show View > Other ...**.Select **Perl > Raw Output Messages** in the dialog which opens. Now activate the view we've just opened. You should see output similar to the output shown in the below image.



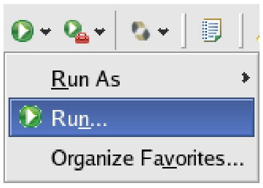
```
HelloWorld.pm (May 30, 2006 10:45:23 AM)
Missing right curly or square bracket at - line 32, at end of line
syntax error at - line 32, at EOF
- had compilation errors.
```

8. Correct the problem in the editor by adding the missing closing curly bracket. Click the **Save** button. The project is rebuilt and the problem indicators disappear.

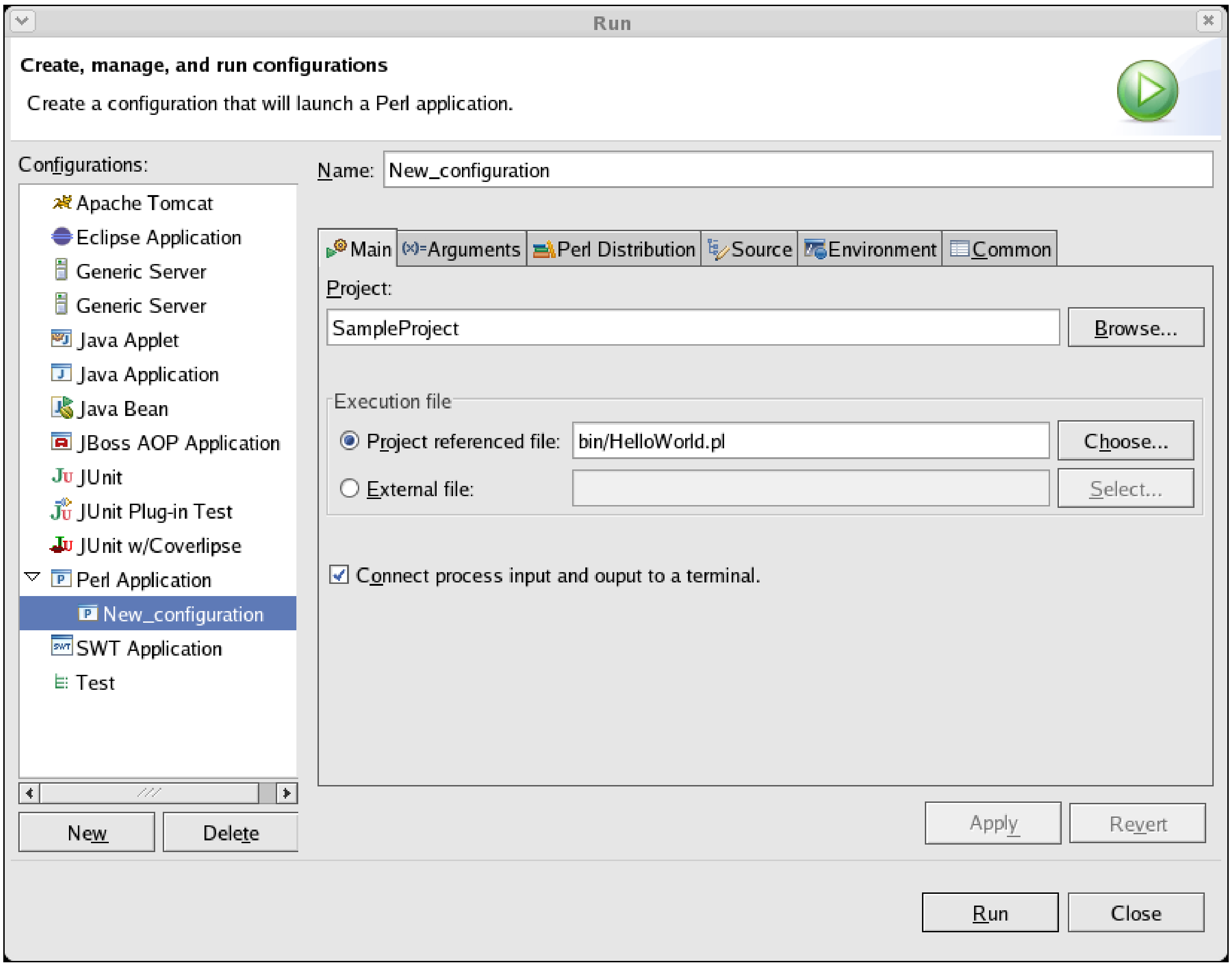
Running your programs

In this section, you will learn more about running Perl programs in the workbench.

1. In the Perl Navigator view, find *HelloWorld.pl* and select it.
2. To create a Launch Configuration, use the drop-down **Run** menu in the toolbar and select **Run...** to open the Launch Configurations dialog.



3. Select **Perl Application** and click on the **New** button to create a new Perl Launch Configuration.



Run

Create, manage, and run configurations

Create a configuration that will launch a Perl application.

Configurations:
- Apache Tomcat
- Eclipse Application
- Generic Server
- Generic Server
- Java Applet
- Java Application
- Java Bean
- JBoss AOP Application
- JUnit
- JUnit Plug-in Test
- JUnit w/Coverlipse
- Perl Application
  - New_configuration
- SWT Application
- Test

Name: New_configuration

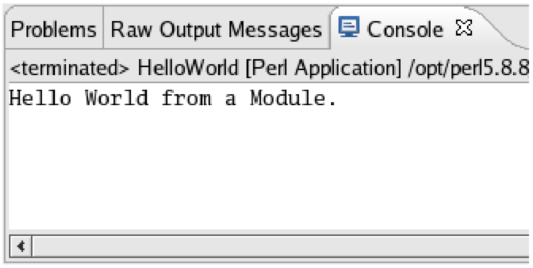
Main | Arguments | Perl Distribution | Source | Environment | Common

Project: SampleProject

Execution file
- (●) Project referenced file: bin/HelloWorld.pl
- ( ) External file:

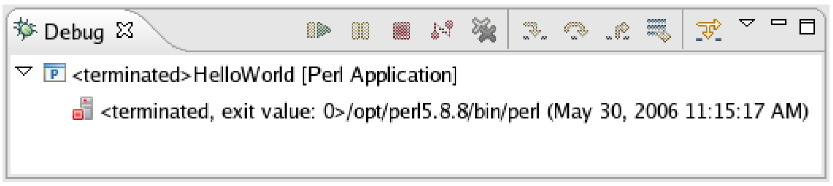
[✓] Connect process input and ouput to a terminal.

4. Enter "HelloWorld" as Name into the new Perl Launch Configuration and press **Apply**.
5. Click **Run** to run the application. The console will capture the output generated by our script.



```
<terminated> HelloWorld [Perl Application] /opt/perl5.8.8
Hello World from a Module.
```

6. Switch to the Debug perspective. In the Debug view, notice that a process for the last program launch was registered when the program was run.



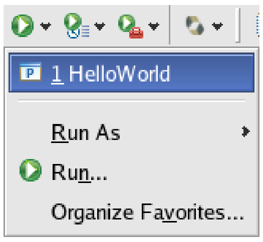
<terminated>HelloWorld [Perl Application]

<terminated, exit value: 0>/opt/perl5.8.8/bin/perl (May 30, 2006 11:15:17 AM)

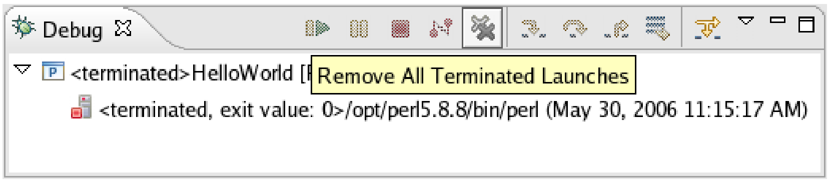
By default, the Debug view automatically removes any terminated launches when a new launch is created. This preference can be configured on the **Launching** preference page located under the **Run/Debug** preference page.

Note: You can relaunch a terminated process by selecting ***Relaunch*** *from its context menu.*

7. Select the drop-down menu from the **Run** button in the workbench toolbar. This list contains the previously launched programs. These programs can be relaunched by selecting them in the history list.



8. From the context menu in the Debug view (or the equivalent toolbar button), select **Remove All Terminated** to clear the view of terminated launch processes.



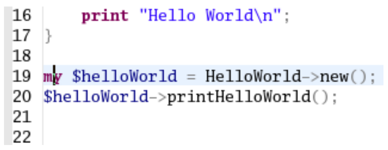
<terminated>HelloWorld [P

<terminated, exit value: 0>/opt/perl5.8.8/bin/perl (May 30, 2006 11:15:17 AM)

Debugging your programs

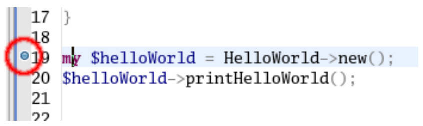
In this section, we'll debug a Perl program.

1. Make sure you're in the Perl Perspective. Locate *HelloWorld.pl* in the Perl Navigator and double-click on it so it is opened in an editor.
2. Locate the line which contains `my $helloWorld = HelloWorld->new();`.

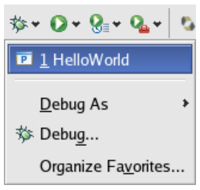
   ```
       print "Hello World\n";
   }

   my $helloWorld = HelloWorld->new();
   $helloWorld->printHelloWorld();


   ```

   Move your mouse pointer to the left ruler and double-click on the left ruler at the position of the above mentioned line. This will create a break point.

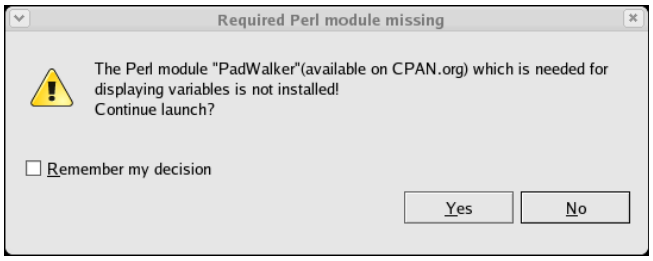
   

   ```
   }

   my $helloWorld = HelloWorld->new();
   $helloWorld->printHelloWorld();

   ```

   The breakpoint icon indicates the status of the breakpoint. The plain blue breakpoint icon indicates that the breakpoint has been set, but not yet installed.
   Note: Once the debugger was started and the breakpoint was installed, a checkmark overlay will be displayed on the breakpoint icon.
3. In the toolbar, select the *Hello World* Launch Configuration (in debug mode) we've made in Running your programs.

4. The script will now be started in a debug session. If the **PadWalker** module wasn't installed in the currently used Perl Distribtion, a warning will be displayed. The PadWalker module is needed to display detailed information about the variables which are being used. The PadWalker module can be downloaded from cpan.org.

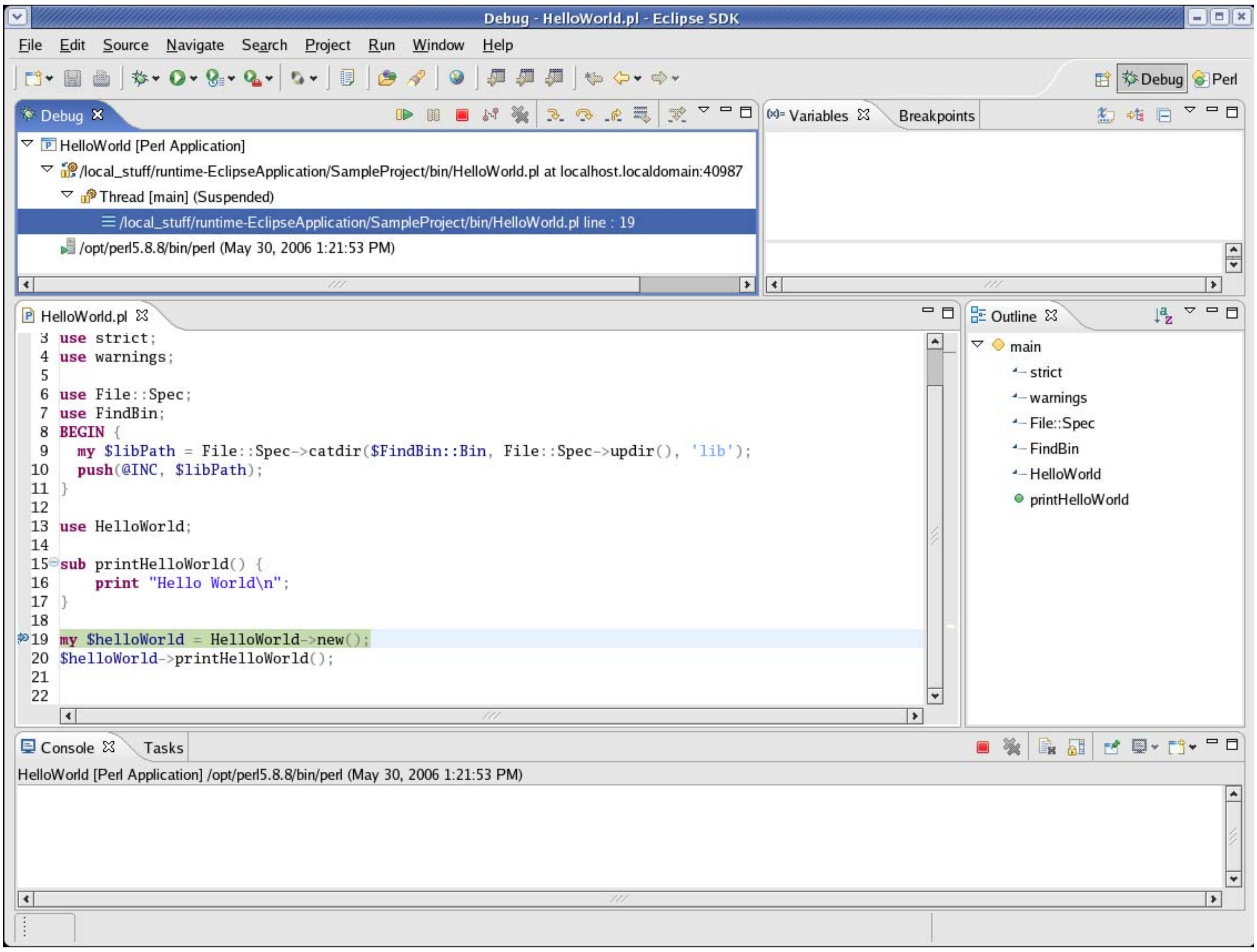
   Required Perl module missing

   The Perl module "PadWalker"(available on CPAN.org) which is needed for displaying variables is not installed!
   Continue launch?

   ☐ Remember my decision

   Yes No

   Note: We strongly recommended installing the PadWalker module.
5. The program will run until the breakpoint is reached. When the breakpoint is hit, execution is suspended, and you are asked whether to open the Debug perspective (if you're not already in the Debug perspective). Click **Yes**. Notice that the process is still active (not terminated) in the Debug view.


Debug - HelloWorld.pl - Eclipse SDK
File Edit Source Navigate Search Project Run Window Help
Debug Perl
Debug
Variables Breakpoints
HelloWorld.pl
Outline
Console Tasks

```
HelloWorld [Perl Application]
  /local_stuff/runtime-EclipseApplication/SampleProject/bin/HelloWorld.pl at localhost.localdomain:40987
    Thread [main] (Suspended)
      /local_stuff/runtime-EclipseApplication/SampleProject/bin/HelloWorld.pl line : 19
  /opt/perl5.8.8/bin/perl (May 30, 2006 1:21:53 PM)
```

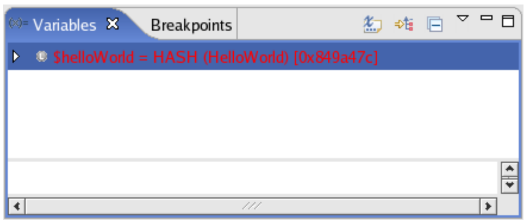
```perl
use strict;
use warnings;

use File::Spec;
use FindBin;
BEGIN {
  my $libPath = File::Spec->catdir($FindBin::Bin, File::Spec->updir(), 'lib');
  push(@INC, $libPath);
}

use HelloWorld;

sub printHelloWorld() {
    print "Hello World\n";
}

my $helloWorld = HelloWorld->new();
$helloWorld->printHelloWorld();
```

```
main
  strict
  warnings
  File::Spec
  FindBin
  HelloWorld
  printHelloWorld
```

```
HelloWorld [Perl Application] /opt/perl5.8.8/bin/perl (May 30, 2006 1:21:53 PM)
```

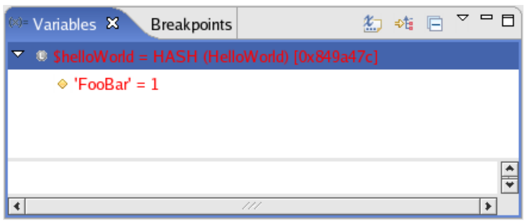
Note: The breakpoint now has a checkmark overlay since the breakpoint was successfully established in the debug session.

6. Now hit `F6`. This will execute a single step in debug mode. Note *$helloWorld* is being added automatically to the Variables view.



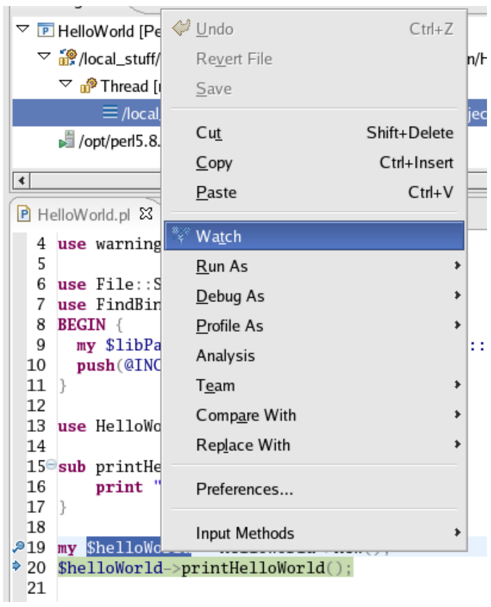
```
$helloWorld = HASH (HelloWorld) [0x849a47c]
```

7. If you've got the PadWalker module installed clicking on the arrow button will provide more information about the values which are stored in the hash.



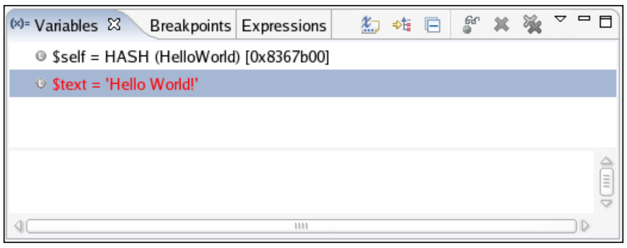
```
$helloWorld = HASH (HelloWorld) [0x849a47c]
    'FooBar' = 1
```

8. In the editor in the Debug perspective, select `$helloWorld` from the line where the breakpoint is set, and from its context menu, select **Watch**.



```
use warning
use File::S
use FindBin
BEGIN {
   my $libPa
   push(@INC
}

use HelloWo

sub printHe
      print "
}

my $helloWo
$helloWorld->printHelloWorld();
```

9. The expression is evaluated in the context of the current stack frame, and shown in the Expression view in the upper-right edge of the Debug view.
10. The Variables view (available on a tab along with the Expressions view) displays the values of the variables in the selected stack frame.
11. The variables (e.g., $helloWorld) in the Variables view will change when you step through HelloWorld in the Debug view. To step through the code, click the **Step Over** ( ) button. Execution will continue at the next line in the same method (or, if you are at the end of a method, it will continue in the method from which the current method was called).



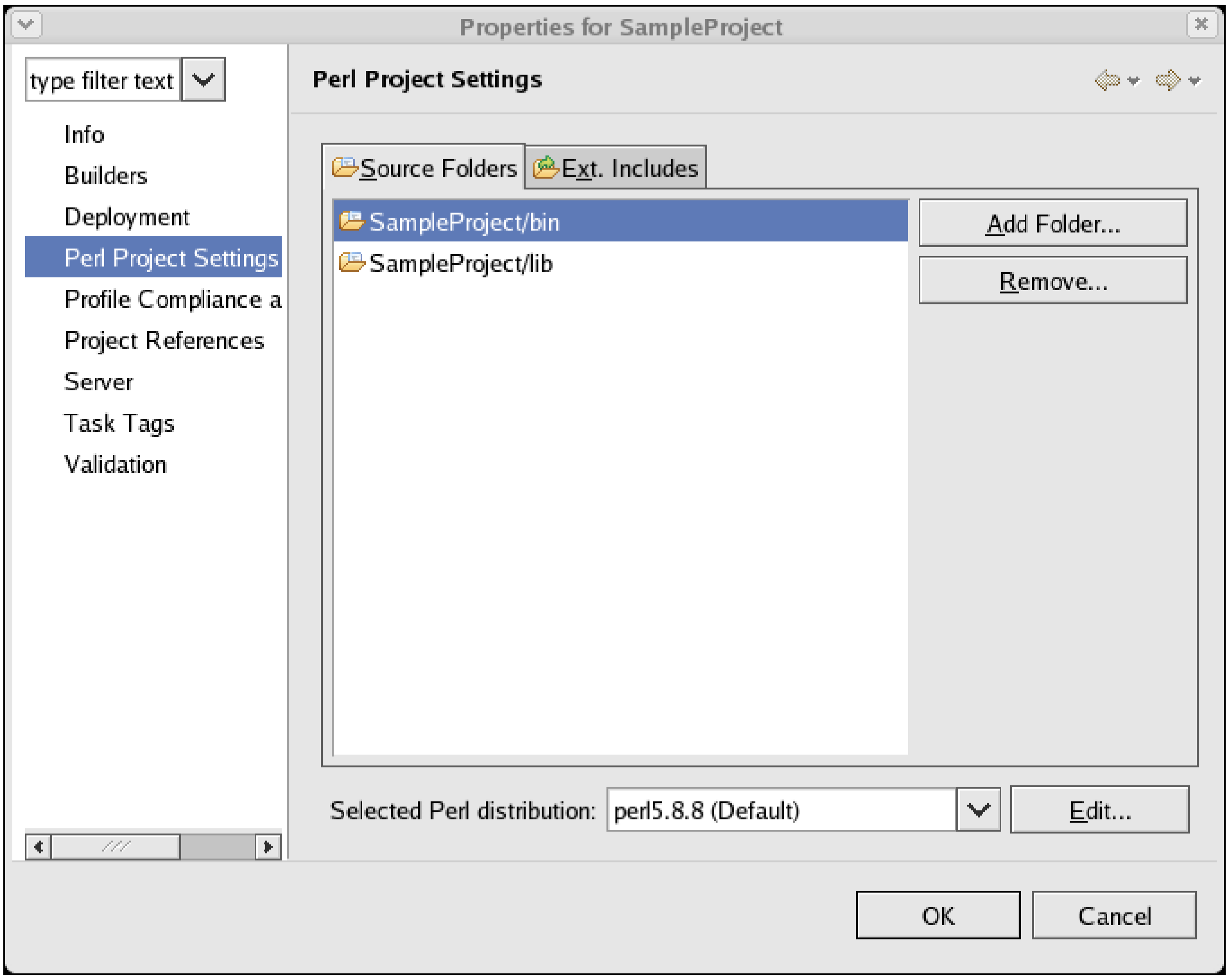
```
$self = HASH (HelloWorld) [0x8367b00]
$text = 'Hello World!'
```

12. There are other step buttons (**Step Into** , **Step Return**) to step through the code. Note the differences in stepping techniques.
13. You can end a debugging session by allowing the program to run to completion or by terminating it.
14. You can continue to step over the code with the **Step** buttons until the program completes.
15. You can click the **Resume** ( ) button to allow the program to run until the next breakpoint is encountered or until the program is completed.
16. You can select **Terminate** from the context menu of the program's process in the Debug view to terminate the program.

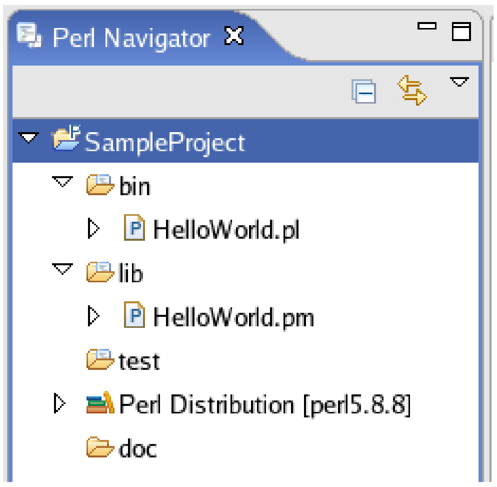
Organizing sources

In this section, we will take a closer look at our current project set up.

1. Select the SampleProject and open the context menu. Click on **Properties**. Select **Perl Project Settings**



2. As we've written in Creating your first Perl project we split the scripts files (.pl) and the perl modules (.pm) into two different folders. The script files were put into the *bin* folder and the modules into the *lib* folder. Let's not dive into a discussion on whether this is useful or not. As usual this is just one way of many on how you might want to organize your project.
3. All folders which are configured as being source folders are added to @INC before a script or module is being compiled; or before you start to execute or debug it.
4. Now let's assume we need an additional folder *test* into which we'll place our regression tests. Doing so is straightforward. Click on the **Add Folder ...** button. Click on **Create new Folder ...** and enter *test* into the **Folder name:** entry field of the dialog which opens. Click on **OK** then check the checkbox before *test* and close this dialog by clicking on **OK**, too. You have now a total of three source folders configured for your project. Close this dialog by selecting **OK** returning you to the Perl Perspective.
5. Finally we'll add a folder into which we'll place the documentation of the project. This is done by selecting the *SampleProject* in the Perl Navigator. Then open the context menu, select **New > Folder**. Enter *doc* into the **Folder name:** entry field. Close the dialog by clicking on **Finish**.



Note the *doc* folder has a different icon then the *bin, lib* and *test* folders. This way you can easily identify the folders which contain source files and which do not.

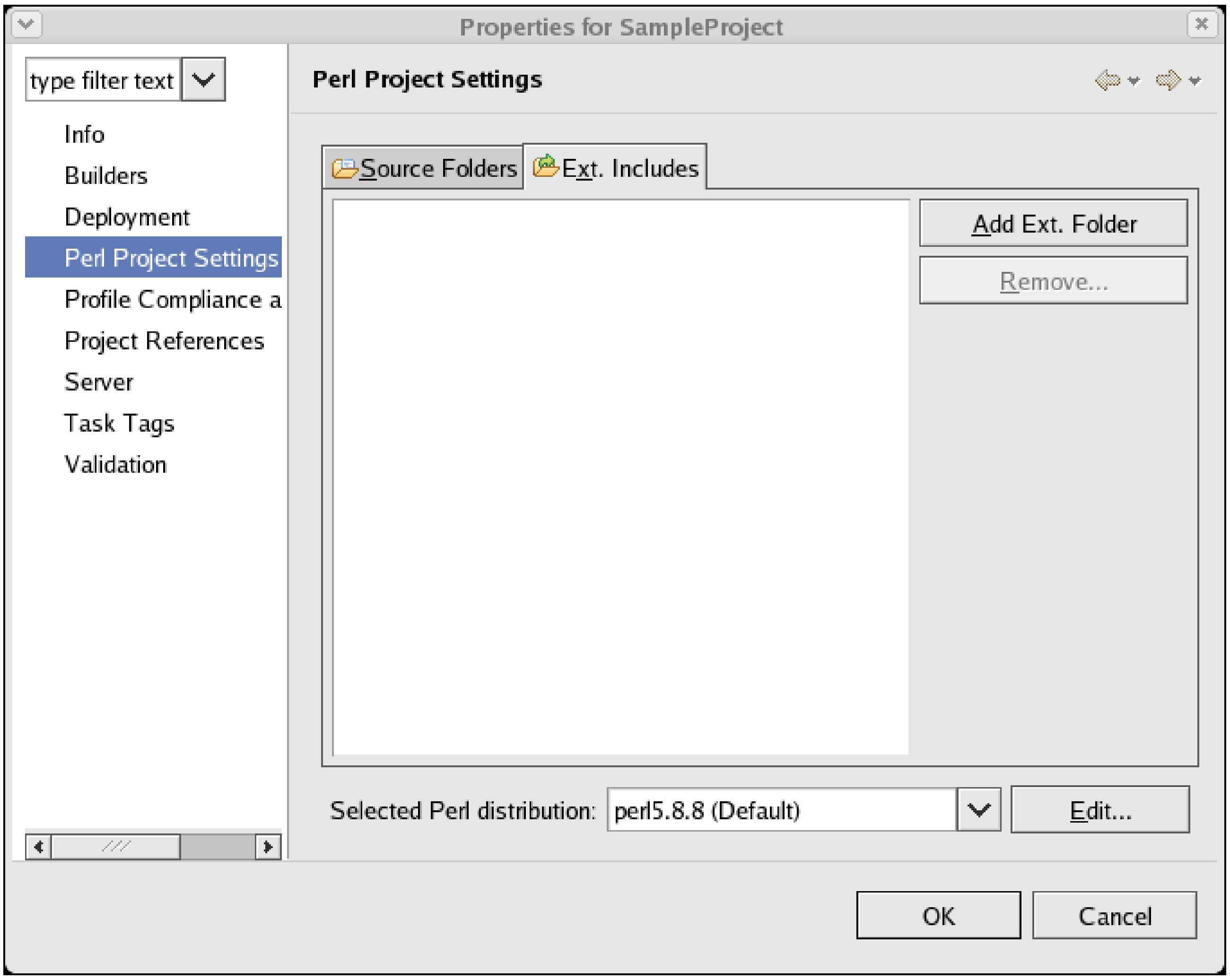
External Includes

If you've got a project which has dependencies into other projects you might want to add them to your @INC so they can be resolved properly for syntax checks, code assist and so on.

Adding an external folder

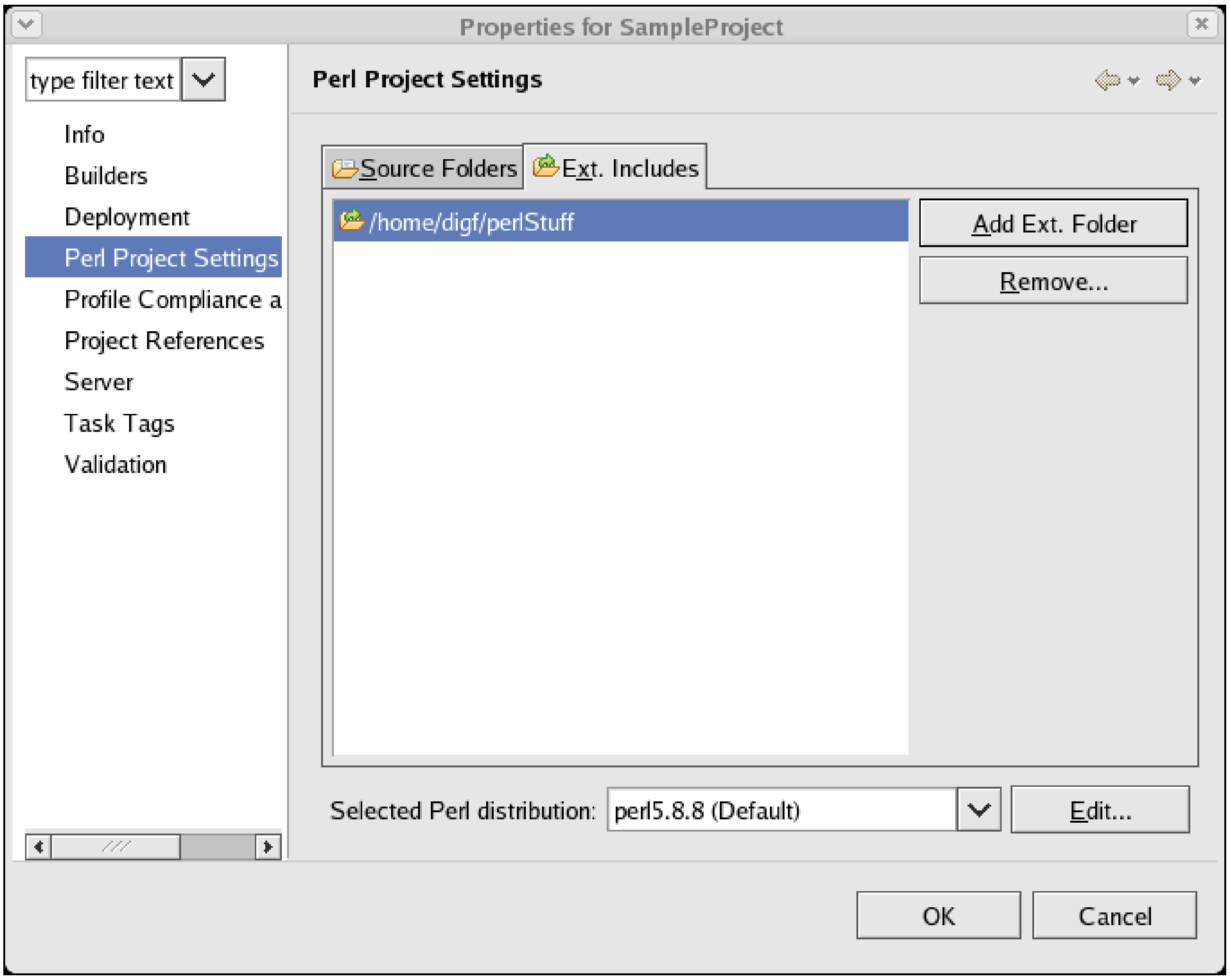
Here we assume you want to add one or several directories to @INC but you don't want to manage the files in these directories within Eclipse; i.e. you don't want to set up projects for these.

1. Open the Perl perspective and select *SampleProject*. Right click and select **Properties**. Select **Perl Project Settings**. Then click on the tab **Ext. Includes**.

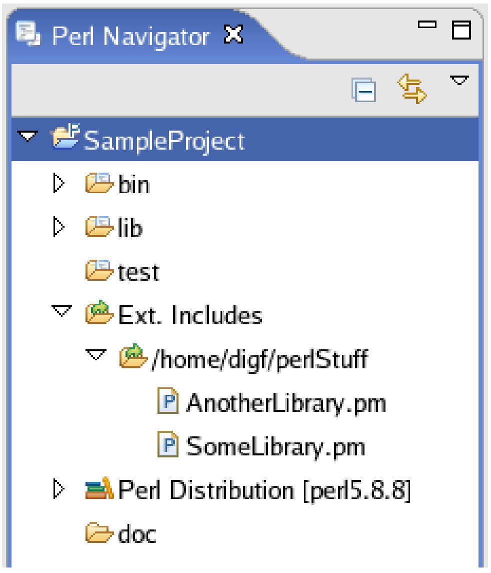


Selected Perl distribution: perl5.8.8 (Default)

2. Now click on **Add Ext. Folder**. Locate the directory you want to add and click on **OK**.

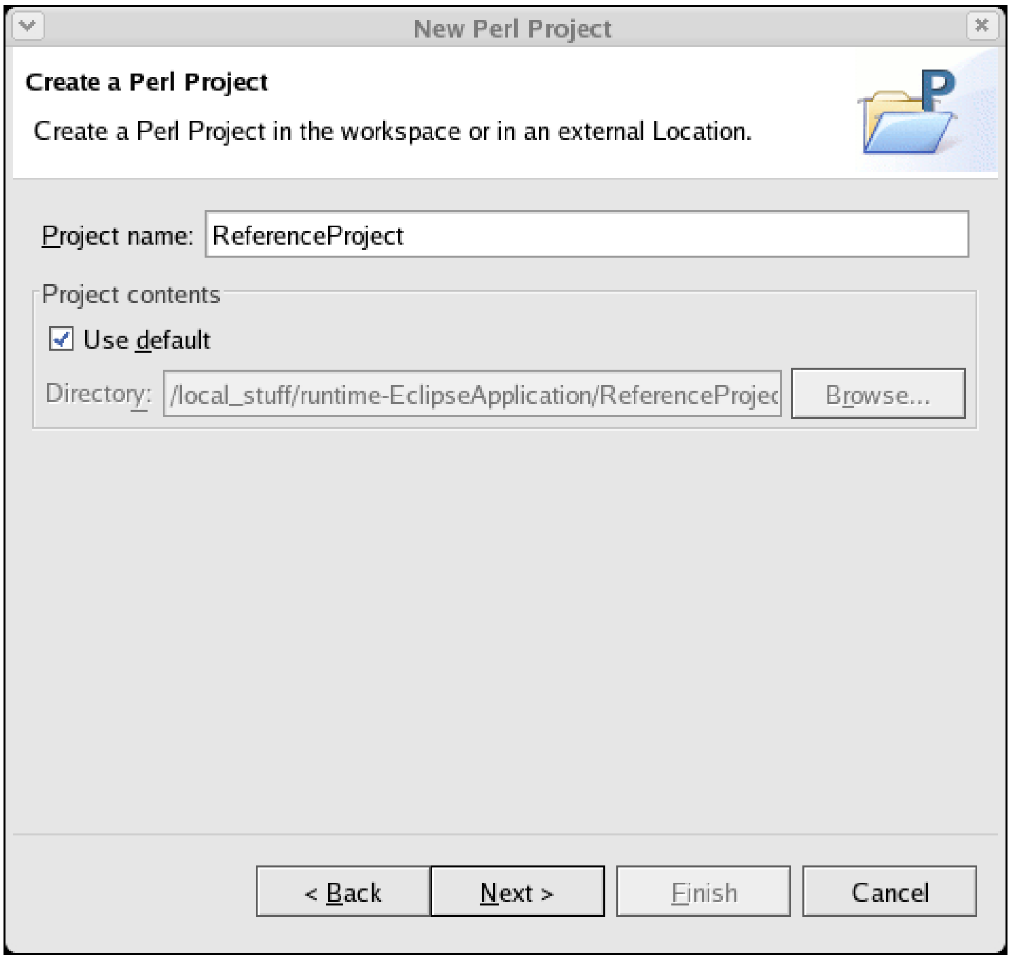


3. Close the preferences dialog and return to the Perl perspective. An additional folder was added to the Perl Navigator which lists the external includes.

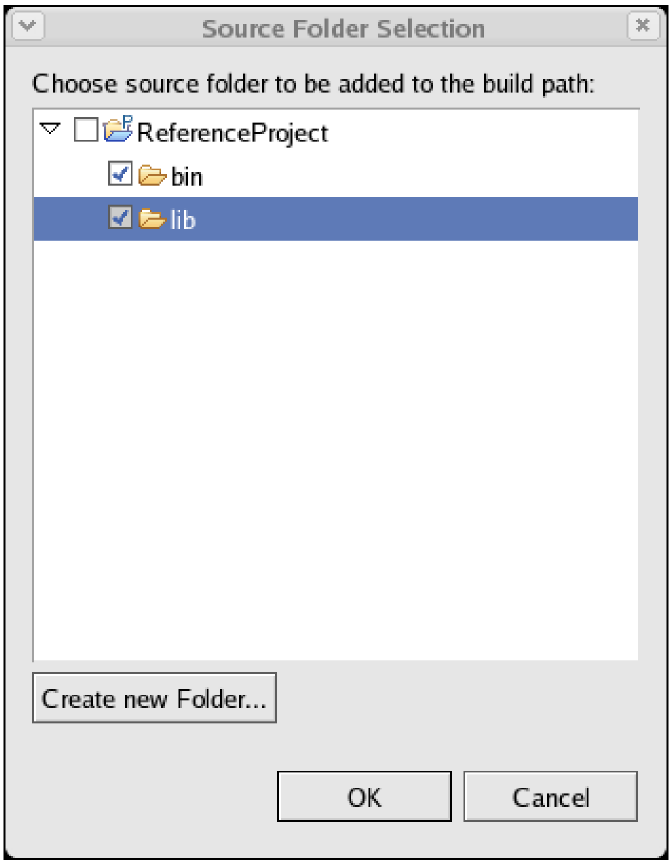


Adding a reference to an other project

1. Before we can add a reference to an other project we first have to create this other project. So switch to the Perl perspective.
2. Then click on **File > New > Project ...**. Select **Perl > Perl Project**. Enter *ReferenceProject* as **Project name:**.


New Perl Project
Create a Perl Project
Create a Perl Project in the workspace or in an external Location.
Project name: ReferenceProject
Project contents
Use default
Directory: /local_stuff/runtime-EclipseApplication/ReferenceProjec
Browse...
< Back
Next >
Finish
Cancel

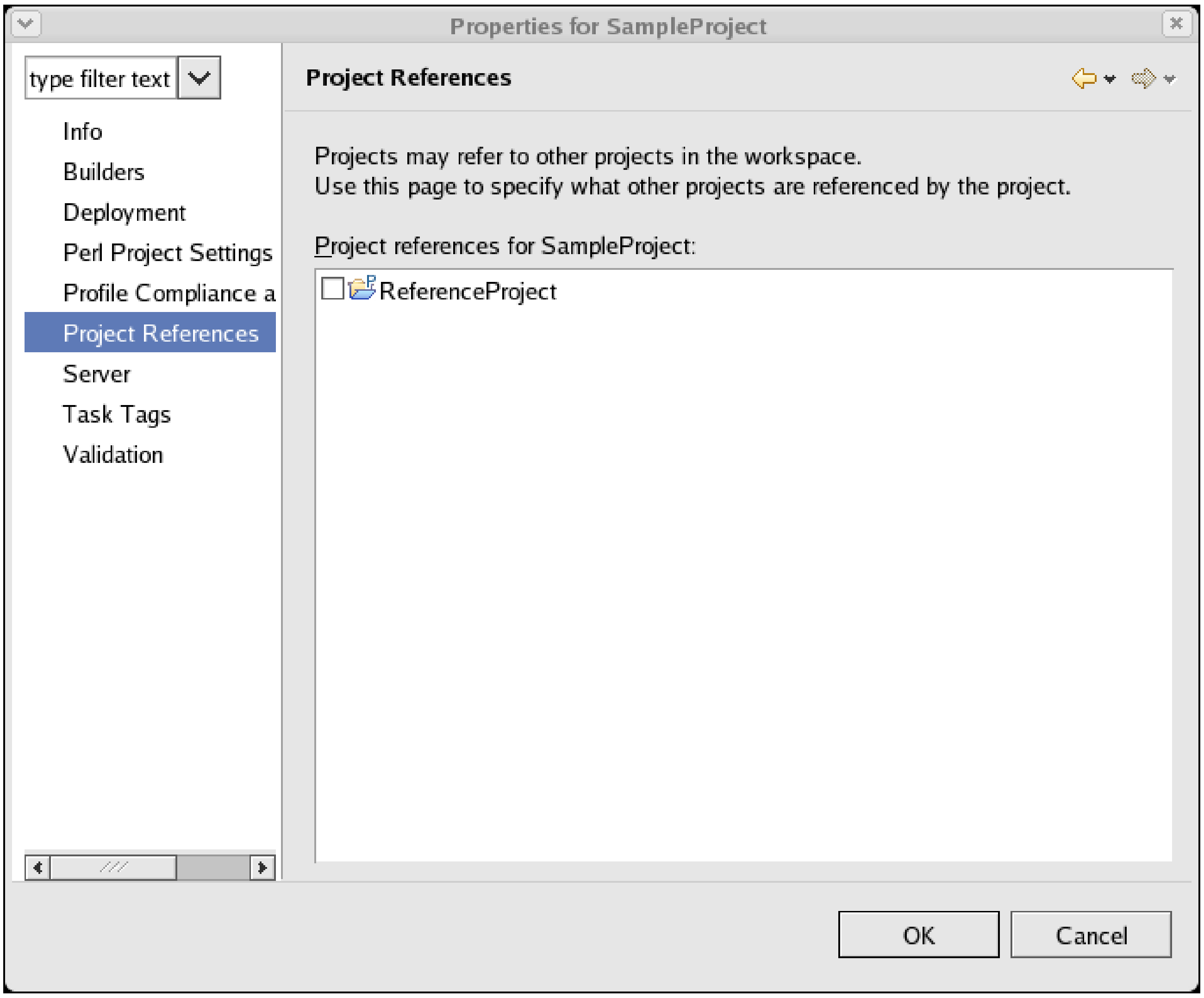
Click on **Next >**. Click on the **Add Folder ...** button and add the folders *bin* and *lib* to the project. Ensure the checkboxes before bin and lib are selected.



Close the dialog by clicking on **OK** and complete creating the project by clicking on **Finish**.

3. Now we're ready to reference the newly created project in our SampleProject. Click on *SampleProject* in the

Perl Navigator. Open the context menu and select **Properties**. Select **Project References** in the Properties dialog.



Project References

Projects may refer to other projects in the workspace.
Use this page to specify what other projects are referenced by the project.

Project references for SampleProject:

☐ ReferenceProject

As you can see in the above picture all projects which might be added are listed. Check the checkbox in front of *ReferenceProject* and close the dialog by clicking on **OK**. The *lib* and *bin* directories of the *ReferenceProject* are now added to @INC of the SampleProject.