What is OptiPerl

v4.2

OptiPerl will help you create CGI scripts in Perl, offline in Windows. It is a fully integrated developing environment for creating, testing, debugging and running perl scripts, directly or through associated html documents.

Read Setting Up to setup Perl in Windows and OptiPerl.

Features in OptiPerl

- Offline editing of CGI Perl Scripts.
- Complete emulation of a real server - scripts can be run indirectly from html documents.
- Live preview of the scripts in the internal web browser.
- Feature packed editor with syntax highlighting.
- Completely integrated debugging with live evaluation of expressions, watches, breakpoints, flow control.
- Remote debugging of scripts located on your web server, remote machine or via loopback.
- Code completion, and hints while programming.
- Automatic syntax checking.
- Box and line coding give a better view of your code.
- Saveable desktops.
- Code librarian that supports ZIP files and code templates.
- Context sensitive help on core perl and module documentation.
- Powerful query editor to create the environment and data sent when calling CGI scripts.
- Many tools like Text encoder, Perl printer, Pod viewer and other.
- Projects to organize and publish a set of scripts.
- Version converter to handle non supported perl functions in windows.
- Opening, saving and running scripts on remote servers.
- Sendmail and date debugging under windows.
- Printing script and exporting as HTML with syntax highlighting.
- Searching and replacing with regular expressions in projects and files.
- Backups using Zip files.
- File and Remote (FTP & Secure FTP) explorer.
- Plug-Ins.

Registering

Buy OptiPerl - Support this great software

OptiPerl is not Free! Some features have been disabled in this demo version. However for a small amount of money payable on-line with your credit card, you may download the full featured version.

Why should you buy this program? Well, consider the following:

- OptiPerl is a great learning tool. You can use it to learn Perl and CGI programming in the comfort of a visual editor, without loss of time and money because of loading from dos perl or being for hours on the internet.

So if you program for hobby or want to introduce yourself in a pleasant way to CGI programming, OptiPerl is for you.

- Professionals designing perl programs and cgi scripts use it. You can save much time by quickly creating, debugging and testing all the ideas you have increasing productivity.

For novice or advanced users, for hobby or professional use, OptiPerl is bundled with all the tools you need to make CGI and console scripts in Perl easy!

Registering will enable you to:

- Receive the full version of OptiPerl.
- Enable editing, debugging and running of unlimited sized scripts.
- Get all future upgrades. You will receive a password (that can be retrieved by e-mail if ever lost) to login to your personal update page which will always have the latest version, your registration info and any related files.
- Receive with priority technical support on OptiPerl.
- Get the full documentation of perl and apache, indexed and searchable.

Not convinced that you need it?

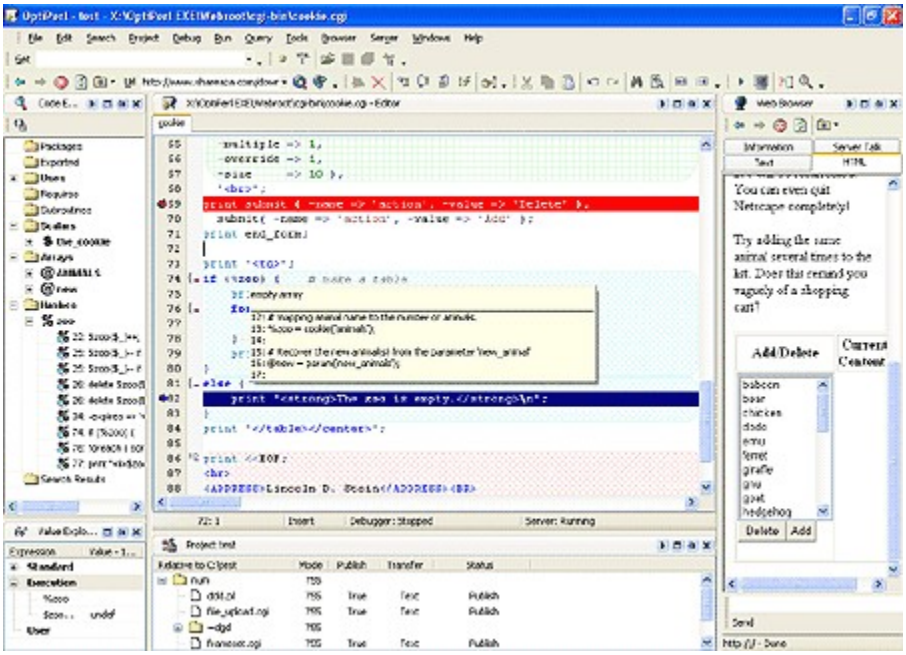
Please read the introduction to see how you can use it.
Or view OptiPerl's features Optically.

You can register with a credit-card worldwide. You can also choose from other means of payment like a bank transfer or a check. Follow the direction on the order page for more information.

OptiPerl Homepage:
http://www.xarka.com/optiperl/

E-Mail:
optiperl@xarka.com

Features Optically
Integrated Development



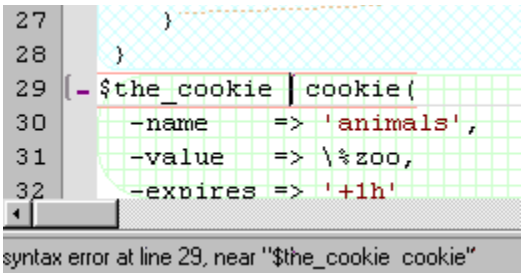
OptiPerl is the complete developing solution for Perl.
Scripts can be created, edited, debugged and run all offline in a Visual Environment.

Smart syntax parsing

```
1   my $var = 1;
2   sub supersub {
3   }
4
5   supersub($var);
6   super_sub($var);
7   super_sub($VAR);
45  print q<
46    <a href="url">
47    click here
48    </a>
49  >;
```

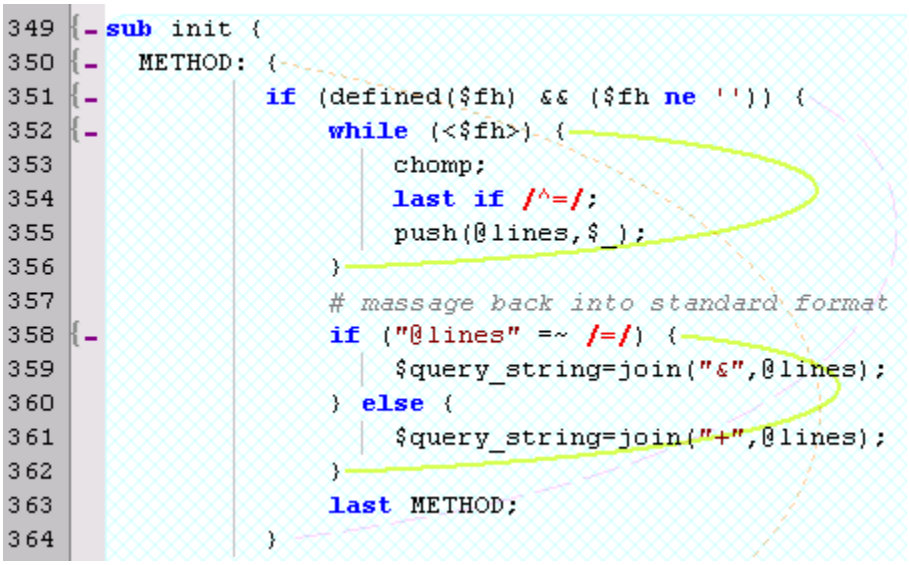
OptiPerl's syntax parser can color code even the most complex perl code. It will even switch to html coding if html code is detected. You can also select from five styles on the fly or create your own.

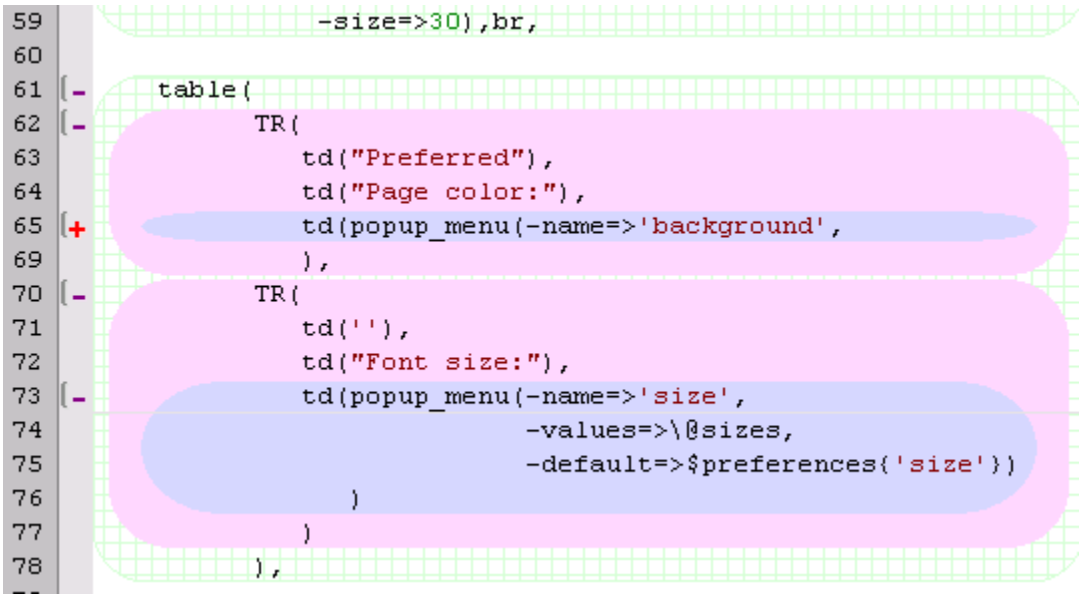
Automatic syntax checking



Error's and warnings are generated while you type, highlighted and listed.

Box & Line coding

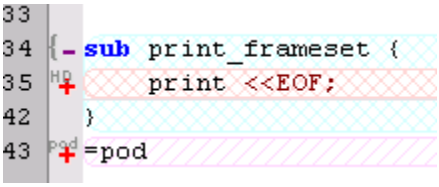




An amazing pioneer feature: Blocks of brackets and parenthesis can be colored and/or linked by lines and boxes, each with a different style depending on the level. Also here

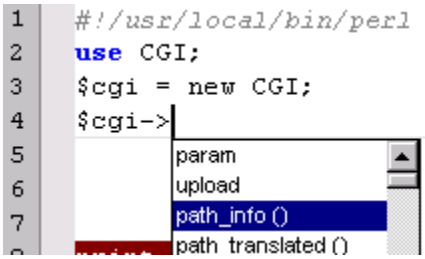
documents and pod elements can be colored in the background. Perl code can become an even better work of art!

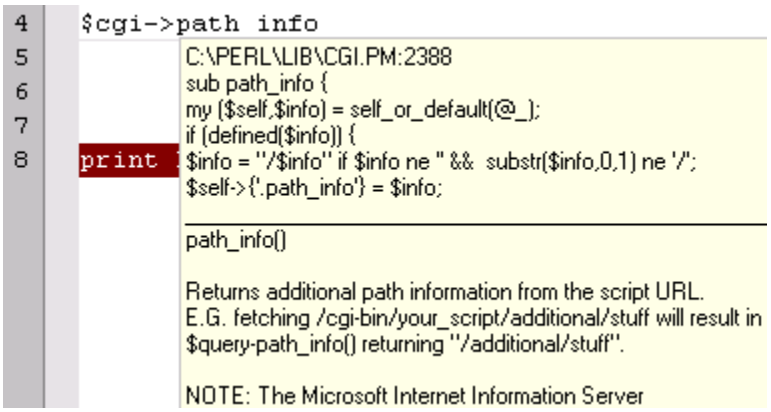
Code folding



Brackets, parenthesis, here document and pod code blocks can fold. You can also have parts of your code folded by default.

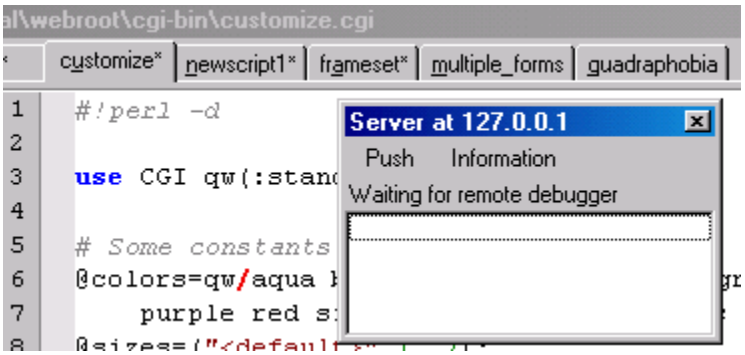
Code Completion





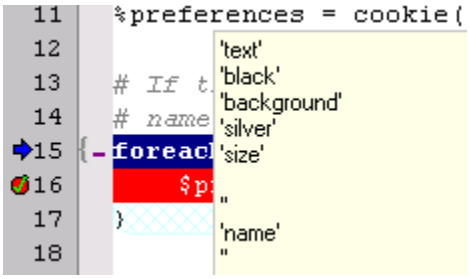
OptiPerl will do everything possible to get subroutines, variables and pod information from used modules in your script so coding will be easier.

Local & remote debugging



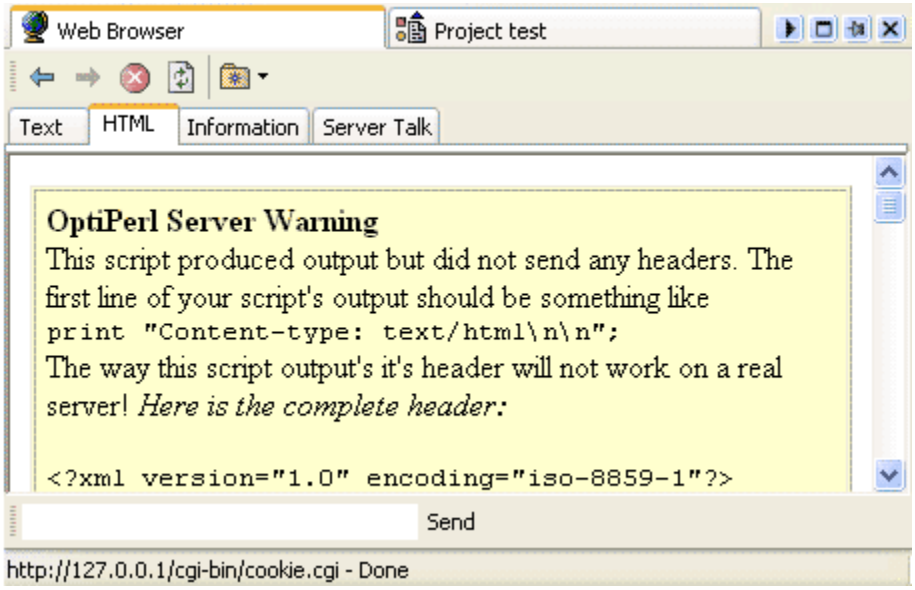
Debug any script your your machine, or debug it while actually running on you web host. Tracing line-by-line, breakpoints and watches are supported.

Live evaluation of expressions



While debugging you can evaluate anything moving...
As a pop-up hint window or in Watches.

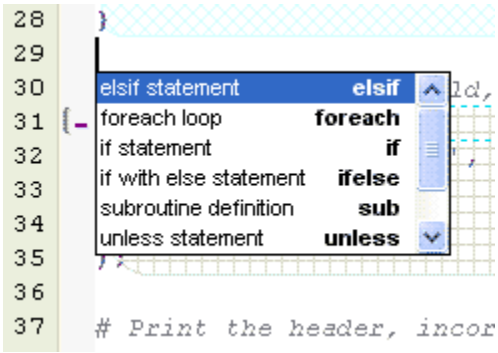
Internal & Third party server support

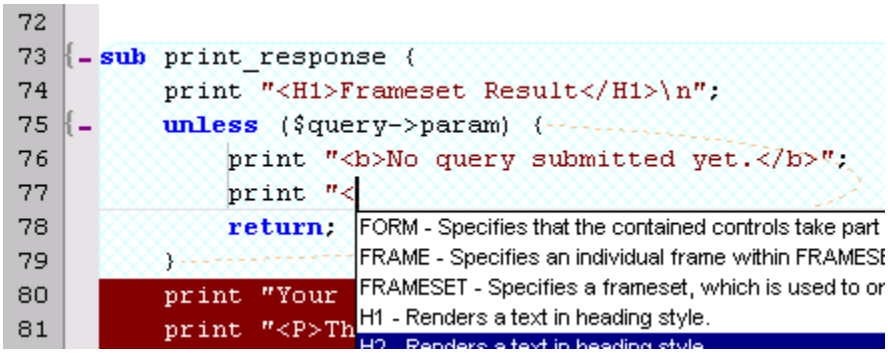


Internal Web Server emulates how your entire web page would be on-line. Instead of 500 errors, it analyzes the output and tells you what went wrong. Also integration with other

servers like apache is supported.

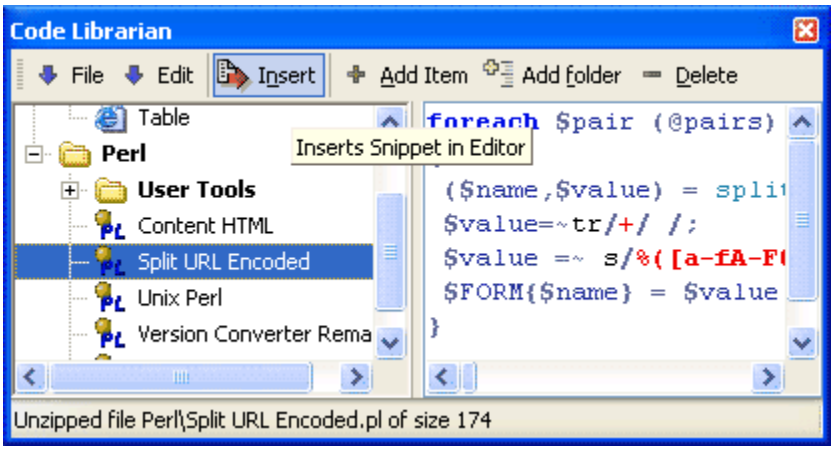
Rich Editing Features





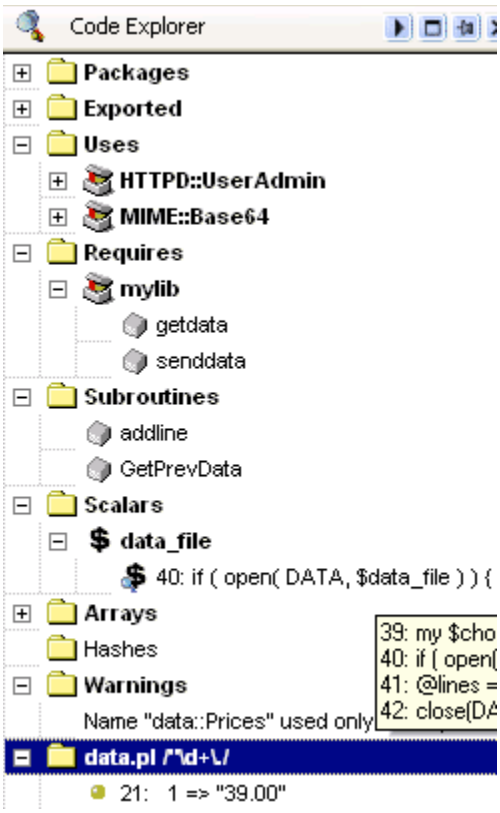
OptiPerl has templates, static code completion for html elements and internal perl functions, and many tools for editing.

Code Librarian



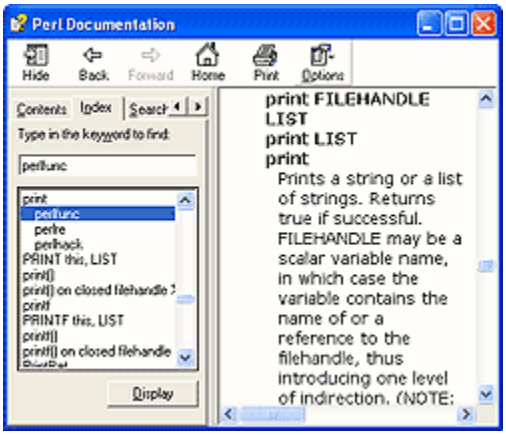
Code Librarian stores and shares perl and html code in a ZIP format. You can also open many zip files and drag files in your code repository.

Navigation in your script



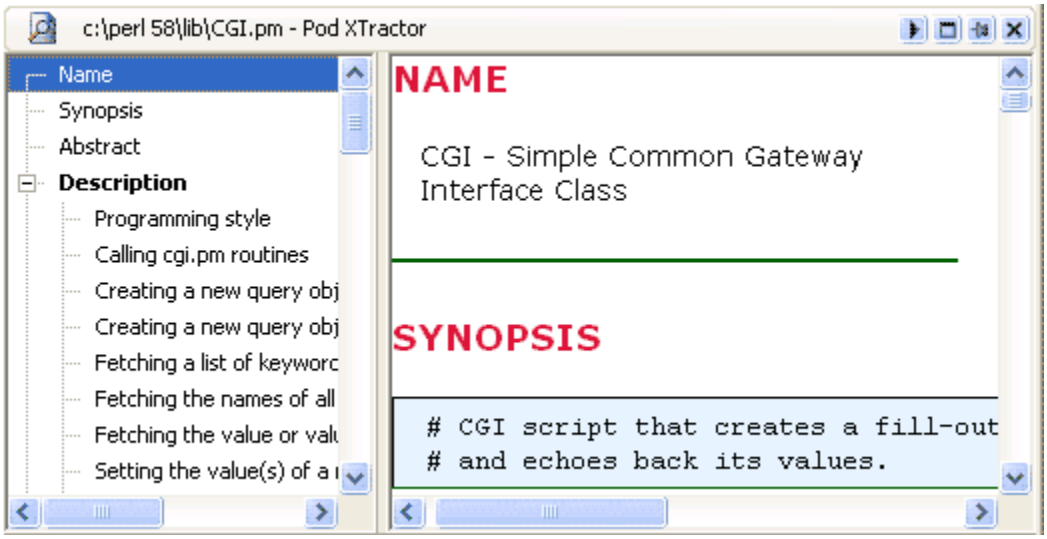
Code explorer lists elements in your program plus errors and warnings evaluated automatically. Subroutines in modules used are also listed and any pod info associated with each subroutine is shown if found. Public variables are shown with each occurence of using them.

Context - Sensitive Help



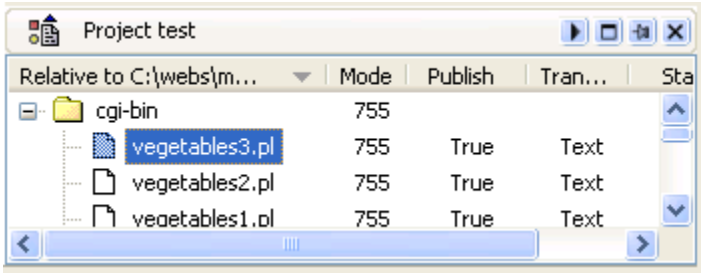
Context-sensitive help with all of perl's core and module documentation and apache documentation.

Pod Support



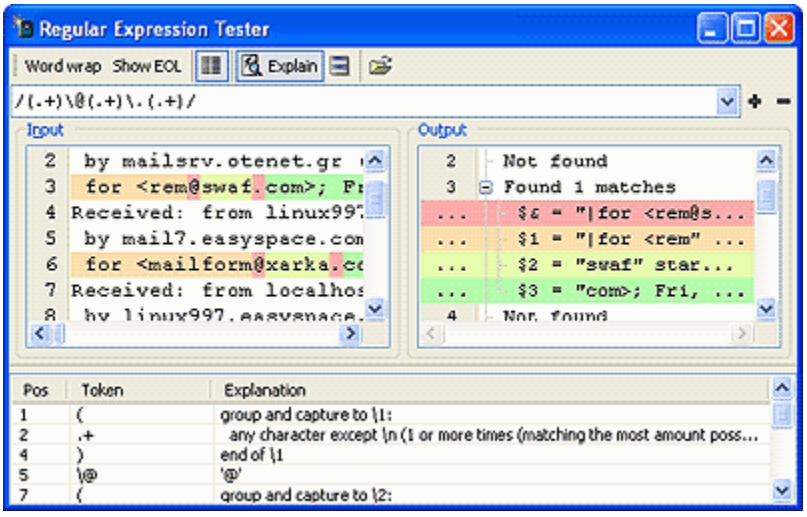
View POD information, embedded or standalone in a easy to use tree display.

Projects



Organize many scripts into a project and publish it.

Many Tools



Regular expression tester, perl printer, script uploader, File & Remote Explorer, perl reformatter and much much more!

Frequently Asked Questions
General Questions

Q: Is OptiPerl a language for making forms on web pages?
A: OptiPerl is not a language. It is a tool that helps programming in Perl. Perl is a language that makes programs for computers. It just happens that it is also good for making cgi scripts, those used for web pages. Actually almost every time there is some kind of interaction between you and the browser, a perl script is being used, from input boxes to search engines..

Q: Can OptiPerl help me learn Perl?
A: It can help you, especially if you want to deepen in CGI programming, because you can do that offline in windows. However you will not be able to magically do stuff, you must still learn Perl.

Q: Why isn't it easy to make windows run CGI scripts?
A: Windows was not made to do this. For script to run from the webbrowser offline, a small server must be loaded when we want to in windows, that actually only serves itself, the user of the same computer. OptiPerl does this internally so you don't have to bother with this. However running an external server like Apache is also supported.

Q: I just started learning perl, I spent some money for a nice book, but I'm not sure yet if I need OptiPerl.
A: As noted above, OptiPerl can help you learn perl easier and faster. And not only that - you are welcomed to use the unregistered version for 30 days. Even with the limitation it has on the size of the scripts, it can load most examples and small scripts. Whenever you feel you need OptiPerl, go ahead and register.

Q: OptiPerl can run my script in many different ways - Which is the best?
A: The best is using the internal server, as you can also load html pages and see how your script works when the pages load it, just like the real thing. Running without a server at all, is good when you just want to run something you just thought of quickly without even saving, or for non-CGI perl. For a complete test of your entire web page, you can use Apache (even though in most cases just having OptiPerl loaded with do as fine!).

Q: If OptiPerl's internal server is so great, why so much support for running with apache too?
A: OptiPerl is also used by professionals for web authoring. Using apache is the closest possible thing to a real server that serves the internet (most use apache also). You may be able to test other aspects of web authoring that do not have to do with perl scripts like apache mod's, .htaccess files etc.

Q: When I press "check for update", I can see there is a new version out. How do I download?
A: If you are using the trial version, then just download again and install. If you are a registered customer, go to the password protected page sent to you when you registered (you will be notified anyway with an e-mail when a new version is out).

Q: Is OptiPerl compatible with Windows 2000, NT and XP?
A: Yes it is. You can also use as an external server the Internet Information Server included with 2000 and XP (read the according section in the help file).

Working with OptiPerl

Q: When I load up OptiPerl I see parts of the desktop. Is this normal?
A: OptiPerl is made using many small windows, not just one big one. So you will see whatever is behind those windows, either it is the desktop or another program. Note that you can drag and drop text into the editor from other programs. If you do not like a multiple window interface, select menu Windows / Single window interface.

Q: How do I add items to the Code Explorer?
A: Code explorer is a navigational tool. It lists elements in your code and is built automatically whenever you edit your script. To go to one of it's elements, double click the line. You can however edit the names of variables in code explorer by left - clicking and pausing on them. Doing so is like performing a search and replace in your script.

Q: I closed the editor window and now I can't open it!
A: Select "Show Editor" from the Editor Menu.

Q: Can I find an opening/closing bracket in the editor?
A: Hold down Control while moving the mouse.

Q: Pressing enter does not move to the next line when typing in text areas in the web browser
A: Use Ctrl-M instead of enter.

Q: How can I dock and undock windows?
A: Dock it to the main editor window by dragging it. If you want to place it over the main editor without it being docked, hold down the ctrl key while dragging it. Undock it by double clicking on its title bar.

Q: I am using a dark background in the editor. I can set light colored text styles for the main editor, but all the other windows used in OptiPerl have black colored text that does not show! How can I change the color of text for the other windows?
A: This is also modified from styles, by changing the style element "White space" (the first one in the list). So if you want dark backgrounds, make sure you select a white font for this element, as this is the color used for all of OptiPerl's windows.

Q: I have an old computer under 300 MHz and OptiPerl is very slow what can I do?
A: Read the section in the help file Options / Other settings / Improving Performance.

Setting Up

Q: What do I need to run OptiPerl?
A: You need Perl!. See OptiPerl's homepage under support for a perl distribution we recommend.

Q: Do I also need a web server?
A: No, as OptiPerl can turn on it's internal web server whenever you need it. You can also run scripts without a server at. But if you ever need it, you can use an external server like Apache instead of the internal server.

Debugger

Q: Can I load the debugger passing a command line (my script is not a CGI)
A: Enter the arguments you want under menu Run / Arguments.

Running

Q: When I press run, the first line in the browser is "Content-type: text/html". This shouldn't happen!
A: You have unchecked "Run with server" in the Server Menu. When this checked, then the script is run through the web server and loaded as a http address. If it isn't then the scripts output is sent to a temporary file, and the file is loaded in the web browser as a file url. Note that this sometimes can be useful. For example, try running a script that creates a cookie without the internal server loaded. The cookie is not created, instead the request that would be sent to the client is shown as the first line. This way you can easily debug cookies. For the cookie to be created of course, load the internal server and select "Run with Server".

Q: What is "Set Starting Path"?
A: In special cases you may need to set the starting paths of scripts, mostly for console applications. This selection also affects when running with the console.

Q: When I run in the console, the screen flashes first.
A: To fix this, open from the start menu a dos command prompt. You will find it full screen; press alt-tab to make it a window and then exit.

Internal or External Severs

Q: When I run a script pressing F9, I am prompted to connect to the internet!
A: If you get this, press OK. The connection will not be made, but the script will be loaded.

Q: When I load OptiPerl's internal server, I get an error message.
A: Make sure that in only one instance of OptiPerl you have loaded the internal server. Or select in the option dialog "allow only one instance". Another possibility is that you might have an external server loaded like apache and Optiperl didn't find it. If you use apache, read more in the corresponding section. If you have Windows 2000 or XP, you may have Internet Information Server enabled that causes this problem with the internal server. Again there is more in the rest of the help file.

Q: I have done the above, but I still get an error.
A: If you are using a firewall, allow "pass through" for OptiPerl in the firewall program to.

Q: Sometimes I get in the status line of the web server "Timeout Occurred", and the output of the webbrowser is wrong.
A: When this happens, it is because the script entered an endless loop or some error prevents it from exiting, so OptiPerl will try to close it. A script that does this has a bug and is not a correct program for CGI use. Try debugging it. One common cause is when the script tries to load an external program that does not exist, causing it to wait endlessly. For example if you use the date command like this:

```
$date_command = "/bin/date";
$date = `$date_command +"%m/%d/%Y %T %Z"`;
```

In windows, this would cause the above error. OptiPerl can help you however if you are using date or sendmail, see the help file.

If this is not the case, because your script is just slow and needs a lot of time until it sends out results, then increase the timeout value in the Options Dialog.

Q: Sometimes the changes in the script I make are not reflected in the web browser. What's wrong?
A: If you see this happening, then try going to the Control Panel / Internet Options / General tab / Temporary Internet Files / Settings and select "Every visit to the page". If you are running in OptiPerl's internal browser, this will not happen, however it might happen in an external browser.

Q: When I first load OptiPerl, load the internal server and run a script, I have to wait 50-60 seconds until it shows (but afterwards the scripts work fine)!
A: Go to the Control Panel / Internet Options / Connections tab / Lan Settings button / Automatic configuration and remove the check mark "Automatically detect settings".

Q: My command line is not passed to the script.
A: It doesn't make sense to send a command line to a CGI script. CGI's only accept queries using a GET or POST method.

Q: I get the output "Forbidden".
A: Make sure that the script you are trying to run is located under the folder set in the "Internal Server" root path.

Q: When I try to run a script I am prompted to download, or see it's text.
A: Make sure that the path to perl is correct in the Options/Perl, and the associations are OK in Options/Internal Server. Press the "Default" button to make sure. If you are using an external server, you will need to read it's documentation on how to run CGI scripts.

Problems

Q: When I first load OptiPerl, I get an error message and the program terminates. What's wrong?
A: You probably have an old version of Windows 95 without Internet Explorer 5 or above. Try installing a newer version of Internet Explorer, or installing Windows 98 and above.

Q: When I load OptiPerl, I get a Dr. Watson application error (windows NT)
A: The same applies as above. Windows NT 4 ships with IE 2 and this will not work. You must upgrade to IE 5.

Q: Why is the "Perl documentation" and "Search for declaration" grayed out?
A: Either you have not installed the documentation (if you are a registered user) or you need to upgrade Windows Html Help to version 1.1 or newer. Get the update at:
http://officeupdate.microsoft.com/2000/downloadDetails/Hhupd.htm

Q: When I right click on a module in Code Explorer, the module does not open in the editor.
A: Make sure the search path (%INC) is correct in Options/Perl. Press the default button to make sure. You can also set the %INC path per project under project options. See also the section "How OptiPerl finds module" in the help file.

Some free web hosting services with a CGI directory

Geocities (http://geocities.yahoo.com/)
NetFirms (http://www.netfirms.com/)
Pro Hosting (http://free.prohosting.com/)
Tripod (http://www.tripod.lycos.com/)
Virtual Avenue (http://www.virtualave.net/)
Xoasis (http://www.xoasis.com/)

Note: If you would like to be included here, please send an e-mail to optiperl@xarka.com

Using Apache as an External Web Server (Optional)

Q: Why must I enter the exe name of the server program I am using (for example "apache.exe") in the Options Dialog?
A: From this name, OptiPerl searches often in the memory of the computer to see if the server is loaded. If it senses that it is loaded, then it will turn off it's internal server automatically and also gray out the menu item to load it. You cannot have two servers loaded at the same time. After OptiPerl senses that an external server is loaded, it will use the paths selected in the Options Dialog under external server. Read more in the section "Using an External Server"

Q: But I haven't loaded apache, and still the internal server option is grayed out.
A: Probably you installed apache in a windows 2000/Nt system, and apache is running as a service, and not in a dos-box. If you need to turn it off, go to the computer management and press Stop on the apache service.

Q: How do I load Apache?
A: Run in from OptiPerl's Tool menu, or select from the start menu the item "Start Apache".

Q: I get "External Server Loaded" even though apache is not loaded.
A: If you have NT/2000, apache could be loaded as a service.

Q: I am getting a 500 error when running.
A: First make sure your she-bang line (the first line) points correctly to perl. For example, it could be #!c:\perl\bin\perl.exe

Q: When I load Apache, the window opens and then closes immediately, and I don't get a chance to read what it says. What can I do?
A: Apache must be giving an error message. To be able to see it, load to command prompt to go to dos and load from there. The window will not close, so check what the error message is. Also check the last line of the error log.

Q: Is Apache loaded if the window opens and then closes immediately?
A: No it is not. You must always have a console window loaded, that can be minimized, but not closed if you want to run CGI scripts in OptiPerl. This does not apply if you are using Windows 2000 or NT. In this case check if apache is running as a service.

Q: What is the correct way to close the Apache console window when I've finished using OptiPerl?

A: An almost-correct way that's fine if you are using Apache just for offline work and not as a web server is selecting the window and pressing Ctrl-C.

Q: The error message in Apache's log is "failed to get a socket for port 80".
A: This has to do with the network configuration in windows. Go to Settings/Control Panel/Network, and check what gets loaded there. A good idea is to delete the items there and re-install them, but make sure you know what you are doing. What must be there for sure is a TCP/IP adaptor in Protocols. This applies for both Windows 98 and 2000/NT.

Version history

List of additions in Optiperl since version 1.

All customers since the first shareware release have received for free the updates, and will continue to get free updates in the future. This is a policy of Xarka Software.

OptiPerl is an ongoing project. Each new version comes from implementations of ideas that users of it have and that is why it is the best visual environment for perl.

We thank you for it's success!

Version 4.2 (November 2003)

Added:

- Function call stack
- Conditional breakpoints
- Pod node in code explorer
- Drag & drop between File explorer, Remote explorer, Project manager & Windows explorer
- Files in project not found marked with red
- Search highlights (menu search)
- Trim lines, unless line is empty (options dialog / Tabs & Lines)
- List of open files menu
- Recursive deletion of folders in Remote Explorer
- Word wrap at window width (options dialog / Visual)
- Different pop-up menu when right-clicking on tabs (tab pop-up menu)
- Ability to select & delete many files in project manager
- Reset button per page in options dialog
- Ability to search & replace in many files using a simple string (not only regular expression)
- Remote sesions are kept alive by sending commands at regular intervals (options dialog / Running)
- Ability to include external variables in query editor (methods menu)
- Open in pod extractor opens .pod file if found
- Back button added (default shortcut: Alt-left arrow)
- Ability to save open files being edited per project
- Drag & Drop in project manager
- Added browsing the results of an advanced search, using search category / Next & Previous Match (shortcuts: Alt - Up & Down)
- String "Page x of n" added when printing
- Double click in session explorer window opens file in editor

Fixed:

- Rare cases of code that created problems with color syntax parser
- Sometimes editor did not shut down
- Minor find dialog problems
- Minor code completion problems
- Session setup window problem when using large fonts
- Replacing in multiple files opens the files only when a match is found
- Auto-view window faster
- Did not close normally when shutting down windows
- Regular expression tester sometimes showed "perl dll not valid"
- Cases of optiperl not loading on some XP systems
- The last line of scripts was cut off if it was empty

- When remote debugging, sometimes the script was not donwloaded correctly

Version 4.1 (April 2003)
Major upgrade.

Added:
- Remote debugging
- Box and line color coding
- Code completion and hints
- New interface with multiple desktops
- Customizable menus and toolbars
- Better color syntax coding and support for many languages
- Multiple color syntax coding
- Sendmail and date support
- Auto syntax checking
- Opening and saving scripts via FTP or Secure FTP
- Remote Explorer
- Print preview
- Regular expression search and replace
- Better support for external server with aliases and running remote files
- Backing up files
- Plug-ins

Code Librarian:
- Storage of code snippets are now in common ZIP files, however the original window has been preserved.
- Can open multiple windows with different ZIP files
- Snippets may be dragged between windows

Code Explorer:
- Searches POD documentation and extracts information about subroutines
- Added node with errors and warnings about script

Editor:
- Right click menu can be customized
- Better tab support
- Tab lines and line coloring

Options:
- Better options dialog with over 200 options
- Customize toolbars and menus window to edit and create new menus and toolbars

Internal Server:
- Completely new internal server. Will not allow any 500 errors, instead will give a detailed explanation of what went wrong
- Access/error log removed and replaced with "Server Talk" tab in web browser
- Optimized to work with remote debugging via loopback to debug scripts that are invoked via web pages

Browser:
- More info on the "Information" tab
- Spy HTTP proxy shows client requests and server responses when navigating the internet

Tools:
- "FTP Send" tool removed.
- Remote sessions database added where FTP and Secure FTP sessions can be setup for the Open/Save remote file dialog and Project Publishing.

User Tools:
- User tools can be moved to toolbars, menus and get a shortcut key and graphic assigned.
- Can integrate better with OptiPerl. Supports reading and writing to STDIN and STDOUT.

Projects
- Tree-like view that includes folders
- More options can be overridden in projects
- Removed options for FTP sessions, since they have moved to the Remote transfer sessions dialog
- Scripts with a "Use Lib (...)" path are parsed and the directories are appended to the selected @INC folders, to help Code Explorer search for modules

Version 3.5 (March 2002)
Major upgrade that has the following changes by category:

Code explorer:
- Rename variables using left-click
- Exports node
- Larger summary in hint window
- Better context handling of variables
- Increased speed of regular expression searching
- Minor bug fixes

Debugger:
- Ability to add breakpoints in required and used modules of the script, even in run time evaluated modules
- Gutter graphics show valid and invalid breakpoints after debugger has started
- If syntax errors are found then the syntax error checking window is loaded
- Bug with certain versions of Windows NT fixed
- Better "Evaluate Expression" window
- You can change values of variables in the "Watches" window

Editor:
- Comment out multiple lines
- Synchronized scrolling option when a secondary editor window is open
- Holding down Control while moving mouse over a bracket shows corresponding one
- Typing brackets highlights corresponding one
- While moving in code, the subroutine in view gets highlighted in the code explorer. By selecting Search/Find Subroutine, it will also get focused
- Improved find & replace dialogs
- Fixed problem with international keyboards
- Regular expressions highlighted in color syntax parser
- Increase and decrease indent buttons

Environment:
- Editing of shortcuts possible

- More options in the Options dialog
- Alt - shortcut keys can be used to access the menu commands and tabs in the editor
- Support for extended keyboard
- Can do a regular expression find on the output and pod viewer browsers
- Minor changes - bug fixes

Running:
- Better tools support
- Selection in Query Dialog to import the form data of an html page (useful for debugging)
- Selection of starting path of the script possible
- More options when running in console
- Support for third party external browser
- Expanded syntax check evaluates first required modules

Projects:
- Publishing through firewall possible.
- Internal and external server options saved with each project

Tools:
- Added File Manager that can do a very fast regular expression search in entire folders
- Improved external tool support
- Backreferences listed for all lines matching pattern in regular expression tester
- View log option in perltidy tool
- Improved perl printer
- Improved pod extractor

Version 3.4 (September 2001)
- Improved code explorer
- The external programs run by the internal server can be set up
- Pattern search in entire project
- Added "Auto-View" tab in web browser, to view a custom file at real-time.
- Added File Compare tool.
- "Find declaration" added. Holding down Control and clicking the mouse on a declaration will search for it and open it in the browser

Version 3.3 (June 2001)
- Project support

Version 3.25 (May 2001)
- Regular Expression tester with "Explain"
- Ability to open many editor windows
- Right click on modules in code explorer
- Monospace font for the editor limitation removed

Version 3.2 (April 2001)
- Perl tidy tool (source code reformatter)
- Information tab in web browser

Version 3.1 (April 2001)
- RegExp Tester
- Minor Bugfixes

Version 3.0 (March 2001)
- OptiPerl is the new name of "Visual Perl Editor"
- Completely rewritten user interface with major improvements.
- Embedded server. Now apache is not required (but well supported)
- Feature packed editor with syntax highlighting
- Improved Code librarian and code templates
- Context sensitive help on core perl and module documentation
- Many new tools
- Improved Script uploader

Version 2.6 (Octomber 2000)
- Minor improvements

Version 2.5 (April 2000)
- Internal Error testing

Version 2.0 (February 2000)
- First shareware release of Visual Perl Editor

Version 1.0 (December 1999)
- Visual Perl Editor was released as freeware. It was the first editor for cgi oriented perl using apache server as a base for running perl scripts. The basic design of it (two panels, one to edit the script and another that showed the output as a web page) has been followed ever since.

Credits

Many thanks to:

- Domizio Demichelis - http://perl.4pro.net/

For extremely helpful suggestions and ideas on improving OptiPerl.

Also:
- John Drago
- David J. Marcus
- J. Walker

License Agreement

EVALUATION VERSION
This is not free software. You are hereby licensed to use this software for evaluation purposes without charge for a period of 30 days.

COPYRIGHT RESTRICTION
Xarka software name and any logo or graphics file that represents our software may not be used in any way to promote products developed with our software. All parts of Xarka's software and products are copyright protected.

DISCLAIMER
THIS SOFTWARE IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT WILL THE AUTHOR BE LIABLE TO YOU FOR ANY DAMAGES, INCLUDING INCIDENTAL OR CONSEQUENTIAL DAMAGES, ARISING OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. YOU ACKNOWLEDGE THAT YOU HAVE READ THIS LICENSE, UNDERSTAND IT AND AGREE TO BE BOUND BY ITS TERMS.

DISTRIBUTION
This software may be distributed freely in its original unmodified and unregistered form. The distribution has to include all files of its original distribution. Distributors may not charge any money for it.

OTHER RESTRICTIONS
You may not modify, reverse engineer, decompile or disassemble this software in any way, including changing or removing any messages or windows.

Copyright

Third-Party software

- Microsoft, Windows NT, Windows XP, FrontPage and the Microsoft Internet Explorer logo are registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.
- Netscape and Netscape Navigator are registered trademarks of Netscape Communications Corporation in the United States and other countries. Netscape's logos and Netscape product and service names are also trademarks of Netscape Communications Corporation, which may be registered in other countries.
- ZIP Library from "Info-Zip" http://www.cdrom.com/pub/infozip/ & http://www.geocities.com/SiliconValley/Network/2114/zipbeta.html
- Perltidy is Copyright (C) 2000, 2001 by Steven L. Hancock
- Package YAPE::Regex::Explain is work of Jeff "japhy" Pinyan, CPAN ID: PINYAN
- Package Text::Balanced is work of Damian Conway, CPAN ID: DCONWAY

Examples are from the book "Official Guide to Programming with CGI.pm" by Lincoln D. Stein - John Wiley & Sons, Inc. - ISBN 0-471-24744-8 - http://www.wiley.com/compbooks/stein/index.html

Setting Up

To use OptiPerl you need a distribution of Perl for Windows. Most perl distrubutions will work fine.

If you first loaded OptiPerl and you had perl installed, then OptiPerl should have already found it.

If you do not have perl installed yet then:

1) Download perl:

You will find the distribution we recommend on OptiPerls support page:

http://www.xarka.com/optiperl/support.html

2) Follow the instructions on the same page to install.

3) When you load OptiPerl, it should find it. If you get an error message that perl was not found, go to menu Tools / Options / Perl section and change to the location of perl.exe according to were you installed, for example `c:\perl\bin\perl.exe`

4) You are ready! If you are new to CGI Perl programming, read more in Basic Stuff.

 OptiPerl has an internal server to help developing CGI scripts, but also supports third party servers like apache. You can also use mySQL. There are also some user tools set up for CVS.

Basic stuff

If you don't know much about how to use CGI scripts in your web page, here is a very brief introduction.

Make sure you have installed first perl as noted in Setting Up.

Let's assume you want to create a web site with some forms. Do the following

1) Select "New html file" and enter the following.

```
<form action="cgi-bin/basic.cgi" method="GET">
Enter some text here:
<input type="text" name="text" size=30>
<input type="submit"><p></form>
```

This creates an input on your page for the user to enter text. Most Web-Editors allow you to create stuff like this automatically. Notice the GET method used here. When the user presses "Submit", the browser will call the URL:

```
http://127.0.0.1/cgi-bin/test.cgi?text=what_you_typed
```

Under the path you installed OptiPerl, you will find a folder "webroot" (like c:\program files\OptiPerl\webroot). Save your file there with the name "basic.html"

2) Select "New Script" and enter the following.

```
#!/usr/local/bin/perl
print "Content-Type: text/html";
print "$ENV{'QUERY_STRING'}<br>";
```

Save in "webroot\cgi-bin" as basic.cgi.

Now time to run. Make sure that you have a check-mark on "Server/Internal Server Enabled" and the root path under "Server/Internal Server Root" is "c:\program files\OptiPerl\webroot" (or accordingly where you installed OptiPerl).
Also make sure "Server/Run with Server" is checked too.

Go back to your html document "basic.html" pressing the tab in the editor and press "F9" to run.
You should see the html document in the web browser. Enter something in the text box and press "Submit"

You should see the result in a dynamically created html page. That's the basics!
If you get an error, please make sure you read about "Setting Up".

3) Now to send this to an Internet server, you might need to change the first line (the location of perl on the web server). If your internet web server has cgi enabled with a cgi-bin directory contact your provider for what this line should be.

Also read more in the FAQ.

Creating a HTML page with a form

HTML Pages can be created manually from OptiPerl's editor, or a HTML authoring tool can be used. Most HTML editors support creating forms and adding elements in it like textboxes and buttons.

1) Microsoft's Frontpage

In Microsoft Frontpage, select menu Insert / Form / Form element to add a form. This will also add automatically "submit" and "cancel" buttons. Then from the same menu add other elements you want. For an example we will add a "One liner text box". Right click on the text box and select "Form field properties". As a name enter "`textfield`". Right click again and select "Form Properties", check "send to other", select "CGI script" and press the Options button. Select as method "POST" and as action enter the perl script that will be called when the submit button is pressed, like

```
/cgi-bin/form.pl
```

See the next section on where to save the html file and how to create form.pl.

2) Manually in OptiPerl

Select New / HTML and enter:

```
<form method="POST" action="/cgi-bin/form.pl">
  <p><input type="text" name="textfield" size="20"></p>
  <p><input type="submit" value="Submit" name="submit"><input type="reset"
value="Reset" name="reset"></p>
</form>
```

again in the first line the method and action is entered.

Where to save the html document

Included in OptiPerl's installation folder is the folder `webroot` used for all it's examples. Save the html to the folder
`c:\program files\optiperl\webroot` with the name `test.htm`

Creating form.pl

Now we will create the script that processes the input (what the user entered in the text box) and prints a page depending on it. Select menu New / Script and enter:

#!perl

```
use CGI qw(:standard);
print header;
$text = param('textfield');
print "<html></html> $text <body></body";
```

Save this as form.pl in the folder `c:\program files\optiperl\webroot\cgi-bin`
This is a simple script that only prints back the text field.

Running

Go to menu Server and select both "Run with server" and "internal server enabled". Afterwards, make sure the last line of the menu shows "(internal) c:\program files\optiperl\webroot".

Open the html page (`test.htm`) in optiperl and press run. Test the submit button.

What is going on

Three things are important here:

1) Notice the form of the html page:

```
<form method="POST" action="/cgi-bin/form.pl">
```

This tells the browser to use a POST method (read more about methods) to send the text of the edit box. The action tells the browser what to do when the submit button of the form is pressed; here it will call `/cgi-bin/form.pl`

2) Also notice the path to the script used. We are calling form.pl in the folder /cgi-bin. This is why we saved form.pl in the `cgi-bin` folder of `c:\program files\optiperl\webroot` and also set the internal server webroot to `c:\program files\optiperl\webroot`.

3) To see what exactly happens when you press the "Submit" button, see the "Server talk" tab of the web browser.

Sending the query manually with the query editor

You can also run form.pl directly and have optiperl send the information it needs. Open the query editor, and from the "methods" menu enable the POST method and disable the other. Enter a name and value pair in the POST tab:

Name: `textfield`
Value: `text entered in text box`

Using the remote debugger via loopback

We will debug here the script cookie.cgi. Open the script in the editor and enable "Internal server" and "Run with server". Run this script a few to understand how it creates a session with the user using cookies. Let's say we want to debug the script after some animals have been selected and the submit has been pressed, so more animals can be added in the contents.

Using the remote debugger

Run the script, and select some animals in the list box. Before pressing the "Add" button, go back to the editor and change the first line of the script adding a -d parameter, for example:

```
#!/usr/local/bin/perl -d
```

Save afterwards. The -d parameter tell's perl to invoke the debugger the next time this script runs. In our case, this script will run when we press the "Add" button. Now go to menu Debug / Listen for remote debugger. This will initialize OptiPerl's debugger, and it will wait for the script to run. Press now the "Add" button. You should get a "Connected" status in the debugger log window. Upon connecting, the debugger will download the source code, and open it in the editor in a new tab, where you can trace it.

 You will notice that the downloaded copy is an exact copy of the cookie.cgi you have in the editor. So why did it have to open it again? Actually it was not opened; the debugger is now connected to perl, and it downloaded the source code. Remember that listening for a remote debugger, means that optiperl will just wait for a debugger and then use it to debug a remote script. OptiPerl does not know which script that will be, and it might not even have it's source code, if the debugger is on a different computer that yours. That is why it must download it's source code so it can display it to you. Note that using remote debugging enables you to debug a script ever by running it on your web host via an internet connection.

Using the local debugger

You can also start the debugger directly on the script. This is easier to do then the above, but does not enable you to use the submit button of the web browser. You will need to manually enter the information expected using the query editor. Open the query editor, and enable the POST and COOKIE methods. Disable all other. Select the POST method, and enter the following pairs:

```
new_animals = baboon
new_animals = gnu
action = Add
```

The above emulates selecting "baboon" and "gnu" and pressing "Add"

The cookie field contains the animals of the current session, which we can edit also. Enter something like:

```
animals = baboon&1&bear&1
```

Before starting the debugger, see what happens when you press run (make sure you remove first the -d parameter added above). You should get in the current contents: 2 baboon, 1 gnu, 1 bear. This the cookie of the session, plus the animals we added with the

"add" button.

Now try starting the debugger and tracing the script. Notice that the @new array and %zoo hash is filled correctly with the data from the query editor.

Debugging a script located on your web host

This tutorial is simular than the previous, but now we will run the script on our web host and debug it in optiperl. You will need to be connected to the internet to continue.

Let's assume your site is `www.mysite.com`. Upload cookie.cgi in it's cgi-bin folder. You should be able to run the script now using `http://www.mysite.com/cgi-bin/cookie.cgi`

To debug however in OptiPerl, upload with the following as the scripts first line:

```
#!/usr/local/bin/perl -d
```

Or just add the -d parameter. If you have setup the transfer session to automatically change the shebang line to your servers path to perl, then when uploading using "Save to remote location" or by publishing, OptiPerl will put the correct shebang and append the -d parameter you added.

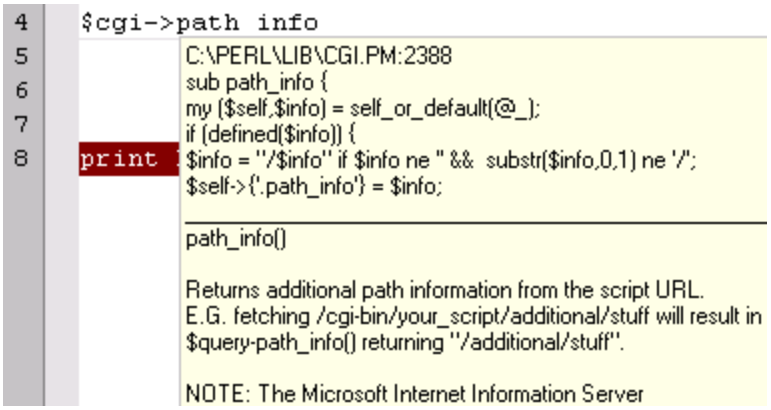
If you uploaded with the -d parameter, the script will not run on your web host. You can only debug it. Read below for more information.

Adding the -d parameter will tell perl on our remote host to invoke the debugger when running the script (in our case we will run it by accessing the URL `http://www.mysite.com/cgi-bin/cookie.cgi)`. However the problem is that perl will also need to know were in the entire internet it should connect the debugger to! We need to tell it to connect to our machine running OptiPerl.

Select menu Debug / Setup & Information. Press the tab "Setting up using remote session". Select the transfer session you used, and enter in the path the remote path you uploaded cookie.cgi, this will probably be `/cgi-bin`. Press Upload files. This sends the configuration files needed to tell the debugger on the remote web host to access our machine.

Select now menu Debug / Listen for remote debugger.

Now we have to access the script. Either select open remote script and run the remote file, or enter manually in the Browser menu the URL "`http://www.mysite.com/cgi-bin/cookie.cgi`". A connection to OptiPerl should be made and the source file downloaded so you can trace it.



Try not to trace into known good modules like CGI.pm, because this is unessesary, and also if you have a slow connection you will have to wait for them to download (if not saved in OptiPerl's cache). So when you are about to trace a line calling a subroutine of CGI.pm, like the header and cookie

subroutines, press "Step Over" instead of "Single Step". Or enter a break point somewhere in your script and then press "Continue script".

Depending on the speed of your internet connection, the responce time after pressing "trace over" or "single step" might not be instant.

IMPORTANT: If you have a dial-up internet connection with a dynamic IP address, before disconnecting from the internet, delete the configuration files sent, by selecting Debug / Setup & information / Setting up with transfer session / Delete files. Or delete the files `.perldb` and `perldb.ini` that were uploaded manually. If you don't, when you terminate your connection to the internet, and the temporary IP you had is assigned to another customer of your ISP, he/she could also create a debugging session with the scipt you were debugging. This also means ability to execute any perl function, and view/edit/delete all files on your web site.

Creating files

OptiPerl can help you make and test perl scripts that are

- Console
- CGI standalone or invoked from a form
- CGI that are server-side includes

Because perl scripts are often used with HTML, XML pages, OptiPerl also supports editing and syntax coloring for most tag-languages plus other languages.

To create a new file, select File / New / New Script for a Perl script or File / New / New HTML for a html document. For other types of files or common templates of Perl scripts, select File / New / Choose from template.

Creating and adding your own file templates

When you select the menu File / New / Choose from templates, the actual directory structure of the subfolder `Templates` in OptiPerl's installation folder is shown. You can modify the existing templates or create your own depending on your personal or team needs. You can also add more folders or subfolders here.

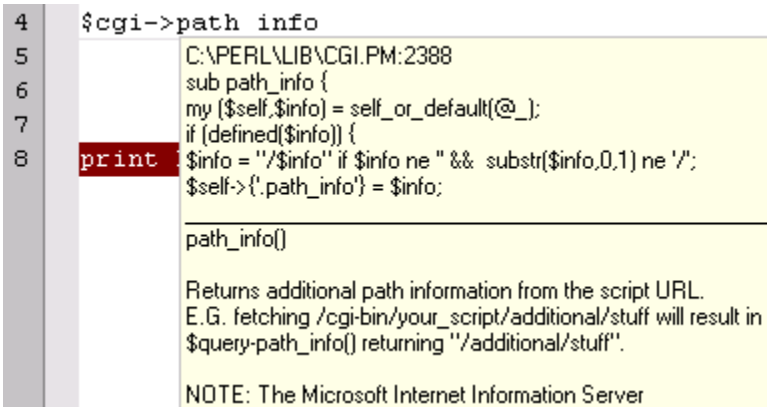
About the line endings of files

Depending on the operating system, different characters are used to mark the end of line (EOL) in text files:

- Window's use \r\n (characters 0a 0d hex)
- UNIX - Linux uses \n (character 0a hex)
- Mac uses \r (character 0d hex)

In OptiPerl you can open or create any type of the above text files. When opening a file, its EOL sequence is recognized and displayed in menu File by selecting one of "Windows", "Unix" or "Mac" format menu items. You can change the format by selecting a different item and then save the file.

When creating a file you can select a default EOL character in Options Dialog / Default.



If you are uploading via FTP and have selected "text" as transfer, then selecting any EOL character will not be important,

since the remote FTP server will change the line endings as needed. However if you are uploading via Secure FTP, then you will need to select a correct line ending sequence.

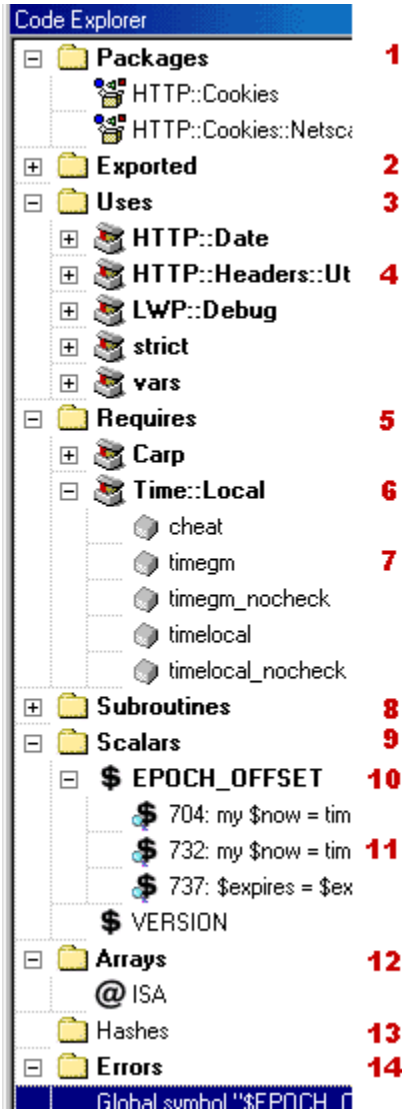
Editing files

The main editor of OptiPerl is a sophisticated and highly customizable editor. Here is a list of some of the features of the editor:

- Syntax highlighting for both perl, html and other languages. The colors may be customized in the Options Dialog. You can also have up to 5 set of colors that can be quickly selected from the Edit menu / Text styles.
- Templates that are invoked with CTRL-J.
- Bookmarks, accessed by right clicking on the editor or using the shortcuts Ctrl-Shift-0..9 to set and Ctrl-0..9 to navigate to.
- A standard Find / Replace or enhanced with regular expressions.
- Version Converter.
- Right clicking and dragging on the gutter selects entire lines.
- Commenting and indenting code.
- Holding down Control and moving the mouse cursor on a bracket character [{(<>)}] will show it's counterpart in the editor.
- Holding down Control and clicking the mouse on a declaration will search for it and open it in the editor.
- Hovering the mouse over a declaration will show a hint with information about it. If debugging at the same time, the expression will also be evaluated and the result show.
- Right clicking accesses a menu where you can control bookmarks, search a word in perl documentation, open the To-Do list. You can also edit this menu.
- Left-clicking on the gutter will add or remove a breakpoint. Read more in Debugging.
- Pressing F1 on a key word will bring up perl's core and module documentation.
- Box & Line coding.
- Code Folding.
- Code completion invoked by pressing Ctrl-Shift-Alt. It's output, whether it will be a hint window or pop-up box depends on where the cursor is when invoked.

Code Explorer

The code explorer window has a centralized view of the subroutines, variables and libraries used in your script. Also the results of a perl pattern search and any errors or warnings are viewed here. The code explorer also lists subroutines of the libraries used by the script.



1) **Packages** in script.
2) **Exported** elements from the modules used in the script.
3) **Uses**. Modules used in the script (use ...).
4) **Expandable modules used in the script**. The node can be expanded to show the subroutines of the module.
5) **Requires**. Modules required in the script (require ...).
6) **Expandable modules required in the script.** The node can be expanded to show the subroutines of the module.
7) **Expanded module listing its subroutines.**
8) **Subroutines** in the script. See also the sub list.
9) **Scalars**.
10) **Scalar node expanded.**

11) **Occurrences of a specific scalar in script.** Shows were the scalar is used in the script in any context.
12) **Arrays.** As above.
13) **Hashes.** As above.
14) **Errors and warnings node.** Displays errors or warnings as you type, and is also updated automatically. Disappears if not errors or warnings are found. See Testing for Errors.
15) **Search results**. Shows the results of a pattern search or logs the replacements made.

All nodes can be double clicked to go to their corresponding line in the editor.

The nodes under Uses and Requires (that list the modules used) also have a right click menu, so you can open them in the editor or in the Pod extractor.

The search results node also has a right click menu, to clear the search results or repeat the search.

How code explorer searches for modules

The %INC path in Options or Project Options must be correct. For more information see how OptiPerl finds modules.

What variables are listed

All global variables show up in the code explorer, either scalar, hashes or arrays. Note however that private variables in subroutines are not listed. For example:

```
$a = 1;            <- Listed
my ($b,$c);        <- Listed

sub test {
 my $d = 4;        <- Not Listed
 $e = 5;           <- Listed
}
```

Results of a pattern search

If had previously done a Pattern Search, then the results of the search will be shown in code explorer, under the node "/pattern/", where pattern is what you had searched for.

Setting the position

This window can be docked to the main editor window by dragging it. If you want to place it over the main editor without it being docked, hold down the ctrl key while dragging it. Undock it by dragging its title bar. Right click on the title bar when undocked for more options.

Renaming variables

You can rename variables using code explorer. Left click on a variable and pause until you can edit it. This is like doing a search and replace in your script.

Exporing the Code Explorer

Select the option Search / Export code explorer. This will output a tab delimited file that can be imported from a database.

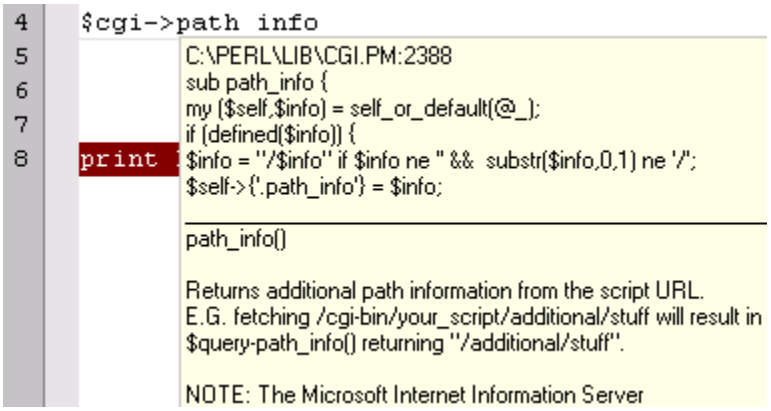
When code explorer gets updated

Anytime you edit a script, after a small delay, code explorer will parse your script. This however happens in an *idle* thread, meaning that the processing power needed will not make editing slower, since updating will pause while typing, using the menus etc. Depending on your processor and the size of the script, the updates might take place instantly while you type. On slower computers and / or larger >200kb scripts you might see it actually disable itself for a few milliseconds to update its tree structure.

Testing for Errors

Automatic syntax checking

While typing, your script can be checked automatically for errors and warnings:



Customizing of whether this feature will be enabled, and coloring of the offending lines can be done in Options / Error testing.

When moving the cursor on to an offending line, the status bar at the bottom of the editor displays the error. Also all errors and warnings can be show in the code explorer under the Error or Warnings node.

You can set up automatic syntax checking from the Options Dialog / Error Testing.

Another option ("disable on paths") is to disable automatic syntax checking when the file is located under some drives or folders. The paths can include the standard perl libraries which are known to be error-free. Example:

```
c:\perl\lib; c:\perl\site\lib; Y:\; \\computer1\C\;
```

Note also that all the subfolders will not be checked either. If you have set-up this option, and need to do a syntax check in one of the files under the paths selected, you will need to do it manually (read below).

When perl syntax checks scripts, it actually executes parts of the used modules of the script, and also BEGIN and CHECK blocks. For most cases this is OK; however if for some reason you do not want this to happen, then check the option Run / Syntax checking only in script. Note though that you might get errors complaing about barewords not being allowed.

In some rare cases, automatic syntax checking should not be called while editing. If you are editing a script like this, you can manually disable checking for the specific script by disabling the menu item "Run / Automatic syntax checking". This item is per script; it's value is saved if needed for the file if it's contained in a project.

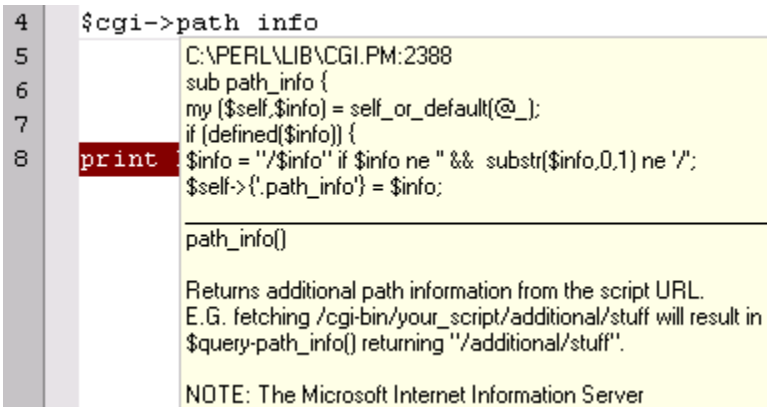
Testing for syntax errors and warnings

In OptiPerl you can test your script for compilation errors and warnings using two methods by selecting *Run / Syntax check* or *Expanded Syntax Check*. Both will check your script for

errors or warnings and report the results in a separate window.

- Syntax check outputs results identical to running perl with a -c switch.
- Expanded syntax check will also evaluate first required modules in your script. This is useful if you are using required modules that declare a variable, and in the script you get false warnings of "used only once".

The Status Window is brought up docked to the main editor. By double clicking each line, you can go to the offending line. If you used the expanded syntax check, you might also get errors and warnings in required modules, which optiperl will load first before going to the line.



In the Options under "Perl" you can select the level of warnings (None,Useful,All).

To-Do List

The to-do list is useful to make a list of future enhancements to your script. They are saved with the script, if the script is contained in a project.

To add a new to-do item, do the following:

1. Right-click on the window and select "add item"
2. Enter the action title, priority, owner, category and notes.
3. Click "OK"

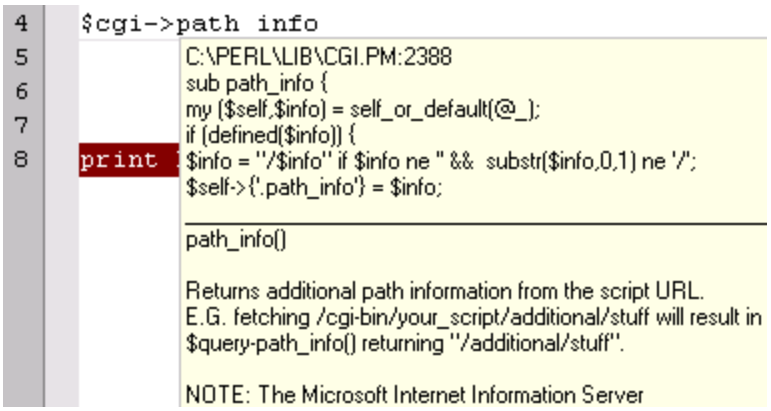
This way you can add new items. Clicking the checkbox also makes pending items done.

Clicking the title buttons, sorts the list according to the corresponding field.

Bracket Highlighting

In perl, brackets, quotes and parenthesis must be matched. Bracket highlighting helps find the matching bracket while typing. To enable this feature, select Options Dialog / Editor / Bracket Highlighting / Auto highlight brackets.

You can also highlight corresponding brackets while pressing the Ctrl button and moving the mouse over the editor. This is enabled from Options Dialog / Editor / Bracket Highlighting / Highlight brackets with mouse.



You can change the display of the highlighted character in Options Dialog / Syntax Coding using element "Bracket matching"

Editor Templates

Editor templates are invoked by pressing CTRL-J in the editor. When pressing the key sequence, a small window is brought up with common routines that are used in perl. Selecting one will insert the text in your editor. Alternatively you can enter the name of the code template in the editor, and press CTRL-J on it. This way the name will be replaced with the code, without showing the window (CTRL-J is the default shortcut).

You can also edit the templates and add your own. Select "Edit Menu / Templates...", to open the corresponding window. Here you can add, edit, sort and delete templates. To add a template, do the following:

1) Press Add Template
2) Go over the text of the newly inserted line "New Template" and press once the mouse button to edit the name of the template
3) Repeat with "Enter Description" to add a description
4) Edit the code in the text editor.
5) Optionally sort the templates, or drag the new template to the position you want.

In the text editor, enter the character "|" to show where you want the cursor positioned when you insert the template.

You can also import templates from a txt file. The templates file is saved in the Application Data Folder as `Perl Templates.txt`.

Code completion

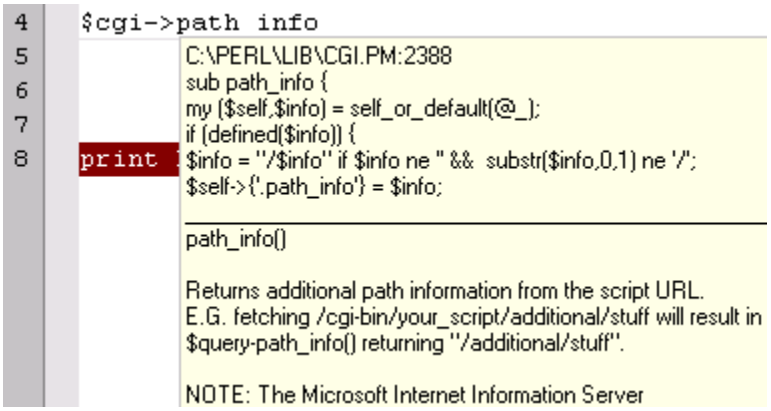
Code completion is a unique feature of OptiPerl. It helps the programmer find variables and subroutines in classes, and also if possible information about their parameters.

Code completion is activated while typing if you have enabled it, or from the Edit Menu / Code completion.

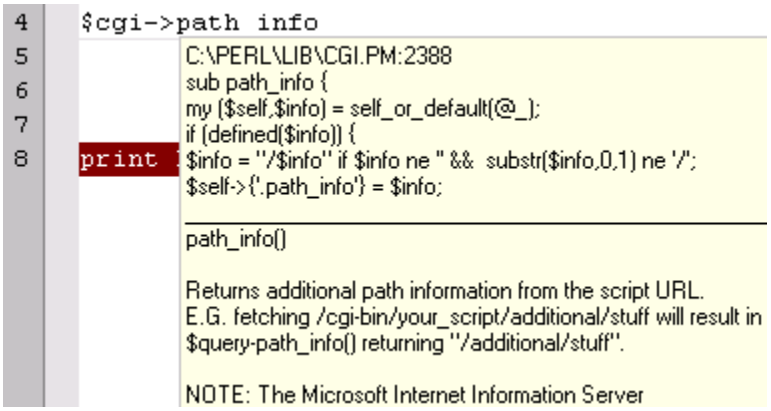
Enter the following:

```
use CGI;
$cgi = new CGI;
$cgi->
```

After pressing the last > an pop-up should appear showing methods for the class. While browsing you may notice that for some methods in the window, optiperl has also found more information, so it appends it's parameters (in parenthesis), for example `remote_host()` in the pop-up window.



Select one of the methods with extra information. The method will be added in the editor. Afterwards, if Hints are also enabled, then more information will appear:

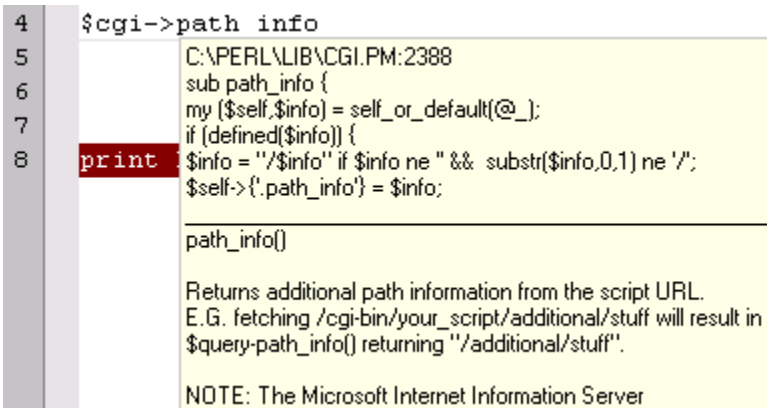


You can also do the following:

`$CGI::` *(pop-up)*
or
`&CGI::` *(pop-up)*

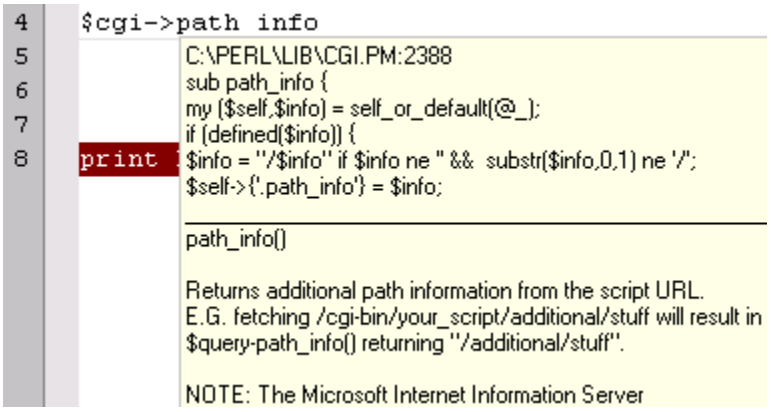
When does it work:
Code completion in the above cases will work with most libraries in the lib and site\lib path of perl. Extra information about methods are extracted if found in embedded pod documentation and if it is written in the way most CPAN module developers write it.

However it will work with your custom modules also, if OptiPerl can find them. So make sure they are in the same directory as the script using them, or the @INC path as set from OptiPerl's options or project options (for more information see how OptiPerl finds modules. Note that a module is found if it's in a **bold** font in Code Explorer's Uses or Requires node.



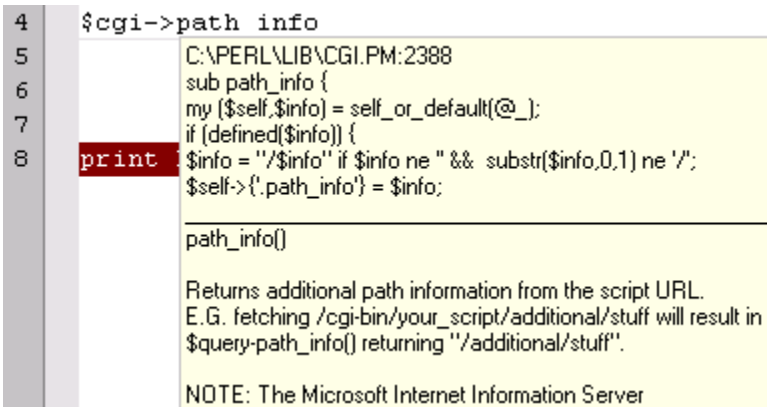
Important: Perl does not predeclare classes like other languages do. Because of this, the results of Code Completion will never be completely accurate, or sometimes may not list all methods in a class.

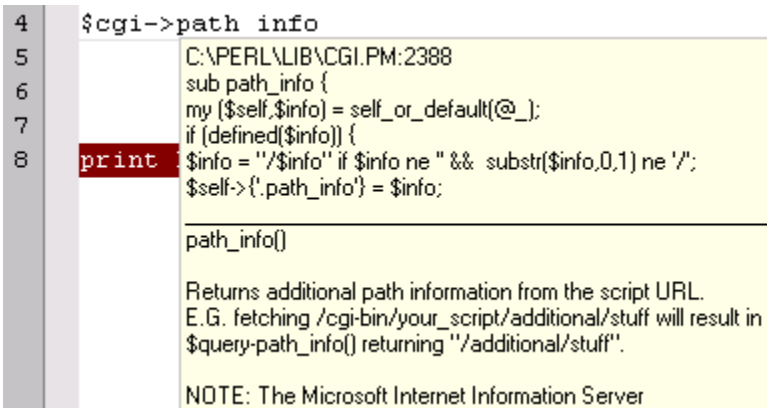
Other Uses



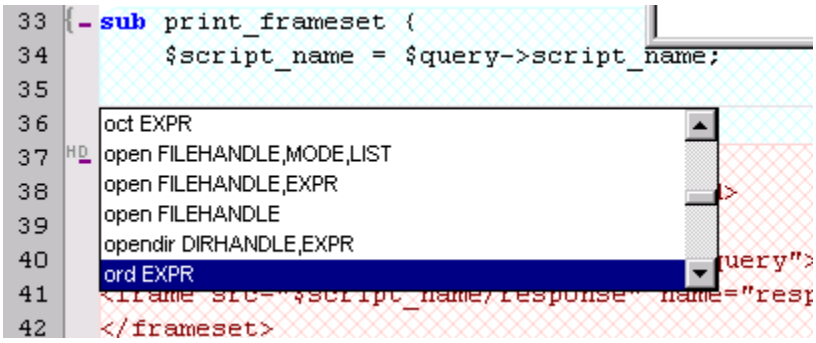
With OptiPerl's code completion you can also do the following:

`print "<` *(popup dialog)*





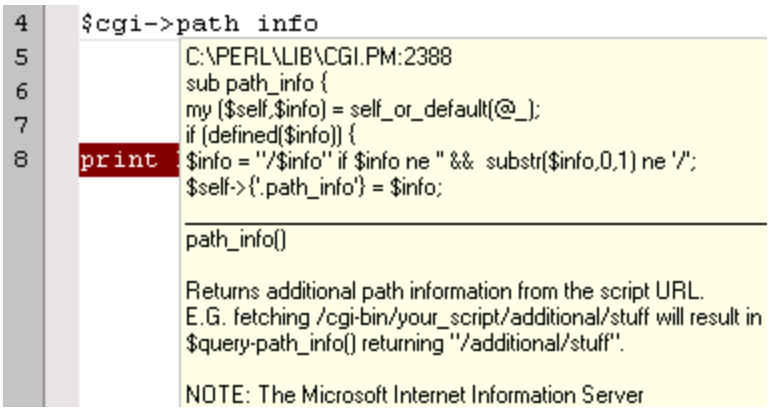
Also all of perl's standard functions are included. Try pressing Shift-Ctrl-Space (the default keyboard shortcut for code completion) while typing:



A small hint box about perl's function also pops up while typing a function or hovering with the mouse.

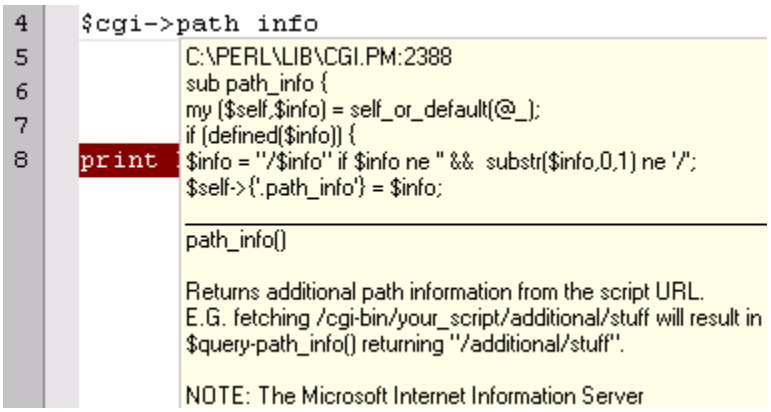
Code folding

Code folding temporarily hides sections of your script enclosed by brackets, parenthesis, Here-Document and POD statements.



While typing, the code folding symbols appear automatically. In the Options Dialog / Code folding you can set what kind of elements should appear with a code folding symbol, and whether sections of your script should be hidden when you first load it in the editor.

The code folding symbols appear in the Code folding Gutter, it's color can be set from the above page in Options.

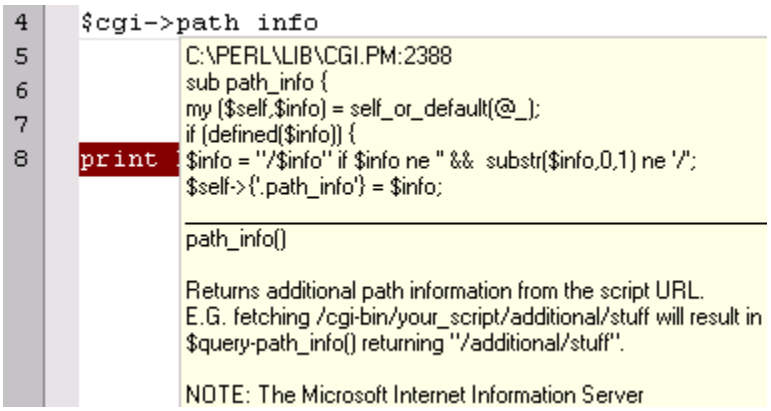


You can select whether the last line of folded blocks will be visible or not, from Options / Code folding, "Fold last line" option. If enabled then the ending line that terminates the block will also be hidden when the block is folded, for example:

```
+ sub subroutine {
```

If not enabled, then the line that terminates the block will be visible, for example:

```
+ sub subroutine {
  }
```



Code folding cannot be used at the same time with word wrap.

Syntax highlighting

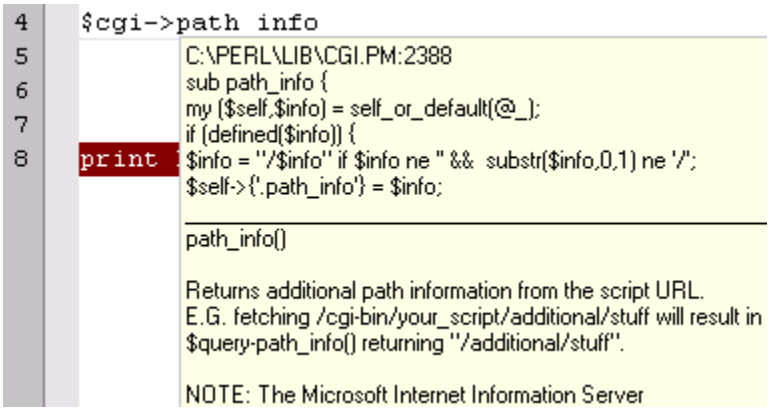
Syntax highlighting means color coding the script's elements in the editor. The following elements of programming can be colored:

- Whitespace
- String
- Comment
- Identifier
- Integer
- Float (like `2.55`)
- Reserved words (like `if, ne, use` etc.)
- Delimiters (`.{}[]()`)
- Regular Expressions (the pattern in `~, m`)
- RegExp Replacement (the replacing text in `s, tr, y`)
- Perl Declared Identifier (see below)
- Perl Internal Functions (coloring of internal functions like `print, flock` etc)
- Pod (pod text)
- Pod Tags (the tags in pod like `=head 1, =item`)
- Perl Variables (color of private variables)

About Perl syntax coding

OptiPerl's perl syntax coding is extremely fast and can color code almost any possible perl code. This includes:

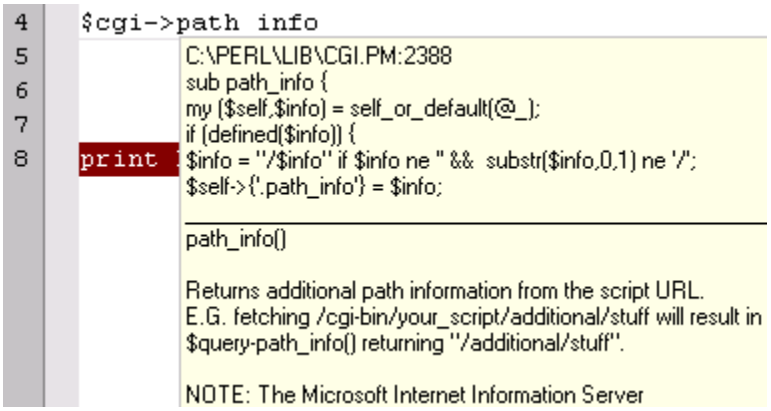
- Multiline q print statements with nested brackets. If html code is actually detected, then html color syntax is used for the string. The same also applies for here-documents.

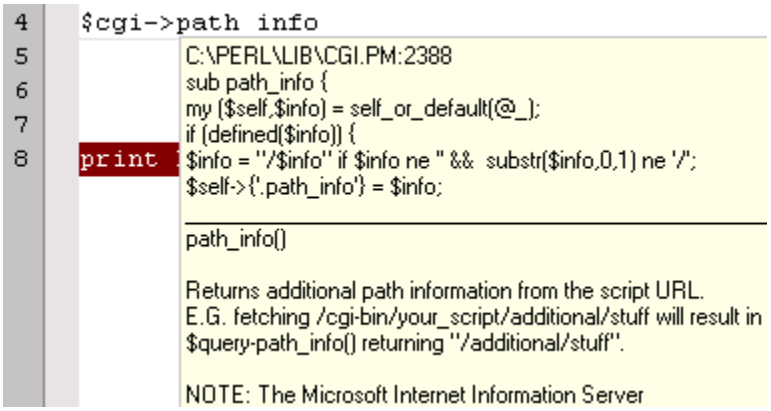


- Multiline regular expressions. The matching pattern is colored using style "Regular Expressions" and if the pattern is `s, tr, y` then the replacement part is "RegExp Replacement" style

```
m<
   (?{ $cnt = 0 })              # Initialize $cnt.
   (
     a
     (?{
         local $cnt = $cnt + 1; # Update $cnt, backtracking-sa
     })
   )*
   aaaa
   (?{ $res = $cnt })           # On success copy to non-local
                                # location.
  >x;
s/([^ ]*) *([^ ]*)/$2 $1/;  # reverse 1st two fields
tr[\200-\377]
  [\000-\177]; # delete 8th bit
```

Used identifier coloring. If enabled from Options dialog, this colors all instances of variables and subroutines, with the "Perl Declared Identifier" style **only** if the identifier has been declared in the same script. However private variables in subroutines, and variables from other modules, are colored using style "Perl Variables". Subroutines used from other modules are colored using style "Identifiers".





Use this unique feature of OptiPerl to minimize errors when calling subroutines and using variables. If while typing the code to call a pre-defined subroutine in the same script, the sub's name does not change to the "Perl Declared Identifier" style (the default is a bold font), then you know that you have a misspelling.

About the "Whitespace" element

The whitespace element has to do with the spacing between words. So the color of the text you select for this element does not affect the colors in the Main and Code Librarian editor, but affects the text color in other editors used in OptiPerl, like RegExp tester, Perl printer etc. This is important if you have selected a dark background for the editor under Options / Editor.

Syntax Coding for other languages

OptiPerl can also color syntax HTML, XML and a plethora of other languages. When you load or create a file, according to its extension OptiPerl decides on the parser it should use. You can also select one while editing from the File Menu / Other Languages. About how OptiPerl makes the decision about the correlation the extension and the default and not default parsers, see Special Files

For HTML documents with / or without scripting languages, instead of the above the following are used:

- Html tags
- Html params
- Script Whitespace
- Script Number
- Script Comment
- Script String
- Script ResWord
- Script Delimiters
- Emphasis
- System Variable
- Assembler

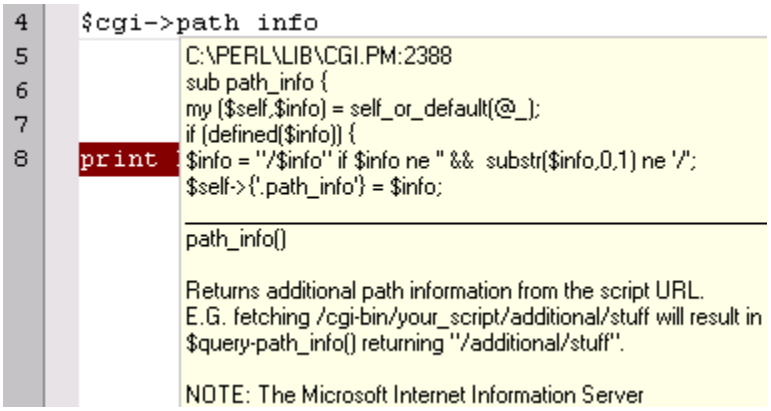
Special are the following elements:

- Breakpoint: Coloring of the breakpoint lines.
- Error line: When double clicking a line in the "Syntax Check" window.
- Debugger: Line that is about to be executed when debugging.
- Search result: When double clicking a line from the Search Results node in Code Explorer
- Bracket matching: Coloring of brackets when the Bracket Highlighting has been enabled.

Box & Line color coding

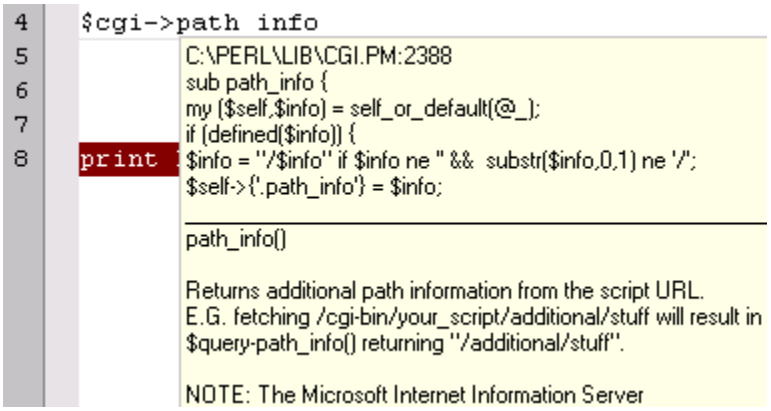
Box and Line coding is a unique feature of OptiPerl that helps visualize perl code.

Line coding



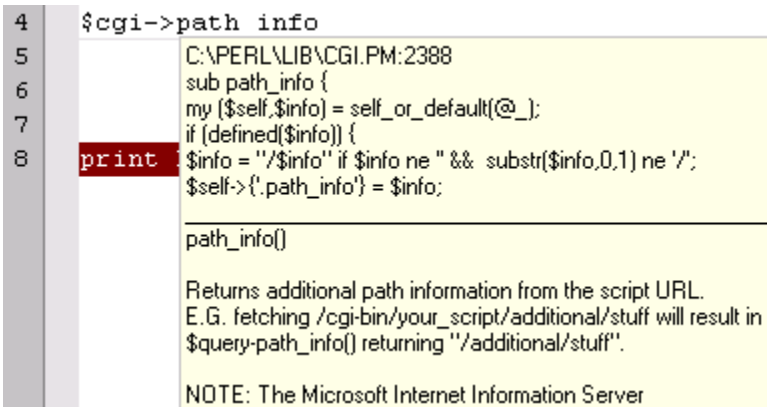
Line coding are curves that connect opening and closing brackets. Their width, style and color symbolize the level of the bracket. The default in OptiPerl is each bracket level to have width of one pixel more than the previous level. Also the first level does not have a line, but has instead a box around it (used better to color subroutines). These are defaults however and may be changed from the Options Dialog / Box & Line coding.

Box coding

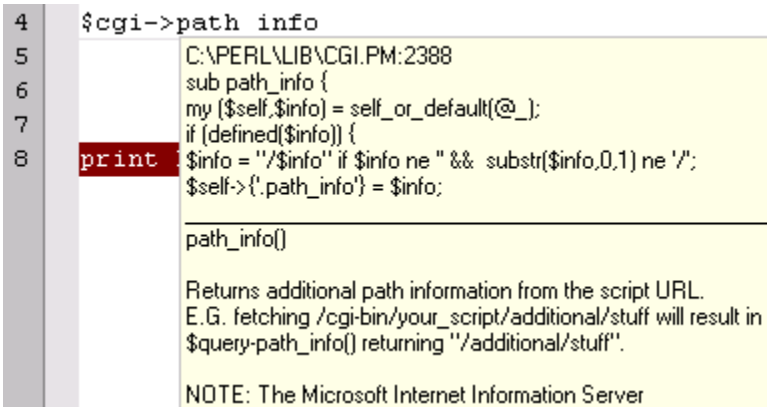


Box coding are boxes that color parts of the script. For brackets and parenthesis, each level has a narrower box by one character on both sides. Also Here-Document and Pod statements can be colored.

The box coding may also be highly customized in Options Dialog / Box & Line coding.



Notice that for boxes you can select a non-solid pattern. Non-solid patterns are see-threw, so you can see the other boxes under them. Play around with the options to fit box & line coding to your preferences.



While color coding can make perl code a work of art (literally!
), proper indentation of brackets should always be used. Remember that you might want to send someone that does not have OptiPerl your code! But if your habits are just uncorrectable, you can also beatify your code using Perl tidy.

Commenting & indenting code

About commenting blocks of code

Often while developing, you may want to comment blocks of code for testing purposes. OptiPerl provides a way to do so from the Edit menu:

- Comment in will add a # character in front of each line of the selected block.
- Comment out will remove the first # character from each line of the selected block.
- Toggle comment will add or remove the first # character of each line, depending if it's already present.

Commenting HTML code

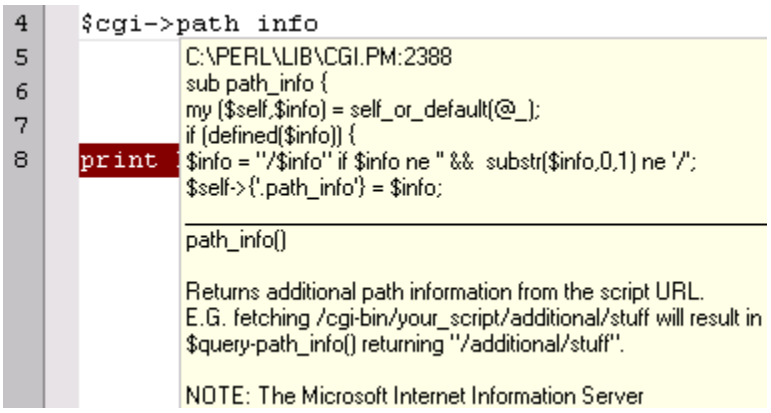
If you try to comment in or out blocks of html code found in multiline q statements or here-documents, then the html comment standard is used.

Indenting blocks of code

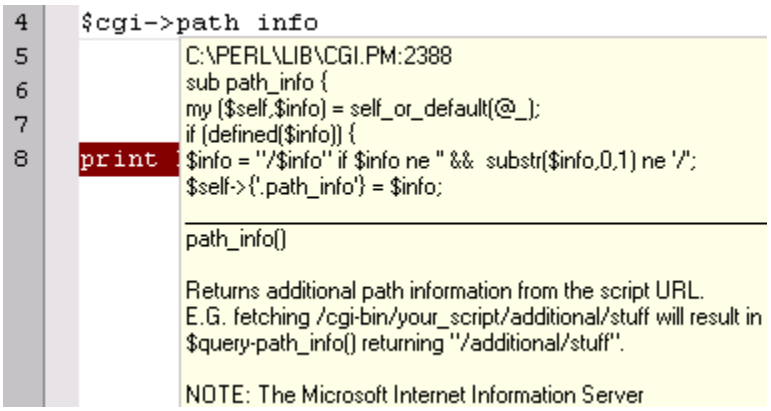
The indent / unindent commands insert or remove spaces in front of the selected block, depending on the setting of the Options dialog / Tab handling / Block indent amount. These commands are on the Edit *toolbar,* and their default shortcut is Ctrl - or +. If you use these commands a lot, you may also want to move them to the edit menu using the customize menus dialog.

How indenting fills in lines

Depending on whether "Use tab character" is enabled in Options / Tab handling, the filling will be only spaces or a sequence of tabs and some needed spaces to increase indent according to the block indent amount.



If you are using tabs in your code, and want to convert a file that was indented using spaces to tabs, select the entire file, click once "increase indent" and then once "decrease indent". This will replace all starting sequences of spaces into tabs and a few spaces to match the previous indenting.



Included with OptiPerl is a user tool, to insert a tab character in front of each line.

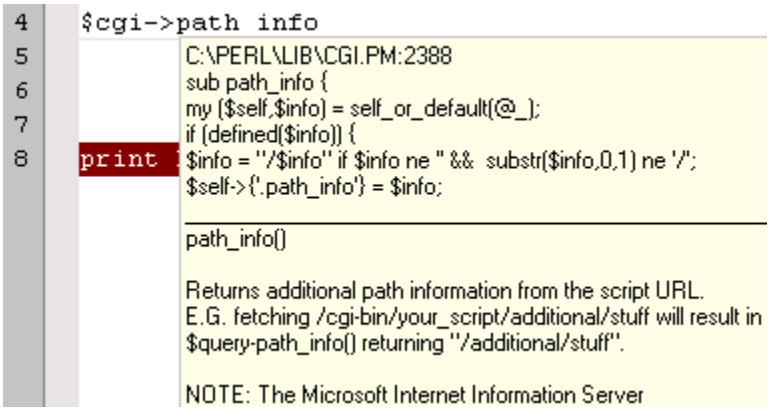
Secondary edit window

The secondary edit windows are windows that can be opened and linked to files open in the main editor. The editors will link to the same file and not copies of the same file. This means that changing text in one will automatically change the text in all windows that are linked to the same file.

If you have the option "Synchronized scrolling" on then also scrolling is linked. This can be useful when comparing two files. OptiPerl however has a File Compare tool.

Printing

You can print your perl code, previewing it first with all the syntax, box and line coding from menu File / Print preview. From this dialog you can also select a range of lines to print. In the preview window, left or right clicking on the image will zoom and unzoom.



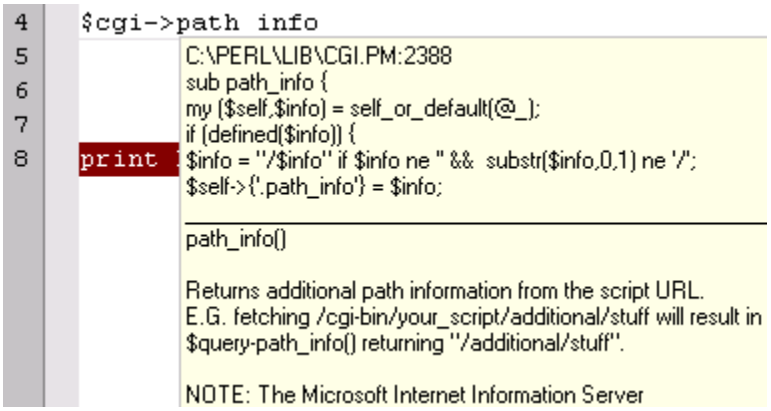
More options for printing can be found in Options dialog / Printing

Before printing a large document with box and line coding, try printing a page first. You might get unexpected results depending on:

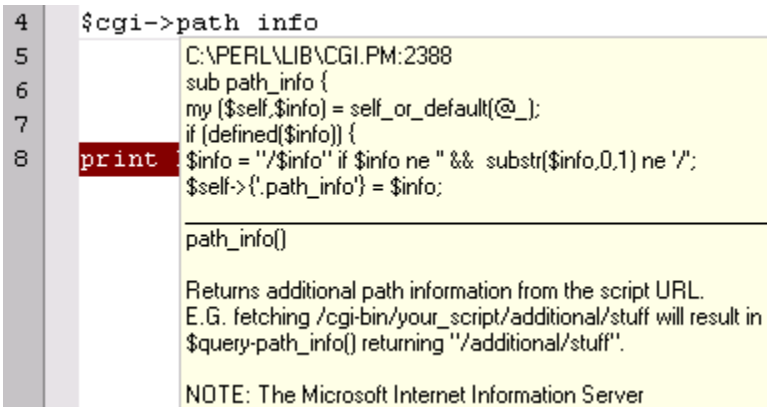
- The printers DPI
- Whether the colors will be converted to gray scale or black.

Patterns in boxes are made with a pixel width of one. On the screen this may seem fine, but on a printer with a DPI lower that 300 the scaling might make the lines disappear or create an unappealing pattern. This is why the option "*Convert all patterns to solid*" exists.

The same problem might exist with very thin line coding. You may specify an override value for all lines from "*Override line widths with x*"



Important: If you have enabled box coding (in the printer options) and some of the text in the output is not printed, then select "compability mode". Deselect it however for a small improvement in the boxes printed.



You can also export a page to a .bmp file. Select the button "Export page" from the page preview dialog. The size of the picture will depend on the current zoom of the preview.

To create a nice windows desktop, find your favorite perl code and select "export". Scale if necessary with imaging software.

Running CGI Scripts

Optiperl is optimized to develop perl scripts for CGI use (although many tools exist solely for console scripts).

CGI scripts are programs that are used for web sites. Perl is more often used to create CGI scripts, and OptiPerl is made to develop them easier.

All the rules that apply to create a perl script apply for cgi scripts in perl. CGI scripts however most follow some extra rules, to work on web servers. These rules are not complicated and OptiPerl helps you know that you have followed all rules. In some cases it even tells you what rules you have broken.

For more information on creating CGI scripts, see the Tutorial section.

You can run CGI scripts the following ways, by having the script open in the editor:

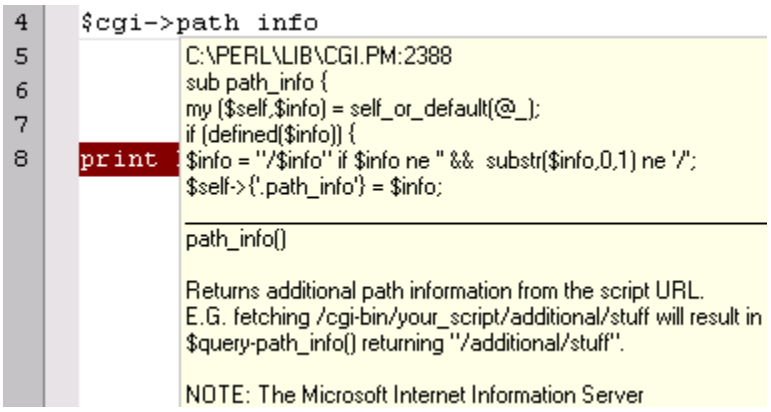
- In **OptiPerl's browser**, by selecting Run / Run in browser.
- In an **external browser**, from Run / Run in external browser.
- In a **secondary browser**, if you have setup one in Options / Environment from Run / Run in secondary browser.

In many circumstances, CGI scripts are not run directly, but indirectly from a html page. Usually pressing a submit button runs the script, and the next page loaded in the browser is the script's output.

For these cases you need to have enabled the option "Run with server". Read more about Run with server for how this is done.

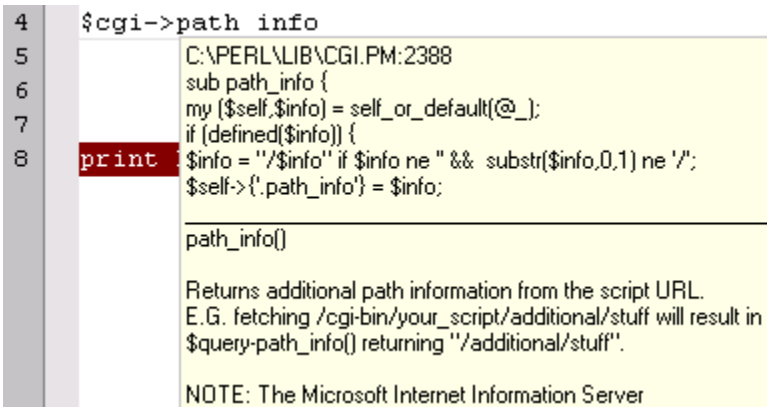
Usually you open the html page in optiperl's editor and run with one of the methods listed above. After the page has loaded, you press it's submit button (or whatever invokes the script) and view the result of the script you are testing.

If you are getting error messages, unexpected result etc., you must read about the importance of the "Run with server" option first.



This section is about running local CGI scripts.

- For console scripts read the next section
- For running scripts on remote servers, see the section "Running remote files".

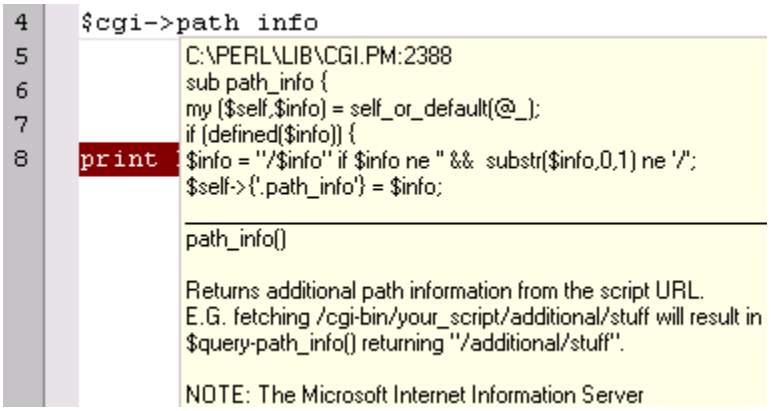


Sometimes you might want to run a CGI script that is supposed to get run from an html page, directly, emulating the input from the html page. Read more about the "Query editor".

Running console scripts

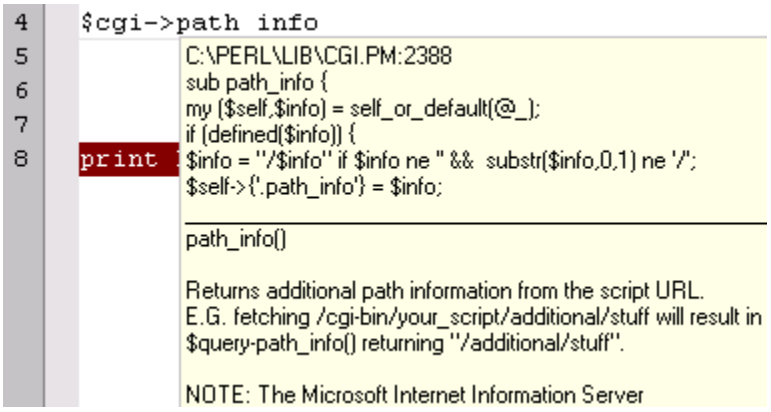
OptiPerl has massive amount of tools to create CGI scripts; however many tools exists solely for console script.

- **Arguments** (command line parameters) can be specified from Run / Arguments. Also is a combo box with previous arguments you had used. Note the argument can be added to the combo box by selecting menu Query / Add values. If you are using OptiPerl only for console work, we recommend customizing the menu, moving this item to the run menu and removing the "query" menu and toolbar.



The array @ARGV contains the command-line arguments intended for the script. Command line parameters cannot be used for CGI scripts.

- **Setting the Starting Path** from Run / Select starting path.



The starting path affects only console scripts.

- **Debugging**. The arguments are fed to the script when starting the debugger.

- **Opening / Saving** to remote locations can save you from a lot of work if the script is intended to run on another machine. Also if it is actually possible to run the script on your machine for developing purposes, don't forget about the version converter that is used automatically.

If the console script cannot be run in windows, then create a custom tool to invoke it some way for testing purposes.

- **Running**. You can run in a dos - console box. When selecting menu Run / Run in console, you first get a parameter dialog that prompts you for extra parameters to run with.

The default parameter (and the simplest and necessary) is

```
%pathsn% %ARGV%
```

%pathsn% gets replaced with the scripts path as a short name
%ARGV% gets replaced with what you entered in the Arguments.
The path to perl is appended automatically in the front.

So if you have a script named c:\perl\test.pl and you have entered the arguments "-a -b" then the above would run perl in a console window like this:

perl c:\perl\test.pl -a -b

You don't have to put %pathsn% at all. You can replace with any parameter you want, even create one-liner programs extremely quickly. For example try

-e "print 'Hello';"

If you are running in the console and the output is large, you might need to pause on each page. To do this, after pressing "Run in Console", enter in the dialog box:

```
%pathsn% %ARGV% | more
```

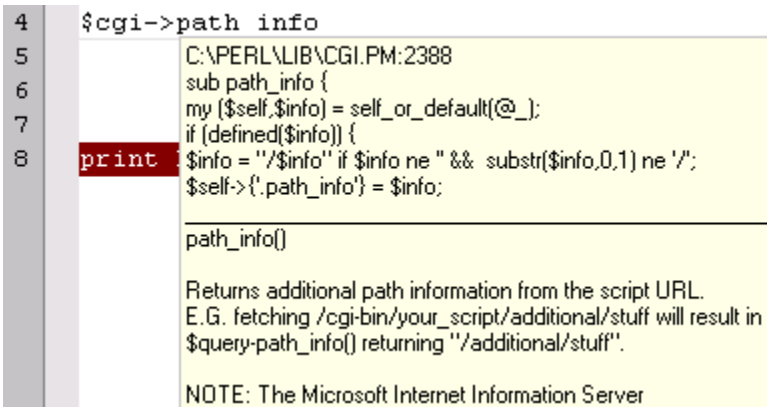
This way after each page of output you will be prompted to press a key for the next page.

- **Running in the browser.** If you are only interested only in the text output, then select Run in Browser, and have the option Server / Run with Server disabled. You will be able to see the output in the "Text" tab of the web browser. Disregard all other tabs. If the script also expects input from <STDIN>, you can enter the text you want in the "Send" edit box of the webbrowser. This will send the text with a CRLF at the end. The edit box also accepts \n \t \r and the special \z which is CTRL-Z (ascii 26) usually used to terminate input.

- **Feeding to <STDIN> automatically.** This can only be done when running in the browser. Open the query editor, and enable the "POST" method. In the POST tab, select "Raw" encoding and enter the string that should be fed in the "Manual" edit box. Again \n \t \r and the special \z are accepted, plus \f<filename> to send an entire file (right click for an open file dialog). For example you can enter:

```
\f<c:\input.txt>\z
```

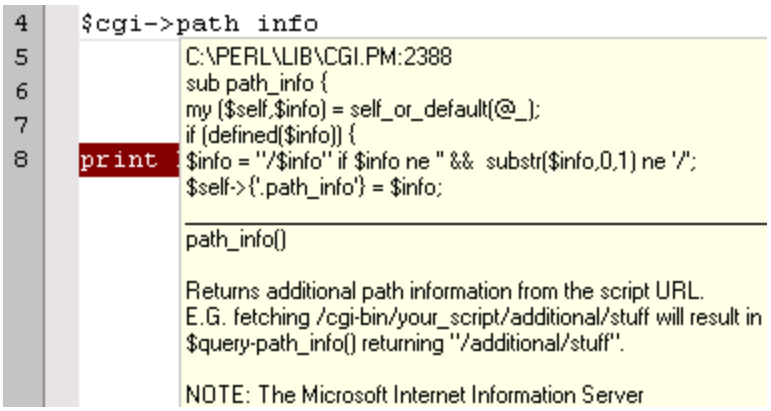
This will send the file c:\input.txt and the character CTRL-Z to end it.
Note that the POST method is used of course for CGI, but by using Raw "encoding" it can really help automation of testing console scripts. The data is fed also if you start the debugger.



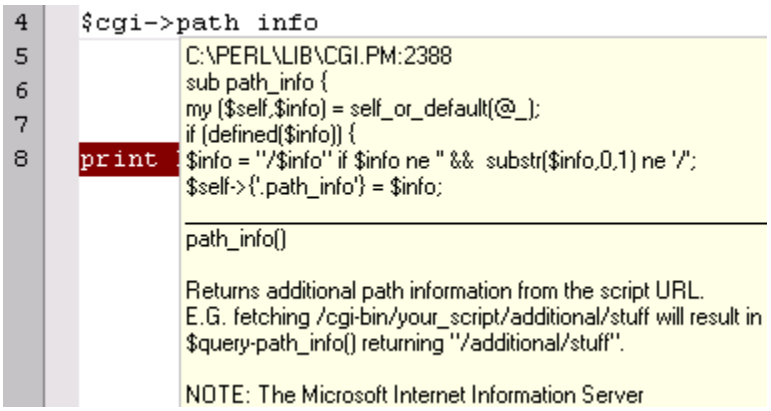
Use the "Run in browser" feature! Don't get mixed up with the word "Browser" which reminds of CGI. The text tab of the Browser window will be an exact representation of the output (a pipe is opened for this). Also the fact that you can automate sending many lines to <STDIN> makes it a big help (you cannot do this in a console window). If for example the script makes 4 questions before output, enter something like this in the "POST" method:

```
Y\nY\n\1\nN\n
```

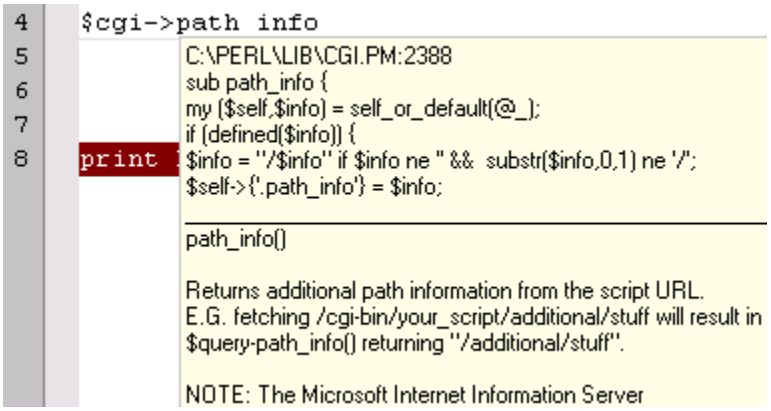
Don't forget that the POST method for CGI scripts works the same way! Only difference is that the data gets URL encoded first and headers are sent telling the script how much data to expect.



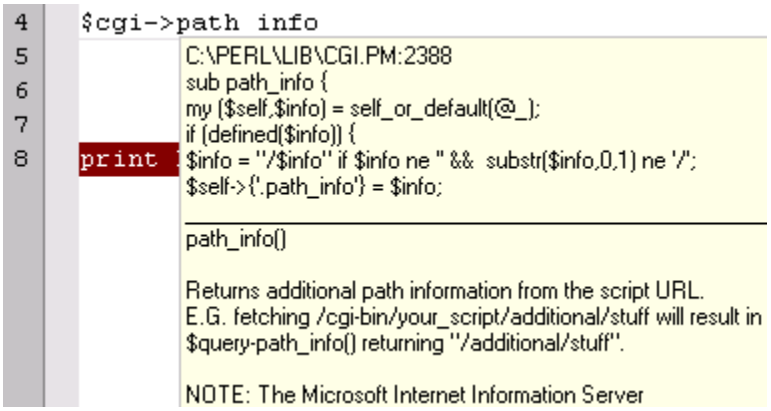
If you want to test the script with many different parameters, it might be difficult to change each time the Arguments from the Run menu. Instead, enter them each time in the Parameter list removing %ARGV%. Or do a little of both, put some standard arguments in Run / Arguments and play with the rest in the parameter dialog each time you run.



To increase productivity, move the Arguments combo box from the Run menu to a toolbar you create next to the editor.



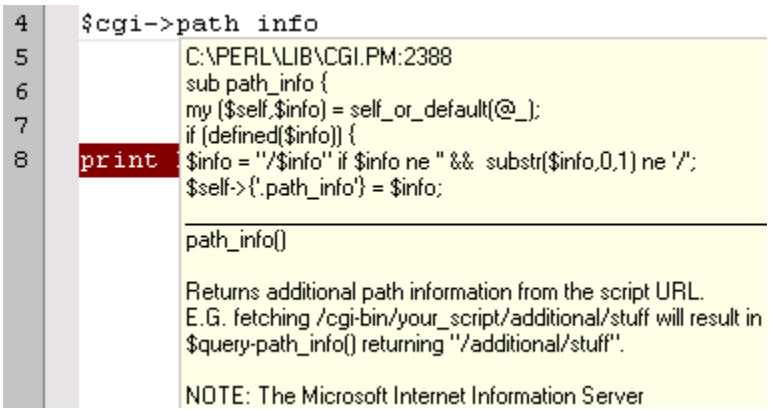
Another interesting thing to do, for the more advanced programmer. Notice that you can also use the `%f<filename>%` tag in the parameters, like for example `%f<c:\test\testparams.txt>%`. When you run, the text of the file testparams.txt will be placed as command line arguments! Make sure that the text has not EOL characters.



If your scripts also depend on their location on the drive, because they use absolute paths, see "Using absolute paths".

Running remote files

OptiPerl does everything possible to emulate a real production environment for running scripts. However it can not emulate a real production server! You might need sometimes, after uploading to a server, to do additional fine-tuning, debugging and running on the production server it self. OptiPerl helps here also; you can open a remote file via one of the remote transfer sessions, edit it, upload it automatically and run in on the server.



See also some common reasons why you might get 500 errors after uploading.

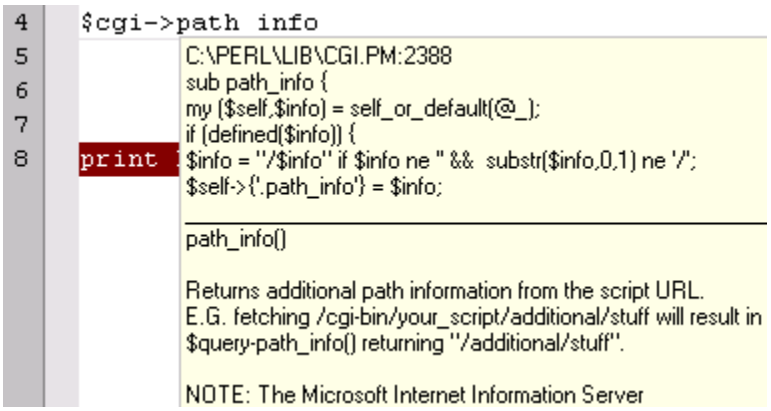
When running, the actual request that it's users would use is sent. If the script gets accessed by http://www.mysite.com/cgi-bin/test.cgi, that's what optiperl will use, by sending the request in it's browser so you can view the result.

However all of OptiPerl's tools are at your disposal for sending the request. You can fine tune the query from the query editor if you want, so you can discover potential holes. Or enable the Spy proxy to view the exact request and response from the production server.

To edit remote files, you must setup an remote transfer session so you can open the files. To run them by sending requests to the server, you must also fill the fields of the remote session dialog that help link absolute paths of the FTP or SFTP server to HTTP requests. Read about possible scenarios to do this.

To open/edit/run/save a remote file do the following:

1) Select menu File / Open remote file
2) Select a session, press the connect button and select the remote file
3) It will be downloaded and opened in OptiPerl
4) Edit it. To run it, have "Servers / Run with server" enabled, and press Run. If the file had been changed, it will be uploaded automatically.
5) To debug it, add a -d parameter in the shebang, select "Debug / Listen for remote debugger" and press run. Make sure the PERLDB_OPTS is set in the remote machine's environment. Read more about remote debugging.
6) To save it, select menu File / Save remote file.
7) To save it on a different remote session, select menu File / Save remote file as. In the dialog select the session and a filename if needed and press save.



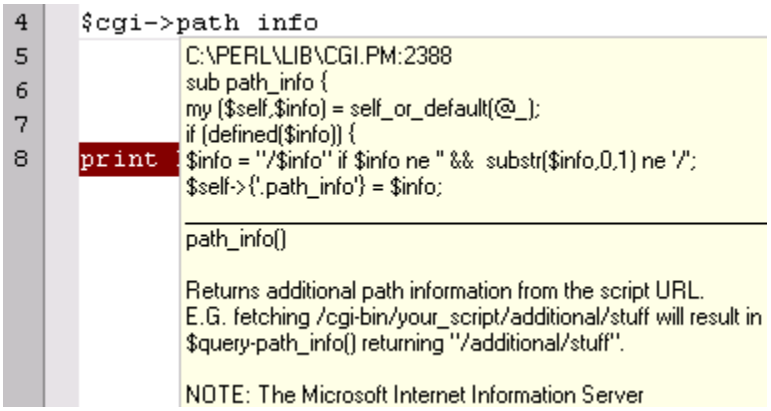
If enabled for the transfer session (change shebang and convert version) then the shebang will be changed when uploading a script to what the remote server needs. Also when uploading, the version can be converted to SERVER and when downloading to LOCAL.

Query Editor

The query editor a versatile window that allows to create the input for testing CGI script locally, or ever sending the information to a remote URL address.

If the CGI script is used to process data from a web page, the best is to load the html page and run with the internal server and run with server and use remote debugging. However in some occasions you might need to create your own query.

- For testing many values quickly.
- To discover potential holes. Many security breaks occur by hackers sending to scripts "naughty" queries.
- The query editor is also used for console scripts, to automate sending input to them (using the POST raw encoding).



CGI Scripts can accept different kinds of information. Read about the various methods in the next section.

What the query editor affects

What you see in the preview tab is sent to scripts when:

- Running without "run with server" selected
- Starting the debugger
- Running with "run with server" selected
- Navigating to any URL when selecting Browser / Go to URL with query

Editing the GET, POST, COOKIE and PATHINFO methods

For each method go to the corresponding page and fill in the dual column grid with name & value pairs. These pairs construct queries like:

```
fname=George&email=george%40mail.com
```

You can also edit the query manually in the manual edit box.

In both boxes you can also add the metacharacters \n \r \t.

Selecting a file for a multipart POST

In the POST tab, either in the value column or in the manual edit box, press right click and

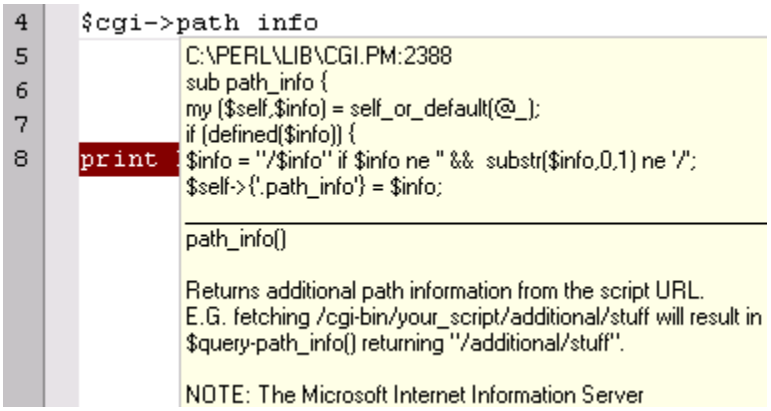
select a file to be sent.

Enabling combinations of methods

From the Methods menu, check and uncheck the methods you want to be used. If for example you want to use the POST method, make sure the item "Enable POST" is checked.

Viewing previously sent queries and saving them

You can save a query so you can access it later - do this by selecting Methods / Add values. The query will be added in the combo box of the Query menu in the main menu. To delete a previous query, select it from the combo box and press "Delete values".



All the information in the Query editor is saved when you close the script, in the active project, as long as the script is contained in the project.

Importing a form from a web page

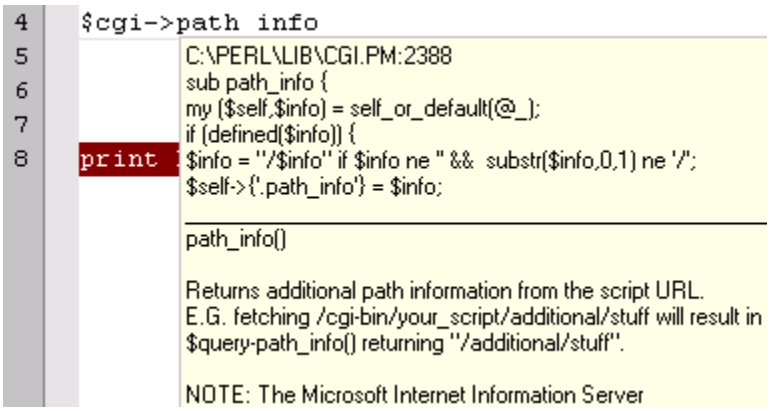
Press the "import file" button and select the html page that has the form data. Or if the html file is being viewed in the internal browser, select "import web". The html code of the form will get parsed into the query editor.

Setting environment variables

If needed, you can fine tune common environment variables servers send to the scripts they run, from the "Environment" tab.

Previewing the result

From the preview tab you can preview what will be sent when running or debugging.



If you develop console scripts, read about how you can automate sending to <STDIN> in Running console scripts.

Information sent to CGI scripts

If you are programming a CGI script that receives any kind of input, you will need to feed it with the input when running or debugging. Here is the kind of input a CGI script receives, usually from the html page that called it:

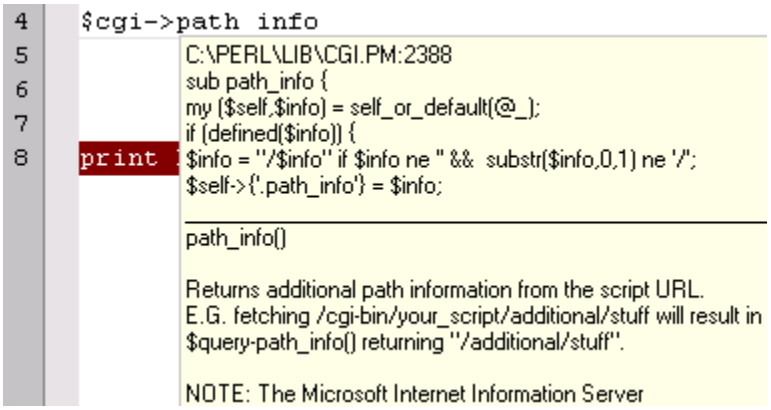
1) **GET Method**

The most commonly used method. When you see a URL like `http://www.site.com/query.cgi?somedata` the part after the ? is the get method. This information is received using something like

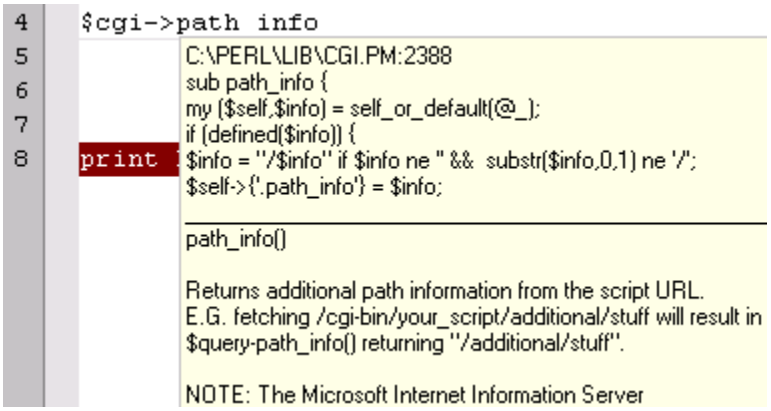
```
$query = $ENV{'QUERY_STRING'}
```

2) **POST Method**

The post method involves the client sending data after its request to the CGI script. The data is not shown like the GET method. The CGI receives the data using <STDIN>.



See "creating an html page with a form" in the tutorial section to see how to use the GET and POST method.



The data received from get and post especially when originated from a web page usually has to be parsed and URL decoded. You can do this manually, but it is not recommended. Use the CGI.pm module; you don't even have to check if it came from a GET or POST method.

```
#!perl
use CGI qw(:standard);
my $text = param('text');
```

There are two methods for sending POST data:

- **x-www-form-urlencoded**

Used most often. The data looks like a GET method:

```
email=george@site.com&text=hello
```

- **multipart/form-data**

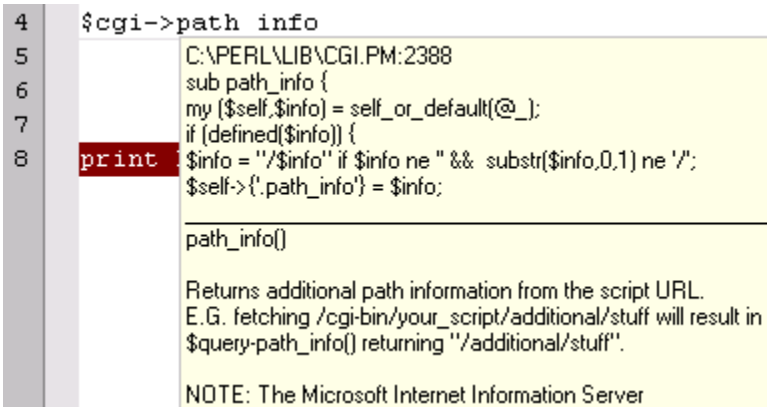
Used mostly for uploading files. Looks something like:

```
-----------------------------7d23e316405c4
Content-Disposition: form-data; name="email"

george@site.com
-----------------------------7d23e316405c4
Content-Disposition: form-data; name="file"; filename="C:\Program
Files\OptiPerl\webroot\cgi-bin\hello.cgi"
Content-Type: application/octet-stream

print "Content-type: text/html\n\n";
print "Welcome to OptiPerl!";

-----------------------------7d23e316405c4--
```



Post data is invisible to the user. However you can see it in the Server talk tab.

3) **Path Info.** Looks like the GET method.

http://www.site.com/cgi-bin/test.cgi/pathinfo

It is read by CGI scripts using $ENV{'PATH_INFO'} or CGI.pm $path_info

4) **Cookie.** When you are a member of a site, revisit it again after a month and see something like "Welcome back Mary!" you are seeing a cookie in action. People complain that you are invading their privacy; but if you want a really interactive site, you need cookies.

We will explain cookies by giving an example of a site that uses them to limit some pages

only to members.

i) You go to a site that you have never visited before and become a member. After entering you personal data, you go to a page created by a CGI script that says "thank you for joining us". Now if you were watching the "server talk" tab of optiperl you would see in the pages response:

```
HTTP/1.0 200 OK
Set-Cookie: name=mary; session=902352; expires=Sun, 21-Jul-2002 16:31:37 GMT
```

The script used CGI.pm with the following:

```
$the_cookie = cookie(
  -name    => $name,
  -session => $session
  -expires => '+1h'
);
print header (-cookie => $the_cookie);
```

Now your browser takes the "Set-cookie" field and records it on your computer.
The query editor of optiperl can show you all the cookies you have gotten in the past while browsing the internet. See its Cookie tab, cookie folder.

ii) Now if you are using the same browser and go back to the same site, your browser will automatically send to the site the cookie it had recorded:

```
GET /cgi-bin/login.cgi HTTP/1.1
Cookie: name=mary; session=902352;
```

Login.cgi gets the cookie using `$ENV{'HTTP_COOKIE'}` or CGI.pm:

```
$name =  cookie('name');
$session = cookie('session');
```

If the $name corresponds to $session then its output is something like:

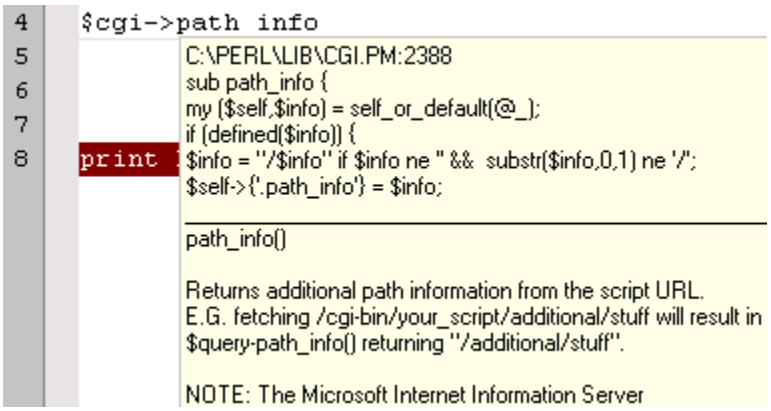
"Welcome back mary" etc.

If there is no cookie or it is cannot make any sense with it then it outputs something like:

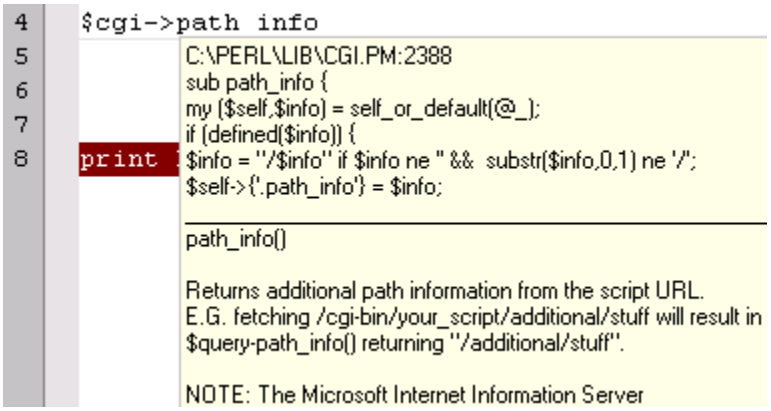
"Welcome to our site. Become a free member using the following form:" etc.

iii) All pages of the site that can be viewed by valid members check for the cookie.

iv) Note that the session here is not the password, for security reasons. It might be a generated number from the user's name and password, just to verify that the returning user is the correct one. If it is not, the password will be asked for again, and a new session number is generated repeating step I.



See the included example cookie.cgi. Make sure you run with both options "Run with server" and "internal server enabled" in the "Server" menu.



If you are really interested study how well know sites on the internet use cookies to create sessions with the user, by enabling "Spy HTTP proxy" in the Browse menu and browsing using OptiPerl's browser, and watching "server talk". A good idea would also be before, to go to menu Browser / Internet Options / Advanced tab / Multimedia section and disable "Show pictures". This is because most sites also have banners with advertisements, which usually also set cookies. This way you can watch only the part you need in the "server talk".

5) **Environment**

Before a server runs a CGI script, it set's up its environment. This includes the $ENV{'...'} values as above plus many other, most of which were read from the request of the client. When you browse a page, depending on your browser, the browser will send the server something like:

```
GET /cgi-bin/test.cgi HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
application/vnd.ms-excel, application/msword, */*
Accept-Language: en
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, */*
Accept-Encoding: gzip, deflate
Accept-Language: en
Connection: Keep-Alive
Host: www.site.com
User-Agent: Mozilla/4.0
Accept-Encoding: gzip, deflate
```

The server parses these fields, and if it needs to run a CGI script for the output, will set up its environment before running with corresponding values. This is how you can get for example the browser the user is using, with $ENV{'USER-AGENT'}. See the Query editor for what these may include.

Usually you don't need to get into this detail, however OptiPerl's query editor can give you full control if you need it.

Web browser

The Web Browser in OptiPerl is the window that you can view all the output of perl scripts and html documents.
In conjunction with the Browser toolbar, it is a basic web browser. However it has an important feature just for debugging:
At any time, all html output can be viewed either as a web document, either as text. Also there is a "send" line, for sending input to <STDIN> when running or debugging console scripts.

Read more in "Debugging Scripts" and "Running Scripts"

When running a script in the browser, there is a major difference on how the script is run, depending if "Run with Server" is selected:

- If "Run with Server" is disabled, then the script is run directly and all it's output is piped to the browser.
- If "Run with Server" is enabled, then the script is launched by sending a target URL to the browser, for example

http://127.0.0.1/cgi-bin/test.cgi

Also from the Web Browser window, under the "Auto-View" tab, you can view at real-time the changes that happen to a file you select.

OptiPerl has the only internet browser with a find dialog that supports regular expressions.

"Run with Server" Option

Most simple CGI scripts can be run directly from OptiPerl, by loading perl with the script, setting it's environment variables, and feeding it if needed with data from the POST method. This is what OptiPerl does when the option "Servers / Run with server" is disabled. Because environment and the POST data is emulated so well just like a real server, the script will be able to run in most cases. OptiPerl then grabs all it's output and sends it to the web browser. A small exception is when running in an external or secondary browser; OptiPerl takes all the output, saves it to a temporary file and loads the external browser with a file:// url to load the temporary file.

However there are some problems with this method:

- You can not really test CGI's that are used to process data from an HTML page, unless you enter the data it expects in the query editor.
- You can not run CGI's that do not produce text output.
- Generally, you cannot really test a complete web site, only parts of it. If your website is an interactive site, creates sessions with cookies, has log-ins, forums, polls, banners etc. you can't really test it.

The correct solution for testing: Running with a server

When enabling the option Server / Run with server, the following changes occur when running files:

- A real http request is sent to the server (that is running offline at that time), like for example:

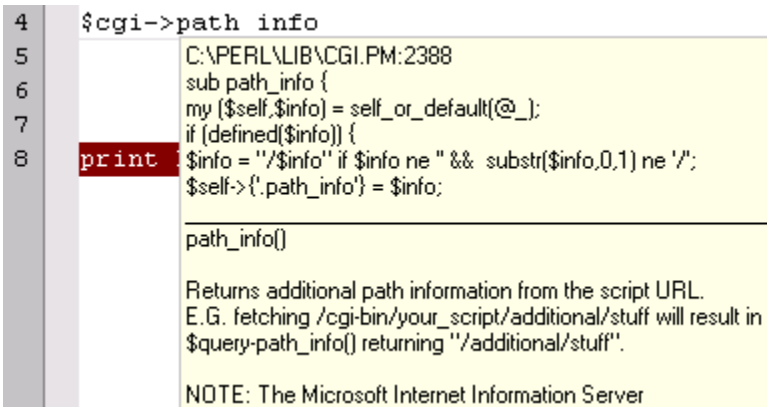
`http://127.0.0.1/cgi-bin/test.cgi`. Before, there was no request from the browser; optiperl just feeded it with output of the script.

- After making the request, just like the real world, the browser waits for the output from the server.

- The "submit" buttons, cookies, all work. If you press the submit button on an html form, the browser takes the data of the form, encodes it and requests the "Action" of the form (usually a cgi-script).

- Invoking the debugger after the submit button has been pressed also works. The browser will keep waiting patiently for the data while debugging.

- Because we are emulating the real world, if you have your entire site offline, you can call `http://127.0.0.1/` and test all html pages with the scripts they use.



But for the option "Run with Server" to work, you must also load a server! This can be OptiPerl's internal server or an external server. If a server is not loaded, then the browser will return a "page cannot be displayed" error just like when navigating to a non-existent URL when browsing the internet. If a server is not loaded, the URL http://127.0.0.1/ does not exist.

What is 127.0.0.1 or localhost?

This is a special IP that on all computers means "loopback". When accessing the special http://127.0.0.1/ or http://localhost/, your computer does a loopback and expects a response from itself. If you happen to have a web server loaded on the same computer, the server will be invoked from the request and it will send the response expected.

On windows NT-2000-XP, you can also use http://<name of computer>/. The name of the computer can be found on the Network identification page of the system properties.

If you are also connected on a LAN, then the IP assigned to the machine will also work.

Internal server introduction

Setting up a web server can be complicated, so OptiPerl provides an internal server. It is not optimized for speed, like most production servers, but for testing purposes.

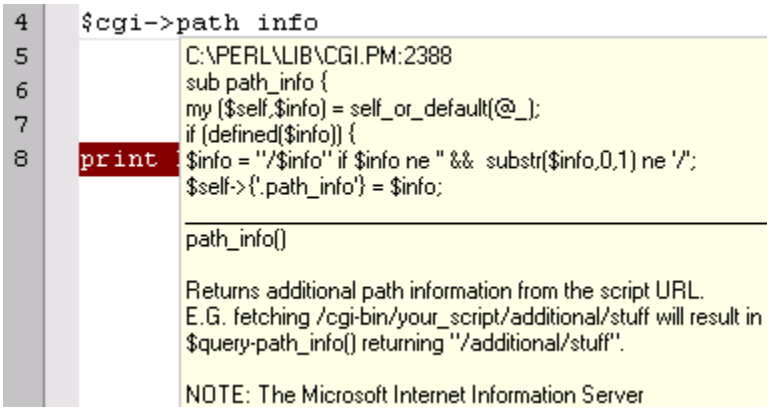
It also does not need any set-up. The only thing it must know is the folder where the scripts you have are located.

To use the internal server do the following:

1) Enable menu Server / Internal server
2) Set it's Server root to the path were all your scripts are located.

To run all the included examples with optiperl, you must select menu Server / Change server root and select folder `c:\program files\optiperl\webroot`. (or accordingly were you installed optiperl).

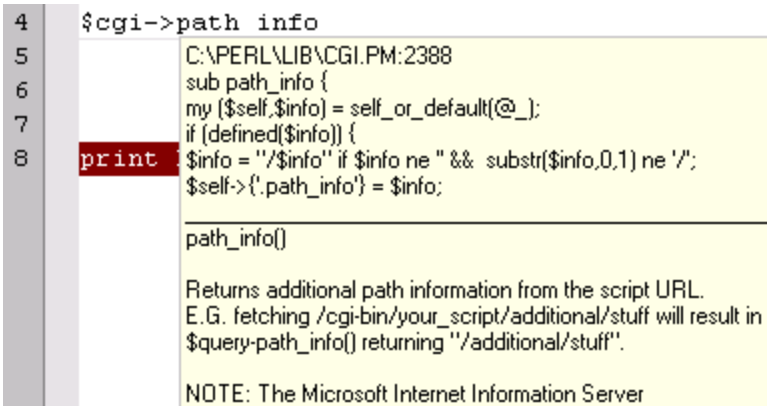
Open now a script from the cgi-bin folder above. Make sure the option "Run with server" is enabled, and press "Run".



If you don't see html output of the scripts, please read the "Possible errors" in the next section.

Notice the url sent to the browser; It is `http://127.0.0.1/cgi-bin/script.cgi` (where script.cgi the filename you opened).

Also try opening the test.html from `c:\program files\optiperl\webroot`. If you run it, the url called is http://127.0.0.1/test.html



To conclude: When we set up the

webroot to

```
c:\program files\optiperl\webroot
```

- the file
`c:\program files\optiperl\webroot\test.htm`
is called as
`http://127.0.0.1/test.htm`

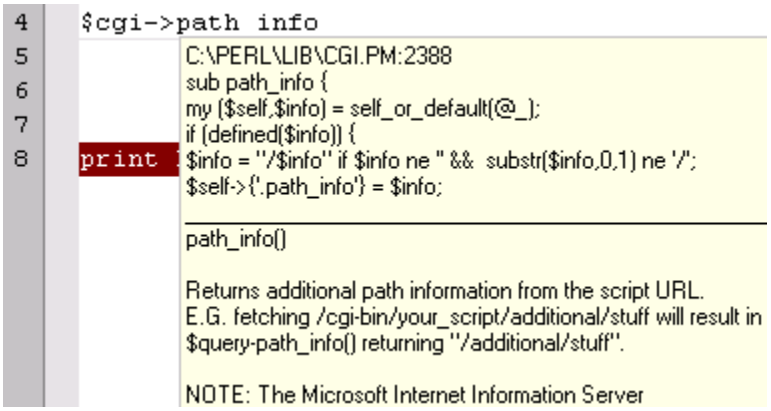
- the file
`c:\program files\optiperl\webroot\cgi-bin\test-form.cgi`
is called as
`http://127.0.0.1/cgi-bin/test-form.cgi`

- If you try to open a file that is not under the webroot, you will get an error if you try to run it.

Also notice the form part of test.htm:

```
<form method="POST" action="/cgi-bin/test-form.cgi" name="PostForm">
```

The action part tells the browser what script to call to process the form. Since the form was called with `http://127.0.0.1/test.htm`, the browser adds the host part to the action above when pressing the submit button, and calls: `http://127.0.0.1/cgi-bin/test-form.cgi`



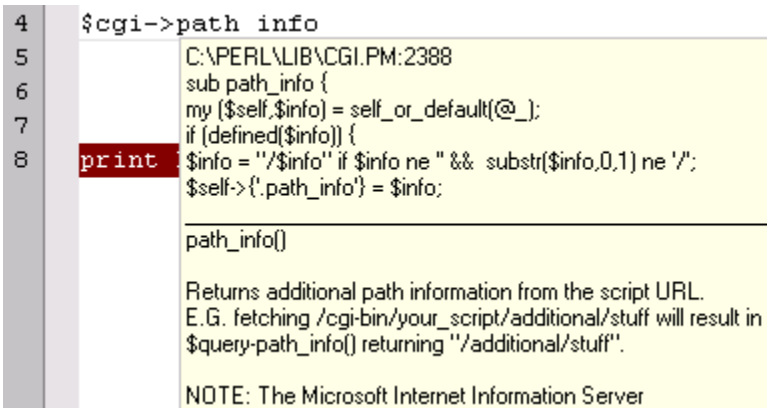
If we change the webroot to

```
c:\mywebs\mysite
```

- the file
`c:\mywebs\mysite\test.html`
is called as
`http://127.0.0.1/test.html`

- the file
`c:\mywebs\mysite\cgi-bin\forum\test.cgi`
is called as
`http://127.0.0.1/cgi-bin/forum/test.cgi`

- Any other file not under `c:\mywebs\mysite` results in an error.



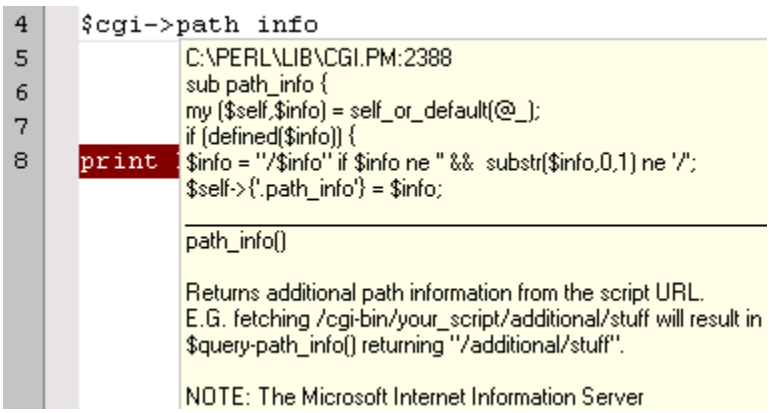
Notice that in all cases, after the host part (http://127.0.0.1/) the relative paths of the file compared to the webroot are appended.

The best thing to do is actually have your entire site offline, copying the directory structure, and setting as webroot the folder you copied all the files. This is important, so you can develop scripts and html pages that will work both offline and remotely, without any (or minimal) changes.

Let's assume your site is http://www.site.com/ and you can access your site with an FTP program. From path / your entire site starts:

```
/index.html
/images/background.gif
/cgi-bin/form.cgi
```

Copy all these files to a folder on your computer, like `c:\mywebs\coolsite`. Select as the internal server's webroot the folder `c:\mywebs\coolsite`.



If you use html publishing software, then you already have your entire site offline. Select as webroot the folder containing all your files and subfolders of the site.

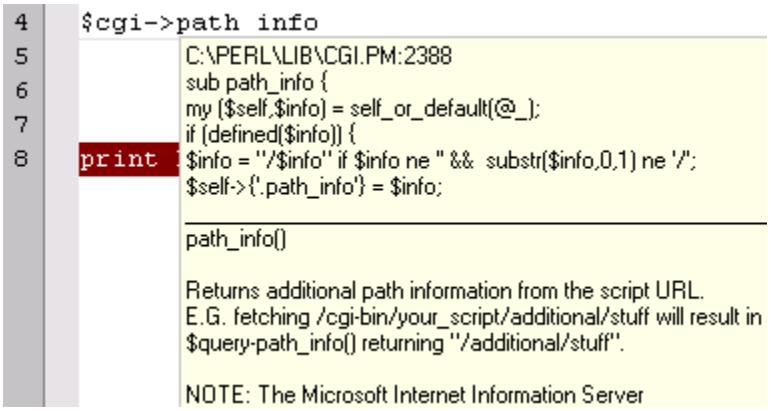
Now just like `http://www.site.com/` will send `/index.html`, the same thing will happen by calling `http://127.0.0.1/` or `http://localhost/`, without being connected to the internet. You can try loading index.html in optiperl and pressing run, or entering the above address in the Browser menu / URL and pressing "Go to url".

Let's assume index.html has a form that calls form.cgi. It could have something like:

```
<form method="POST" action="/cgi-bin/form.cgi" name="Form">
```

If the webroot of the internal server is correct, this will work on both servers without any change. When running `http://127.0.0.1/index.html`, the browser will call `http://127.0.0.1/cgi-bin/form.cgi` after pressing "submit". On `http://www.site.com/index.html`, the browser would call `http://www.site.com/cgi-bin/form.cgi`.

Before setting the webroot, think how the url's would be translated. Suppose by error, we set the webroot to `c:\mywebs\`. Now `index.html` would have to be called as `http://127.0.0.1/coolsite/index.html` and the form.cgi as `http://127.0.0.1/coolsite/cgi-bin/form.cgi`. We would also have to change the Action in the html page to `action="/coolsite/cgi-bin/form.cgi"`. All the above would work, but after uploading to www.site.com, the html page would not work, since `http://www.site.com/coolsite/cgi-bin/form.cgi` does not exist.



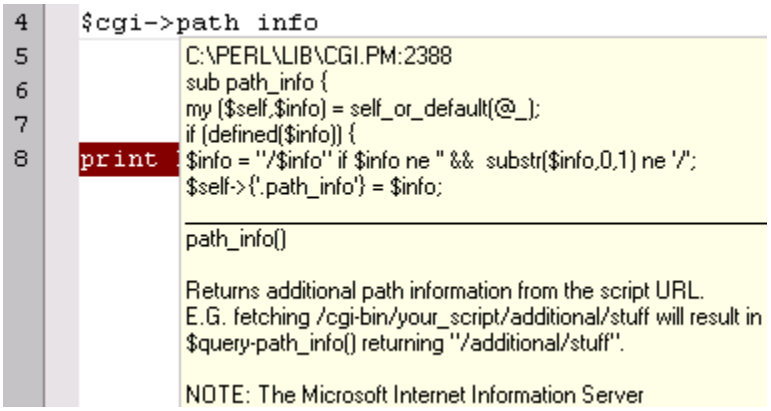
However your site might have a different directory structure (like a different htdocs and cgi-bin folder), but OptiPerl can support any configuration using aliases. Read more in possible scenarios.

More about the internal server

The shebang line

The shebang line is the first line of the script that calls perl. It looks something like `#!/bin/perl`

On all servers, it is required that it points to a valid path to perl. You should check out with your web host to see what the exact path is, since it will be required when uploading.



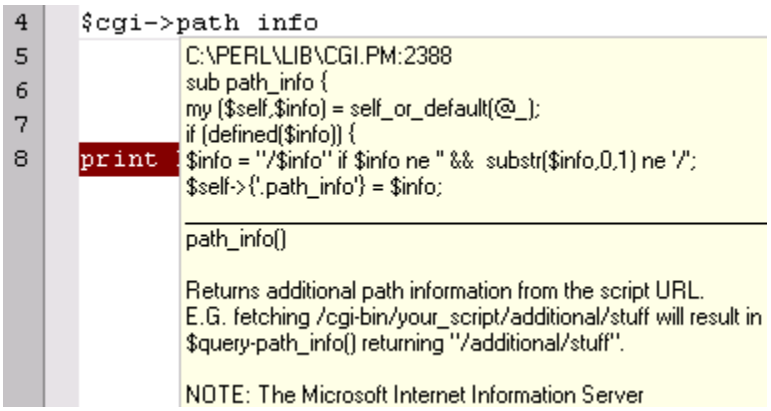
But you don't need to change it on each script; OptiPerl will do it automatically when uploading if this is setup in the remote transfer sessions.

However when using the internal server, this line can point to anything, because it is not used. The internal server knows when to load perl by the file extension, and this can be setup if needed from "server associations".

Strange headers and 500 errors

On a production server, if you upload and something is wrong with your script, you get a "500 error" page when running it. If you are lucky enough to have access to the servers error logs then you can see what was wrong. Another issue is when warnings occur before the header is printed. They are not shown anywhere, but they might pose a security risk.

OptiPerl's internal server on the other hand never gives a 500 error, and never swallows a warning message in the header. Actually it will try to analyze the output and tell you what is wrong. You can identify this easily when running, from the yellow box that appears on top of the output:



If the error is fatal, then this will be the only output. You may also get the full output, plus a warning box about a strange field in the header. This will probably be a warning message from one of the modules the script uses.

Remote debugging via loopback

The internal server is optimized for this work. Read more about Remote debugging.

Access and error logs

The internal server does not generate access and error logs, like most servers do. Instead it prints every request and response plus details of what it does in the "server talk" tab of the web browser.

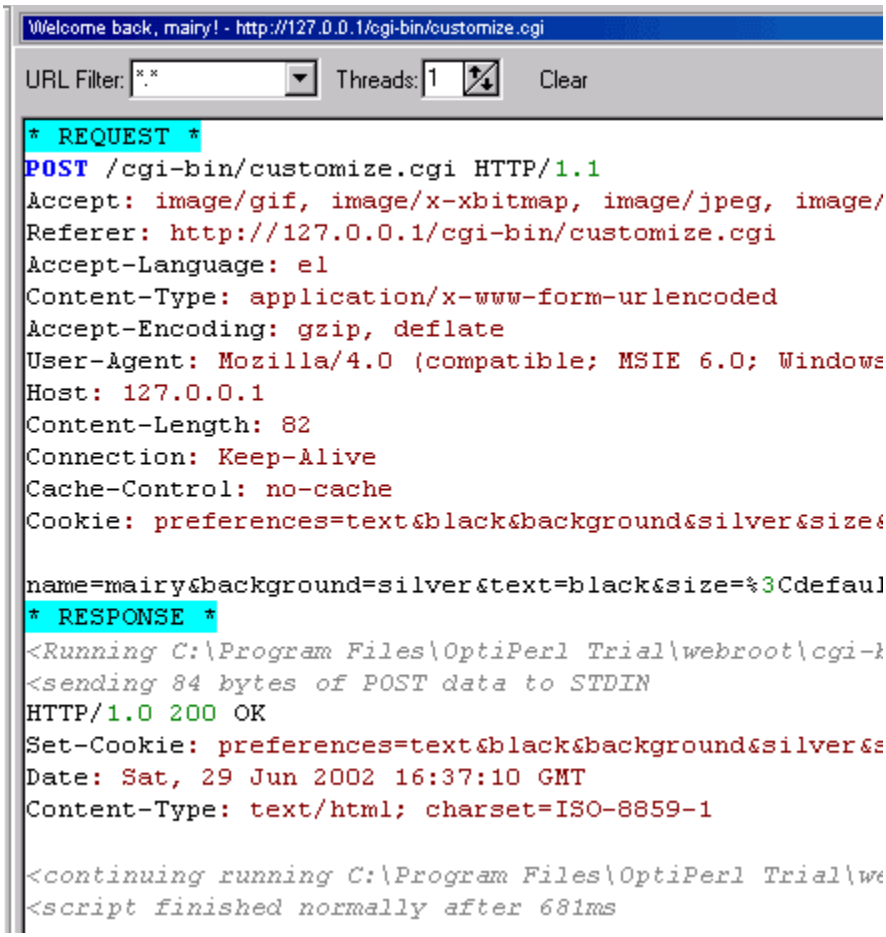
Internal server associations

The server associations affect the internal server. This list contains associations of file extensions an external programs. The default is the association of cgi,pl,plx files to perl.

If you have other associations in your web site that link to perl scripts, or want to set-up other programs also, changing the values will be necessary.

Server talk tab

The server talk tab of the web browser is an extremely detailed log of the headers sent an received from the internal server, and it is also used when the "Spy HTTP proxy" is enabled while browsing the internet.



Under the * REQUEST *, the complete header send from the client (the web browser) is shown. You can see all the queries sent plus the post data.

Under the * RESPONSE * you can see the complete header sent from the server in response to request. If the internal server is being used, you will also see the what the internal server is doing.

Using the spy HTTP proxy

If enabled, all headers sent and received by OptiPerl's internal browser while browsing the internet will be shown. To use, enable it, and navigate to any URL by entering the address in the menu "Browser" and pressing "Go to URL"

Use the spy proxy on your own scripts after uploading on your server.

Using the spy HTTP proxy on an external server via loopback

If you are running on an external server via loopback, you can also see the headers sent and received, but you **must** change the "Host url" in the Options or Project options dialog to `http://localhost/` or `http://<name of machine>/`

Settings how many threads to watch and filtering headers.

If you access a page that loads many graphics and / or scripts, the output might get mixed up from the multiple requests. Increase the number of threads watched (this does not affect the internal server) and add a wildcard to show only the requests / responses on the matching files.

External servers

While the internal server of OptiPerl can be used for testing most websites, there are some case were a third-party server must be used:

- If your website relies on SSI (server side includes), then you must use a server that supports them. OptiPerls internal server does not support SSI.
- If you also use other scripting languages like ASP and PHP.
- If you use mod_perl and want to test the scripts in a more realistic environment.
- If your site relies on settings in .htaccess files.
- OptiPerl is intended for both beginning and advanced users; advanced users may want to use a real production server running offline for developing. Still however in most cases, even in very complex projects the internal server will do fine.

Any server however integrates very well with OptiPerl. Many settings, like in the Options and Project options under section "External server" are used solely for this purpose. The settings needed to integrate are:

Access Log File - Error Log File

The access log file of the external server. Most servers create files that describe the errors and accesses made to them. Enter the path to the files here. You can view them from menu Server / View error & access logs. In the error log, if standard perl errors are detected with line numbers, then double clicking will go to the offending line. *Examples:*

```
c:\apache\logs\errors.log
c:\apache\logs\access.log
```

Executable

The filename only (without a path) of the server executable. Used by OptiPerl to check if it is loaded. Using this filename, OptiPerl searches in the task list of the computer to see if the server is loaded. If it senses that it is loaded, then it will turn off it's internal server automatically and also gray out the menu item to load it (you cannot have two servers loaded at the same time). After OptiPerl senses that an external server is loaded, it will use all the settings under the "External server" section described here. *Examples:*

```
apache.exe
```

Consider external server loaded

If you are running an external server and OptiPerl cannot find it then check this option, so OptiPerl will manually consider the server loaded and use all the settings under "External server".

Document root

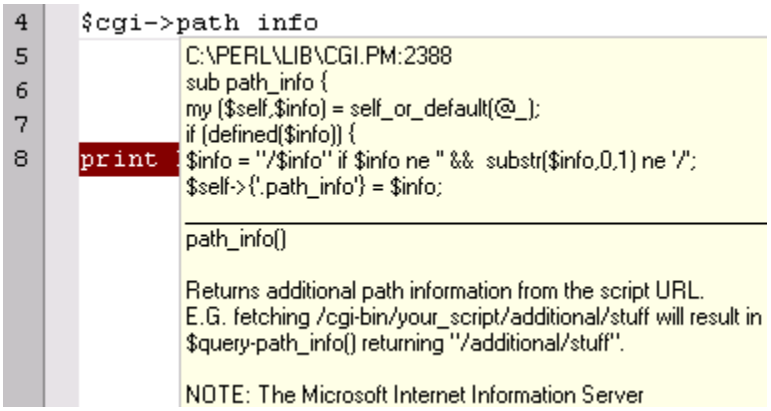
The document root path of the external server. This is usually the path that contains the html documents of the site. *Examples*:

```
c:\apache\htdocs
c:\mywebs\www.mysite.com\
c:\mywebs\mysite\public_html
```

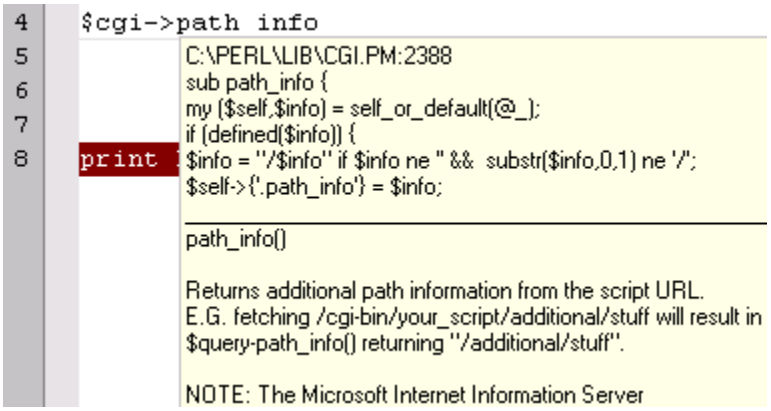
Aliases

Aliases used for external server, column separated. Example:
`/cgi-bin/=c:\webroot\cgi-bin;`

This is necessary in some cases, like when the cgi-bin folder is not under the document root folder. Read more about Aliases.



With remote debugging via loopback, the internal server adds automatically in the script's environment the value needed to invoke the debugger. If you are using however an external server, you will need to do some set-up from menu Debug / Setup and information.



In some cases, for each project, you may want to have a different server, or a different server configuration. All the above options can be overrider per project from project options.

More Information in the next sections:

- About Apache
- About mod_perl
- About Internet Information Server

Also read the section "External Server" in the FAQ.

Aliases

Aliases used for the internal or an external server

Probably you will want to have your entire web site off-line, with all it's subfolders, exactly as on your remote host, so that developing will be easier. However if all the html documents and scripts are not under a single folder, you will need to use aliases, to correspond URL addresses and absolute paths to the files.

For example if you have your files under `c:\webs\myweb`, and your cgi-bin folder is under `c:\webs\myweb\cgi-bin` then aliases are not needed. Setting a webroot of `c:\webs\myweb` will map correctly all URL requests to the files, bu the path in the URL address. `http://localhost/index.html` and h`ttp://localhost/cgi-bin/test.cgi` will view files `c:\webs\myweb\index.html` and c`:\webs\myweb\cgi-bin\test.cgi`.

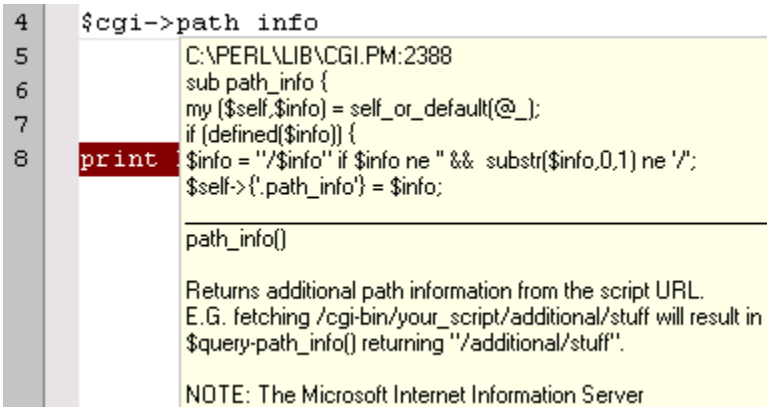
But if your server cofiguration has a cgi-bin folder outside of the path were you have html documents (like apache servers) or you have other mappings of url addresses to real paths, then you need aliases. For example, if you have folders:

c:\webs\myweb\htdocs that hold all your html files and
c:\webs\myweb\cgi-bin for scripts, then you will need to setup an alias:

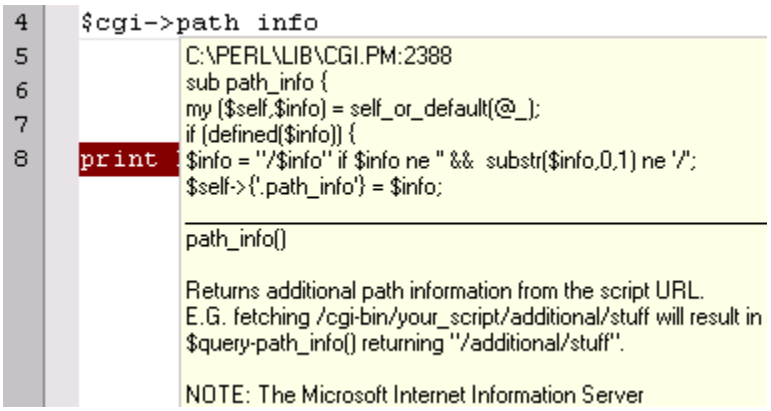
```
/cgi-bin/=c:\webs\mysite\cgi-bin\;
```

This way accessing `http://localhost/cgi-bin/` will get `c:\webs\myweb\cgi-bin` and not the incorrect `c:\webs\myweb\htdocs\cgi-bin` (presumming you have set up `c:\webs\myweb\htdocs` as document root).

If you use the internal server, enter this in the "aliases" field of the internal server. If you use an external server, enter in aliases for the external server.



See common scenarios for some common set-ups.



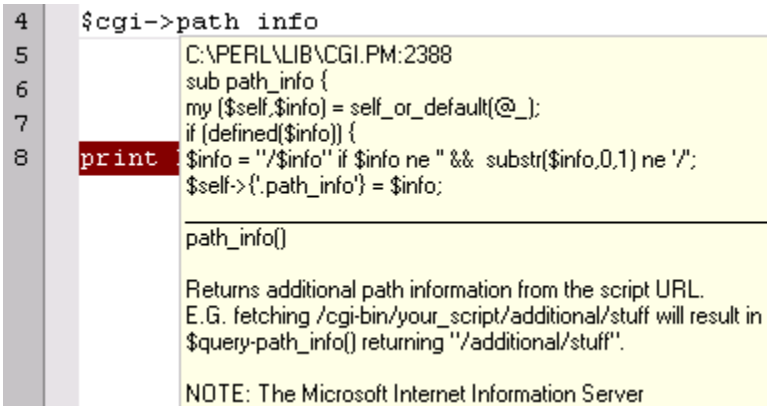
If you use Apache, the aliases in it's httpd.conf file can be parser automatically by pressing the "parse" button in the options or project options.

Aliases in the Remote transfer session window

Aliases in the "Running remote files" section of the transfer session setup are like the above, but work the other way around. Optiperl knows the remote path of the file, from when you downloaded it. If you try to run, optiperl will guess the url that needs to be sent to access it. Again all your files are under a single folder, aliases are not needed. Here are some examples:

Document Root: `/htdocs/`
Links to: `http://www.mysite.com/`
Aliases: `/cgi-bin/=http://www.mysite.com/cgi-bin/;`
`/perl/=http://www.mysite.com/perl/;`

With the above, a remote file opened from `/htdocs/` will be run as `http://www.mysite.com/index.html` and a file from `/cgi-bin/` will be run as `http://www.mysite.com/cgi-bin/test.cgi`



See common scenarios for some common set-ups.

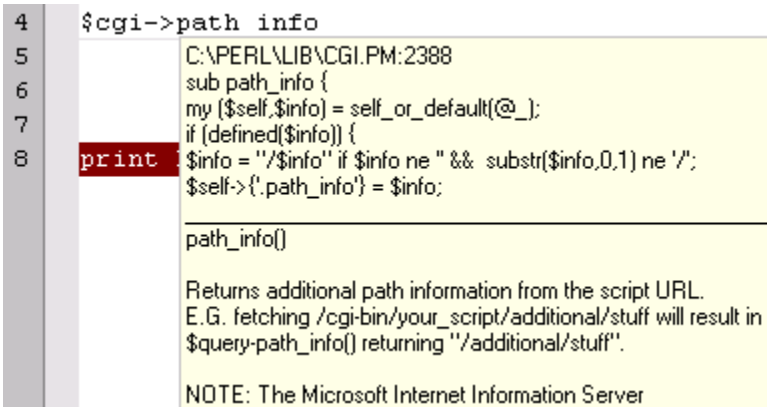
Using Apache

Here we will explain how to set up Apache to run your scripts off-line. This section is not necessary, but you might want also to test your webpage under a real server. This might especially interest you if your internet provider uses Apache as it's server. Many internet hosting services do use Apache.

There are many distrubutions of apache. We recommend a distrubution that comes bundled with perl and also mod_perl. See the setting up section. We also recommend installing in a root folder like `c:\apache`.

After installation, copy your website files under the `htdocs` and `cgi-bin` folders. If you installed under `c:\apache` enter the following in the Options dialog under section "External server":

- Access log file: `c:\apache\logs\access.log`
- Error log file: `c:\apache\logs\error.log`
- Executable: `apache.exe`
- Document Root folder: `c:\apache\htdocs`
- Aliases: `/cgi-bin/=c:\apache\cgi-bin`



You can also import apaches httpd.conf file instead of entering the above. It's path is c:\apache\conf\httpd.conf

Executable is the executable name of the server of you are using, not the full path to the server. Giving the correct executable name, enables OptiPerl to know when the server is loaded or not and if so send the correct relative directories to the URL when pressing "Run". If you load or unload the external server while OptiPerl is running, you can check if it has found it in the Server Menu (You will see a label "External Server Loaded").

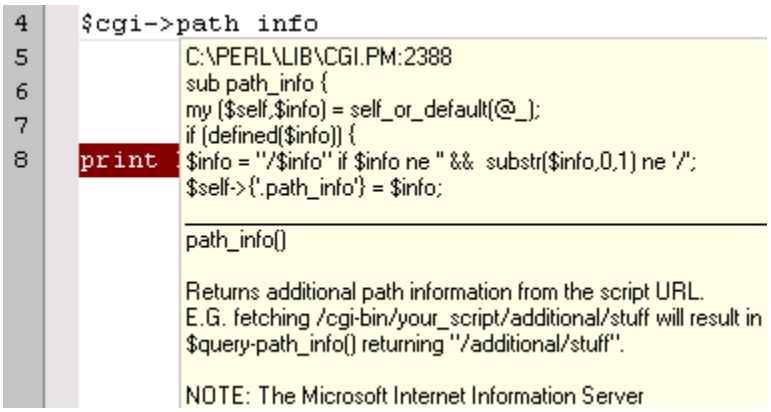
If you are using Windows NT and OptiPerl still cannot determine if the server is loaded or not, then manually press "Consider External Server Loaded".

The log files are used to have a live view of them in the Log Windows

Apache runs by selecting "Start Apache" from the start menu. Start it (a console window - minimize it) and your web browser, and enter `http://127.0.0.1` (you don't have to be on the internet). If you get a message "Go offline" press cancel. You should see the Apache's help. If not, try putting any html document named index.html in the directory `c:\perl\htdocs`, or `c:\apache\htdocs`.

Note that whenever CGI Scripts are to be run, apache must be loaded and working. If apache's window opens and closes very fast, it has not been loaded. Check the F.A.Q if this happens.

Whatever CGI script you put in directory `C:\apache\cgi-bin`, can be run by putting an address in your browser `http://127.0.0.1/cgi-bin/test.cgi` etc. A good idea is to try this also. Copy hello.cgi into the directory c:\perl\cgi-bin and enter in your browser the above address. If you get errors, something is wrong.



IMPORTANT: To run CGI scripts in windows using Apache, you must change their first line to

`#!C:\perl\bin\perl.exe` (where you have your perl interpreter)

When you upload them afterwards to the server, change to

`#!/usr/local/bin/perl` (or wherever perl is at the server)

Optiperl can change the above automatically when uploading either when publishing your project or saving to a remote location.

Setting up Server Side Includes

The internal server of OptiPerl does not support SSI. However you can use apache as an external server for offline testing. Create in the \htdocs folder a file named ".htaccess" (notice the starting dot) with the following text:

```
AddHandler server-parsed .shtml
Options +Includes
```

Now all html files added in this folder that have an shtml extension will be SSI enabled. For example:

1) Copy the "hello.cgi" script into the \cgi-bin folder of apache
2) create a file named "hello.shtml" in the \htdocs folder with the following text:

```
<html>
Running an SSI<p>
<!--#exec cgi="/cgi-bin/hello.cgi" -->
</html>
```

You should see the following output when running the shtml file (make sure apache is running)

```
Running an SSI
Welcome to OptiPerl!
```

Apache and mod_perl

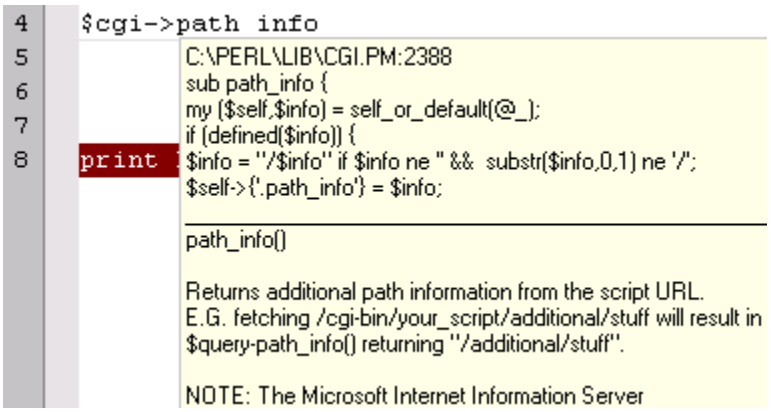
You can develop mod_perl scripts using OptiPerl. Usually scripts optimized for mod_perl will work by launching each time perl like the internal server does, but the best would be to actually install apache on your computer with mod_perl to do it the correct way. It would be risky to develop mod_perl scripts any other way, because of the way apache runs them.

Read more about Using Apache for the set-up. What needs to added extra is an alias in Options or Project options / External server / Aliases that looks like:

```
/perl/=c:\apache\perl\
```

If you want to run remote files, on your remote mod_perl enabled server you will also need an alias in the remote transfer session, that will look like:

```
/perl/=http://www.mysite.com/perl/
```



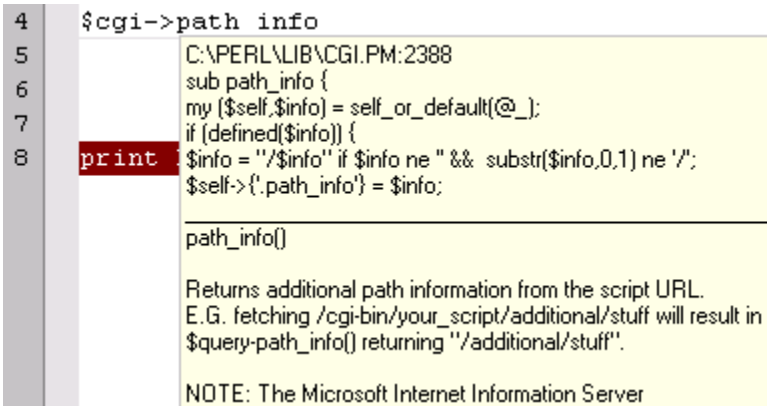
If you are porting a script to mod_perl, use the export function of the code explorer to create a list of all the variables used. Print a hard copy of the text, and check each variable for correct initialization.

Log windows

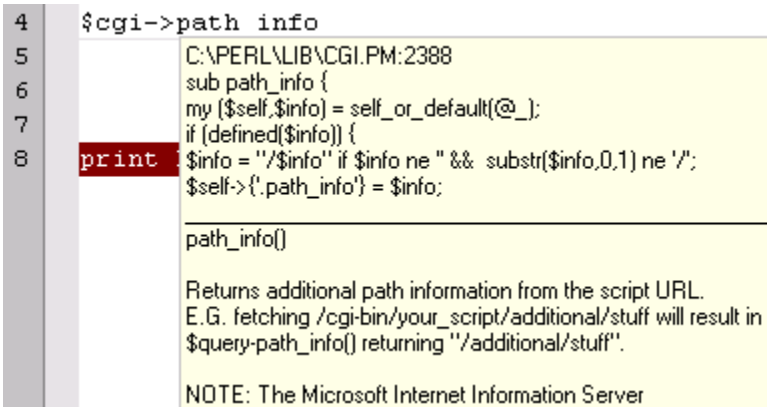
In OptiPerl you can have a live view of the logs created by the web servers. Most servers create two files logging activity of the web site:

- Access Log. This shows the files served by the server. If you have an external server loaded, then it lists the file "Access Log File" in the Options or Project Options dialog.
- Errors Log. This shows the errors encountered by the server. If you have an external server loaded, then it lists the file "Error Log File" in the Options or Project Options dialog.

Right clicking on the window enables you to make it stay on top of the application. If the logs have a line that identify a line, double clicking the line will take the cursor there.



The internal server does not produce logs. Instead it does some very detailed explaining in the "Server talk" tab of the webbrowser.



However if using an external server via loopback, you cannot produce the same output in the "Server talk", even if you enable "spy proxy". If the host is set to http://127.0.0.1/. Change to http://localhost/.

Using Internet Information Server

Internet Information Server (IIS) can be used instead of OptiPerl's internal server or apache in a windows 2000 or NT environment. To use it with OptiPerl, you need ActiveState's ActivePerl installed also.

Windows 2000 Professional ships with IIS. To install go to Control Panel / Add remove programs / Add Windows Components and check the checkbox "Internet Information Services". Install all of it's components.

After installing, find the folder c:\inetpub\wwwroot and create in it the folder cgi-bin.

Now go to Control Panel / Administrative Tools / Internet Services Manager and note the name of your computer. Expand this node and then expand the node "Default Web Site". You will see as a node the newly created folder cgi-bin. Select it and press "Properties".

On the tab "Virtual Directory" select "Execute Permissions" as "Scripts and Executables". Also press the button "Configuration" and then the button "Add". In the "Add/Edit Application Extension Mapping" window, enter the following:

Executable: C:\Perl\bin\PerlIS.dll (make sure you have ActiveState's ActivePerl installed)
Extension: pl

Repeat the above for the extension cgi and press ok to save settings.

Go to Control Panel / Administrative Tools / Personal Web Manager and press the button "Start" if the service has not already started. Press the link in the window that starts with http:// to test if it is running (e.g. http://computer)

To test that everything is OK, copy all the examples of OptiPerl from \program files\optiperl\webroot\cgi-bin to the folder \inetpub\wwwroot\cgi-bin. The scripts should be accessible by entering in your browser http://computer/cgi-bin/script.cgi (replace computer with the name of your computer shown in the window).

To set up OptiPerl, go to Tools / Options / Server and enter in the "External Server" box:

External Server: inetinfo.exe
CGI Folder: c:\inetpub
HTML Folder: c:\inetpub\wwwroot
and press OK.

Load now a cgi script from the folder c:\inetpub\cgi-bin and run it. Make sure you have selected the option server /run with server.

All scripts you create must be saved in the above folder. Note that if you have enabled IIS, then the internal server of Optiperl will not work. You can easily start and stop IIS from the window Control Panel / Administrative Tools / Personal Web Manager.

Using IIS is recommended if you have a web page that uses FrontPage extensions and want to test it offline.

Introduction & scenarios

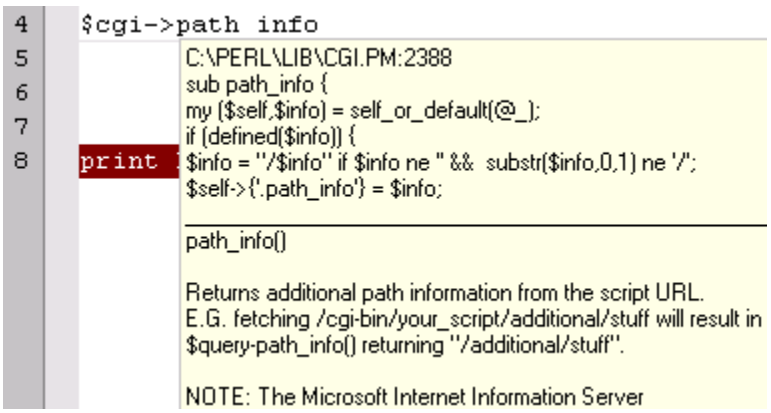
When you have finished developing your script offline, if it's meant to be used on a production server, you will need to upload it

OptiPerl uploads files two ways: Either by saving the file to a remote location from menu File, or when publishing a project. Both methods send files via FTP or SFTP, and the setup is done from the Remote transfer session dialog in menu "Tools".

Transfer Sessions

OptiPerl uploads files two ways: Either by saving the file to a remote location from menu File, or when publishing a project. Both methods send files via FTP or Secure FTP, and the setup is done from the Remote transfer session dialog in menu "Tools".

A transfer session describes the server and parameter for uploading files, plus parameters when running a remote file.



Remote server fields

Session Name
Name of the session. This is used to identify the session in the project or opening/saving a remote file.

Type
Type of transfer session.

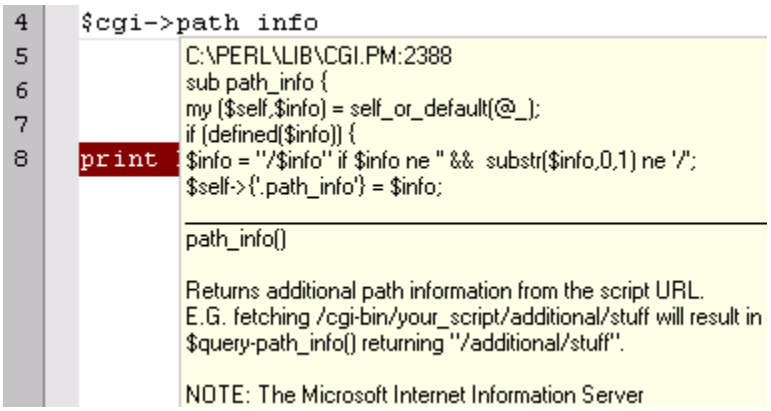
Server
Address of the server, for example `ftp.mysite.com`

Username
Username used to login

Password
Password to login.

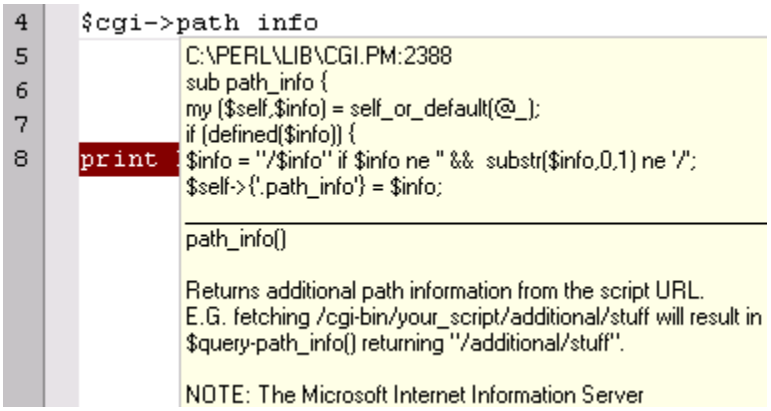
Account
If your server needs an ACCT command, enter the account here. Usually not used.

Port
Port of the server, usually 21



Firewall settings

If you connect via a firewall, enter here the settings of it.



Remote running

The fields here affect running a remote file. Read more in running remote files.

Document Root
The path of the server that links to a URL address. This is usually just `/` (the root) or `/`htdocs.

Links to URL
The URL for calling files in the above path. For example `http://www.mysite.com/`

As an example, if you enter `/htdocs` as the document root and `http://www.mysite.com/` as the URL, and you open the file in the path `/htdocs/news/index.html` , when pressing Run, optiperl will call `http://www.mysite.com/news/index.html`

Aliases
For many servers, entering the 2 above fields will be enough for the entire internet site. However your server might have additional paths that link to URL via aliases it has. For example apache with mod_perl does the following:

/perl/ Links to http://www.mysite.com/perl/
/cgi-bin/ Links to http://www.mysite.com/cgi-bin/
/htdocs/ Links to http://www.mysite.com/

This would need two additional aliases entered:

```
/perl/=http://www.mysite.com/perl/;/cgi-bin/=http://www.mysite.com/cgi-bin/
```

If you did not setup the above, then a file in /cgi-bin/ would no be able to be called from OptiPerl, since it is not a folder under /htdocs/

Shebang line
How the shebang line should look. This depends on the path to perl of your server. For example, enter here something like:

```
#!/bin/perl
```

Note that the shebang gets changed only the original has a shebang; It is never added.

Change when uploading
Whether to do the change to the shebang when uploading / downloading. This option is here so it can be easy to temporarily disable it.

Automatically convert version
Whether the version converter should convert your version to "SERVER" if you are uploading or publishing, and to version "LOCAL" when downloading from menu File / Open remote file.

Notes
To keep notes about the server.

Save passwords
If selected, the passwords for the server and firewall will be saved in OptiPerl's Application data folder. If this is deselected you will be prompted on each session of OptiPerl.

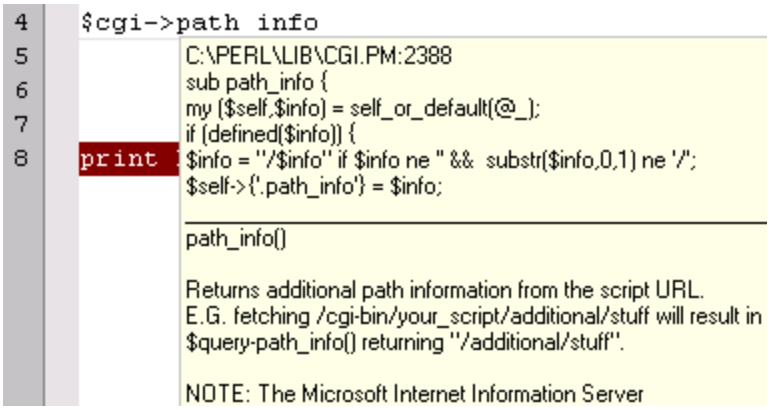
Under the database, there is a navigator for all the sessions in the database:

"+" Adds a new record
"-" Deletes a record
"Up" edits the selected record
"v" Saves changes from a previous edit
"x" Cancels changes from a previous edit

Open & Save to remote location

Just like local files, OptiPerl can open and save a remote file on a remote server. This is done from menu File / Open and Save to remote location. When doing this, you have some extra features:

- All the FTP sessions are handled from the Remote transfer sessions dialog. Especially the section "Remote running" has to do with files opened with this method.
- If "Run with server" is enabled, different rules apply to the URL that gets sent to the browser, and depend on the section "Remote running" in Transfer sessions setup dialog. For example, you can setup the transfer session, so that running the remote file `index.html` downloaded from `ftp.mysite.com` at path `/htdocs/index.html`, will actually call `http://www.mysite.com/index.html`. This is done with the "Document Root", "Links to URL" and "Aliases" field in the above section.
- If the option "Upload automatically" is enabled in the Options dialog / Running then the file will be uploaded automatically when edited before running.



A remote file however when opened, is not only saved in memory; a temporary file is also created in the Application Data folder under subfolder Sessions\Session Name\Remote Path to file\Filename. This has some advantages:

- All internal and user tools that require a real file to function work.
- You can open cached files from menu File / Open cached remote file that know automatically that before running will need to be uploaded first.

Project Publishing

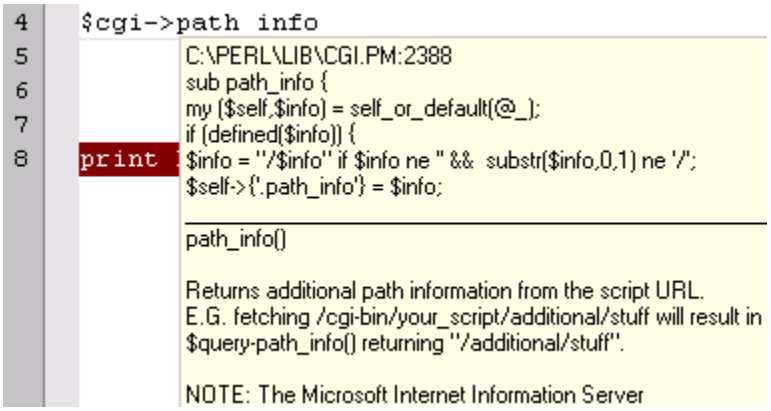
The publish function of the project manager, uploads all modified or new files to the remote transfer session you have selected in the project options. After the upload, you can get a detailed log of the transactions from menu project / Log.

Just like when uploading via the save to remote location, the following happen before each upload in the background:

- The version is converted to SERVER
- The shebang replaced

However in projects you can also select a transfer mode and permissions for each file and directory. These will be used if the file is first uploaded. Also any non existent directories are created.

Note that project publishing is not a synchronization of local and remote folders. Read more the technical details of publishing in "working with projects", and how to set-up your project.



Project publishing is optimized for speed. If you are working on many files that are associated together, and want to test them on your remote host, pressing the "publish" button is very fast. Only modified files will be uploaded each time.

Version Converter

The version converter is for quickly making two or more versions of your script because of the changes that have to be made in the local version of your script (the one used when off-line for testing) and the one you want to publish. Usually changes have to be made that include:

- Some commands that do not work under windows like flock
- Paths to files that Perl writes that are Unix specific. For example if you want to write a file like "/docs/logs/count.log", in your local version in windows, this must be changed to "count.log" or "\cgi-bin\docs\logs\count.log" if you have created those two directories under the server's root.

To fix these problems, you can modify your scripts code to run in two ways

First add somewhere in the beginning of the script this line:

`#@Local#?#@Server` if you are editing the local version or

`#@Server#?#@Local` if you are editing the server version.

This way OptiPerl "knows" what version is run. After adding the above, in the search menu you will see which version you are editing in the "Version: xxx" line.

Example

Enter the following lines:

```
#@Local#?#@Server
#?flock(OUTF,2);
my $semaphore_file=
 'counter.sem';#?'/tmp/counter.sem';
```

Notice that in the Search menu you will get a label "Version: Local"
Now press "Swap Version", and the above will change to:

```
#@Server#?#@Local
flock(OUTF,2);#?
my $semaphore_file=
 '/tmp/counter.sem';#?'counter.sem';
```

Notice that all lines with a '#?' where swapped at that point.
Also the search menu will be updated with the label "Version: Server"
If you press again "Swap" then the first version will be back again.

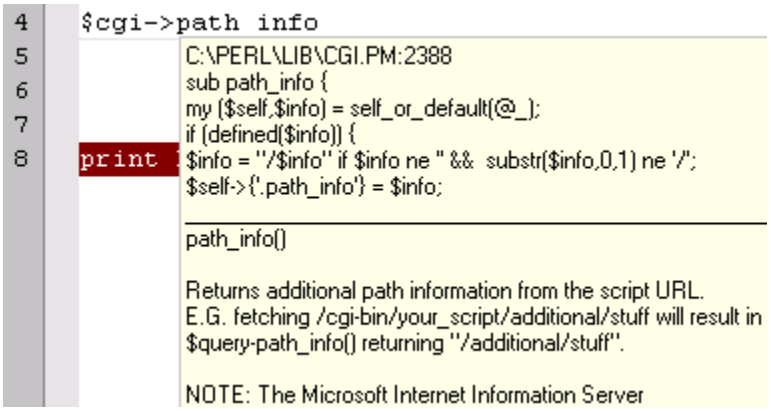
You can also add longer lines with many version remarks "#?", for example:

```
#@Version A#?#@Version B#?#@Version C

print "A";#?print "B";#?print "C";
```

Here each time "Swap" is pressed, the next version is selected. Accordingly the Version is update.

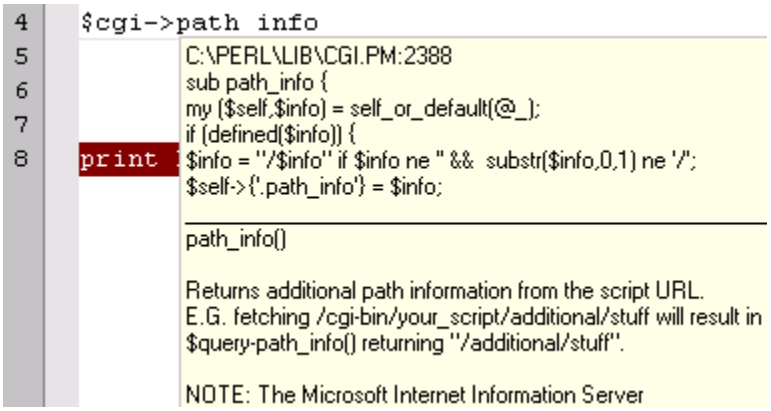
If you do not add the `#@Server#?#@Local` line, then the version converter will still work, but OptiPerl will not know which version you are editing.



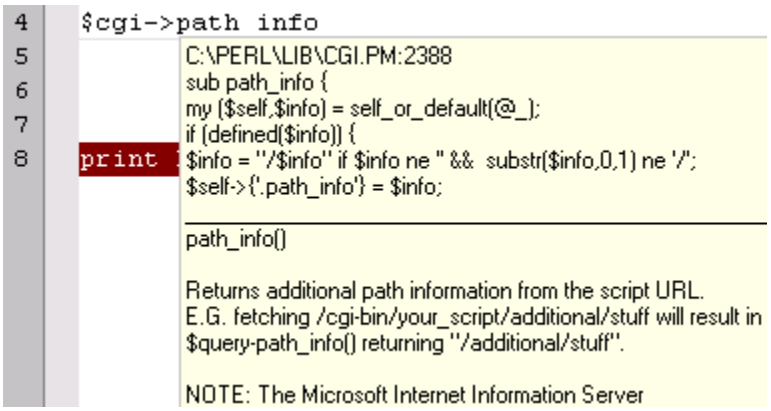
There is an advantage to letting OptiPerl knowing which version you are editing and using specifically the labels "Server" and "Local":

- When opening or saving a remote file to a server, the version gets automatically converted.
- The project manager will make the conversion automatically in the background before uploading.

Both of the above will work only if you have selected "Version convert to server" on the transfer session you are using. The update will not actually be shown in the editor, but done in the background.



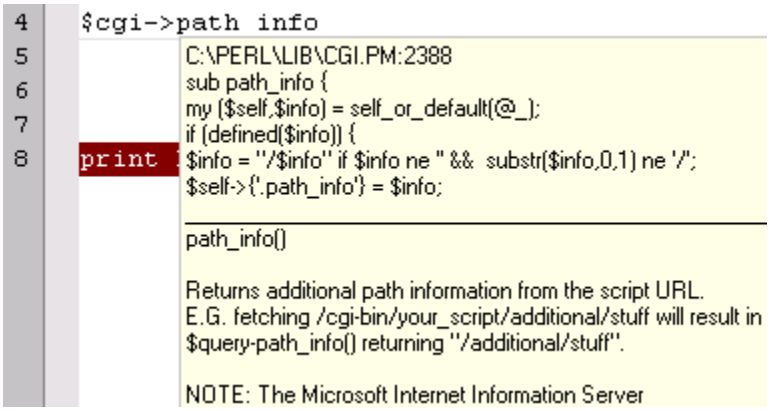
Do not use version converter remarks on the shebang of the line; the shell does not like this. The shebang is treated also when uploading, if the corresponding option has been selected in the transfer session.



Included is counter.cgi, that demonstrates writing two versions of a script.

About debugging

When you are done programming and run your script, you want it to do what it was designed for. However usually it will not, it might produce unexpected results, probably because your code is not working the exact way you except it to do. If your program is over 10 lines and uses variables (like most script are!) then there are two things you can do to correct the code: a) Keep changing what could be wrong, running continually until you "get it right" b) Debug it.



Bugs and syntax errors are different. Syntax errors are parts of the code perl cannot understand, and can be fixed using error testing. Bugs however are parts of the code that perl understands, but do not work as expected.

Debugging is like running, but slowly, at the pace you want, line by line usually, watching what exactly is happening with your code. While debugging, you can watch for example how variables change values or how the if statements work. It is like examining the tree rather than the entire forest.

After you design a script, development includes a cycles of coding and debugging. Upload to a production server only after you thoroughly tested your script.

When the debugger is started, you will see the first line of your script get colored noting it is the that this is the line about to be executed (we will call it the execution line). From here you can do the following (all selected from the Debug menu):

- **Single step.** Selecting this will process the execution line. If the line calls a subroutine, then you will enter the subroutine.
- **Step over.** Will process the execution line as above, but if it calls a subroutine, then the subroutine will also be called, without the flow to enter it.

These are the basic and important commands of the debugger.

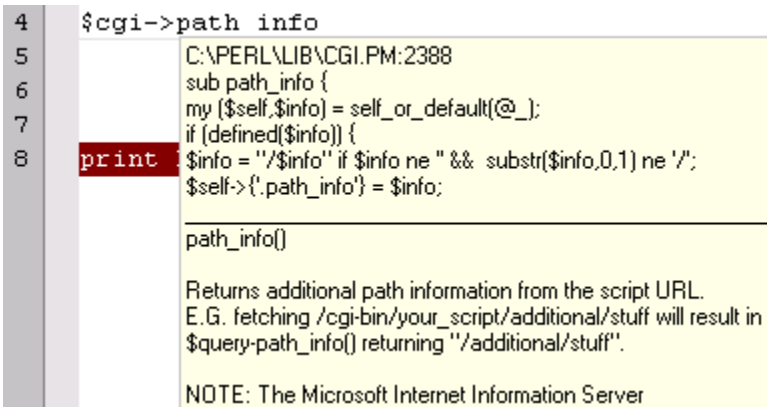
- **Return from subroutine.** If you are in a subroutine, it will be executed untill it ends, and return the execution line to the line after the line that called the subroutine. If you are not in a subroutine, then the script will be executed until it terminates.

- **Continue script.** Like the above, but will execute your script until termination even if the execution line is in a subroutine.

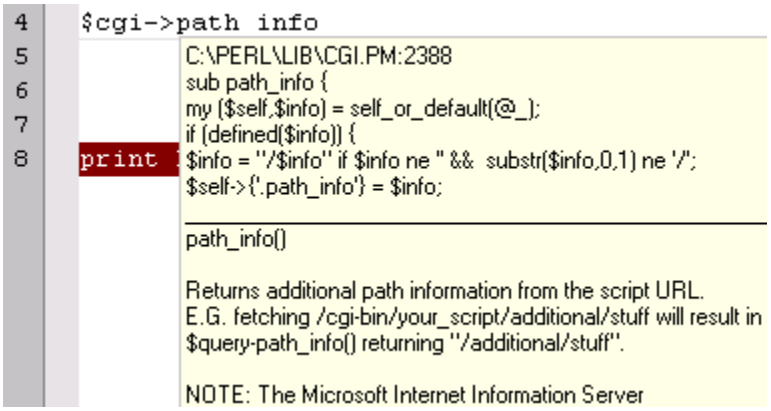
Often you know that a part of the script is correct, but need to debug a part after the good part. You don't need to trace every time the good part because you can use breakpoints:

- **Breakpoints**. Enabled or disabled, by clicking on the gutter, next to the line you want to set as a breakpoint. If you have set a breakpoint in a subroutine, and the execution line is about to call this subroutine, then selecting "Step Over" will run the subroutine until it reaches the breakpoint, and this will become the execution line (the script will pause running again). In the same manner, selecting "Continue script", will stop on a breakpoint, if the program's flow passes over it.

Most perl programs also use modules. If the execution point is about to run a sub it a different module, then the other module will be opened in the editor, and show the new execution point.

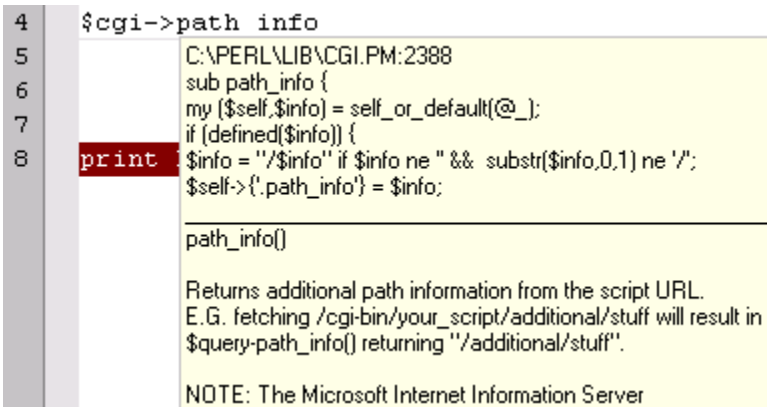


If you are using modules that are included in standard perl distribution, you know they work; they don't need any debugging. So if the subroutine you are calling in the execution line belongs to them, you can select "Step Over". This way you can avoid entering well known working code. You might want this of course if you are debugging your own modules.



On some lines, there is no sense on adding a breakpoint, for example lines that contain only remarks, because these lines will never be executer. After the debugger loads, valid breakpoints are noted with a tick graphic in the gutter, while invalid have an x mark.

- **Viewing a value of an expression.** An integral part of debugging is examining the value of a variable while the execution has stopped. If you have Auto-Evaluation enabled, hover the mouse over a variable or expression to see it's value. You can also select a part of the text and hover over the selection too see how this expression is evaluated.



Note that the word under the cursor is evaluated OR the selected text under the cursor. Selecting first the text can help you evaluate expressions with spaces like:

```
print "<b>$k</b> = ".$ENV{$k}."<p>";
```

Select first the above expression and the hover the mouse on it.

- **Watches.** See more in the Watches section.

Debugger's Status

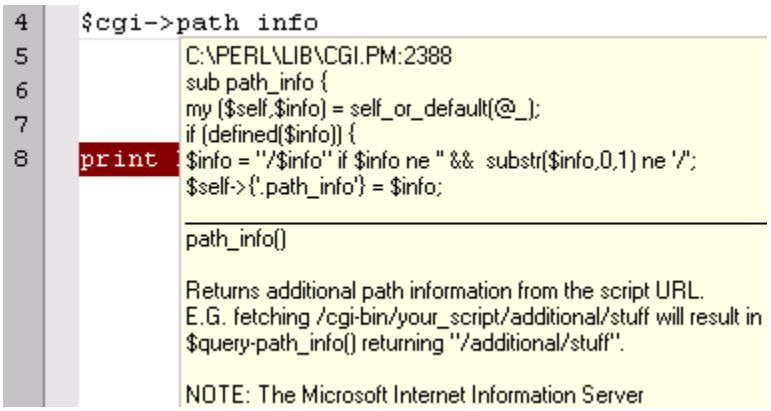
The debugger's status is displayed in the status line of the main editor. Status can have the following values:

- *Not loaded*: The debugger is not loaded. Press "Start Debugger" to load.
- *Stopped*: The debugger has stopped on a line.
- *Processing*: The debugger is processing a command you have sent it.
- *Terminated*: The debugger has finished execution of the script.
- *Running*: The script is running.

After your script terminates, you have an option to "Stop Debugger" or "Restart Debugger". If you will not be re-debugging your program, "Stop" is recommended, so that the debugger will be unloaded and the resources freed.

While running the script, all output is sent to the Web Browser, so you can view the output in either text or html format, by selecting the corresponding tab.

Your script might also read from <STDIN> (if for example it expects user input from the keyboard). If this happens, you will see that the status will be "Running", as the program is expecting user input. Enter the input in the edit box of the Web Browser and press "Send" to continue.



If you are debugging locally, and have enable POST in the query editor, note that the for the first <STDIN> in your script there will be no wait, because OptiPerl feeded the STDIN with the Post data.

More advanced uses

While running the script or even after it has terminated (but with the debugger running) you may also execute the following:

- List subroutine names
- List variables in package
- Methods callable

The above can be executed with entering a perl search pattern first or without

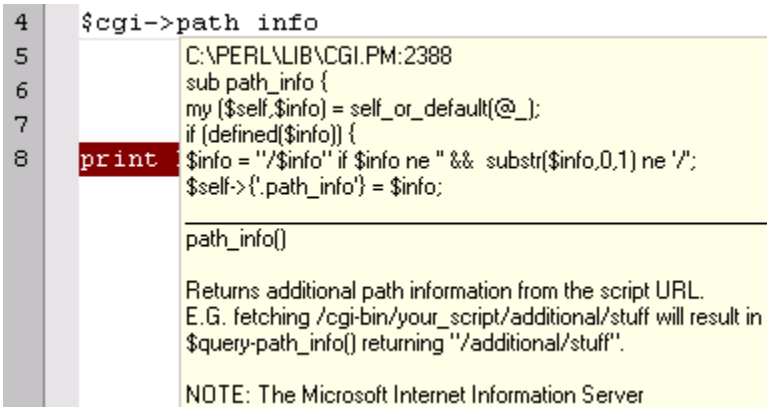
- Evaluate expression: The result from this is identical to an expression entered in the watch window.

Watches

In the watch list window you may enter expressions that are evaluated throughout the programs flow. To add a watch, right click the window and enter the expression. You can also change a value of an expression, by clicking once on the value, and typing the new value. For example if your programs uses the a hash named %h you may enter "%h" to see how it changes while your program is working. Actually you may enter any expression, not only variables.

While debugging, changing a value in the "value" column, will also change it in the program's flow.

Note that the expressions are only evaluated if the nodes that hold them are expanded. If you are debugging via a remote connection, and tracing each line is slow, then you may need to collapse some uneeded nodes for faster evaluation.



You can change the default set of watches by modifying the file "`Default Watches.txt`"

Debugging locally

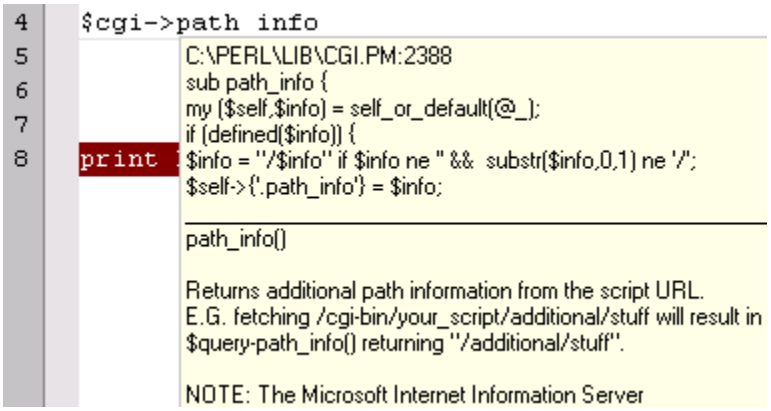
To debug a script, select menu Debug / Start debugger. Before the script starts however you may do one of the following:

- If the script is a console script, enter a number of arguments under menu Run / Arguments and select a starting path for it.
- For CGI scripts, enter a query in the query editor. The environment will be setup for the script and post data will be feeded.

Note however that debugging this way does not allow debugging after a submit button has been pressed in an html page that calls the script. To do this you need remote debugging.

Remote debugging

Remote debugging means running a script on a remote machine, but controlling it's flow on the machine running OptiPerl. Remote debugging can also be used via loopback, to invoke the debugger after an html page calls a script.



The window Debug / Setup & information provides detailed help and tools for configuring remote debugging. It is also updated automatically if your computer connects to the internet or a network, or changes it's IP.

Debugging via loopback

To debug remotely via loopback, do the following:

- Make sure you have configured OptiPerl so that the script or other scripts in the same folder can be run with the option Server / Run with server enabled.
- Open the script you want to debug and add a -d parameter to it's shebang. For example:

```
#!perl -d
```

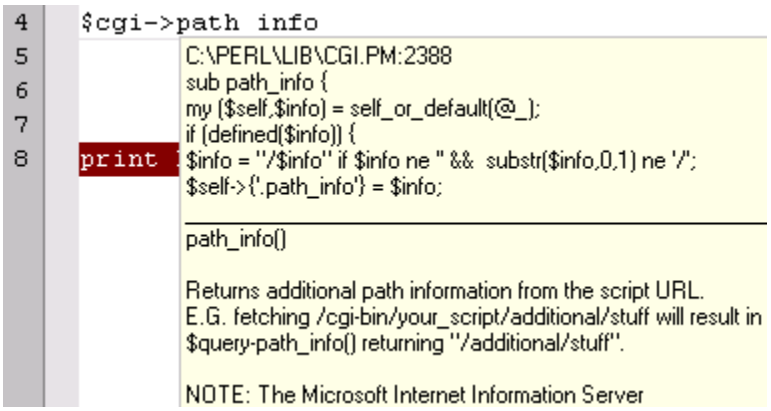
- Select Debug / Listen for remote debugger.
- If you have the internal server enabled, select Run. The debugger should be invoked and the execution line should go to the first line of the script.
- To invoke the script from an html page, load the page and invoke the script (like by pressing the submit button). The same result as above should occur.
- You can do the same from an external or secondary browser.

If you are using an external server, make sure that the computer's environment settings contains a line like:

```
PERLDB_OPTS=RemotePort=127.0.0.1:9010
```

If however you are using the internal server, this is not needed, as the internal server add's this automatically to the script's environment.

See also menu Debug / Set-Up & Information window / Debugging via Loopback for more information and setting up if you are using an external server.



Read the tutorial "Using the remote debugger via loopback" for an example.

Debugging on a remote machine

To debug on a remote machine, you will either need:

- Access to it's environment settings, so you can add the `PERLDB_OPTS` setting. Most web hosts don't allow this however.
- If it's running windows, access to it's registry. If the machine is connected via a network, optiperl can try to enter the required setting via a remote registry connection.
- In most cases, just uploading two files (read below) will do fine. This is most common way to setup on most web hosts (especially the economy class!)

To setup:

- Select File / Open remote file and open the file you want to debug on the remote machine.Add a -d parameter to the shebang. If you cannot open the file via File / Open remote machine, just make sure it has a -d parameter in it shebang (by uploading via project publishing, manually editing via the network etc).
- Set up a line in the machines environment or registry to point the debugger to the computer OptiPerl is running like:

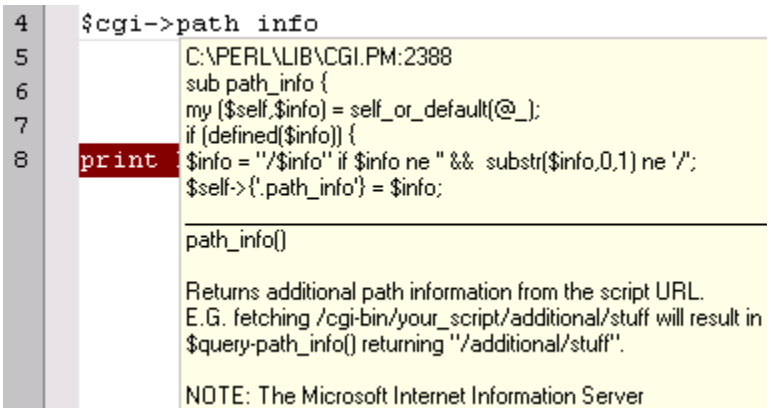
```
PERLDB_OPTS=RemotePort=197.12.3.1:9010
```

Where `197.12.3.1` the IP address of the computer OptiPerl is running on. Press the "Info" tab on the "Debug" pop-up window for more information about this.

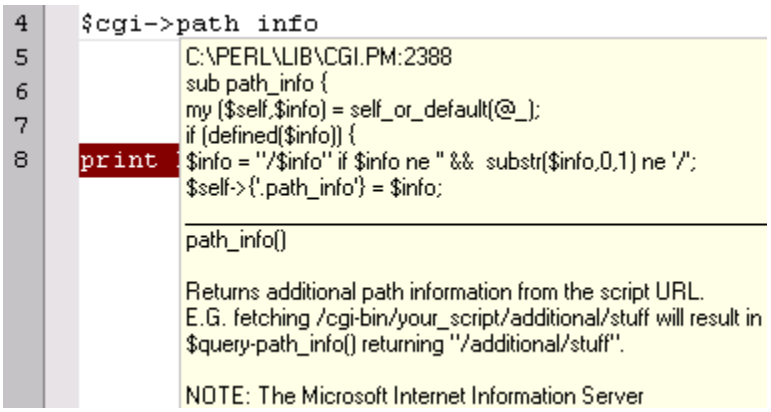
If you do not have access to the environment or registry, upload the debugger configuration files via the "Set-up and Information" dialog on the tab "Setting up using remote session".

- Select Debug / Listen for remote debugger, and invoke the script on the remote machine, either by running as a remote file if it's transfer session is set up, or invoking it by entering the URL to the script from any web browser.

When the script waiting to be debugged is runned, OptiPerl asks the remote debugger for the script's source code, so it can display it in the editor. This is why it is not necessary to actually have the script that is going to be debugged open in the editor. The source code when received, is saved locally in the Application Data folder. This way if you debug it again, it will not have to be downloaded again, unless of couse changes have been made, which OptiPerl will detect and query you to download again.



You don't need any special version of `perl5db.pl` on the remote machine. The standard version 1.07 that comes with most perl distributions will work fine (it is located in the folder \perl\lib\)



Read the tutorial "Debugging a script located on your web host" for an example.

About the debugger used and a possible problem

OptiPerl accepts any connection made from the module perl5db.pl included in the standard perl distribution. This module is called by default when you run `perl -d` from the console. However some third party debuggers might override the setting `PERL5DB` (either in the registry or by having it set in the environent) to invoke their own debugger. If this setting is set to a non standard debugging module, OptiPerl will not be able to process the data sent from the remote debugger. This is not a problem when debugging via loopback, becuse OptiPerl will make sure this value is not set.

If however you are debugging on a remote machine, you will have to make sure that on the remote machine this value is either unset or set with text like:

```
BEGIN { require "perl5db.pl" }
```

- Under UNIX, delete the environment variable PERL5DB if it exists.
- Under Windows, also make sure that it does not exist in the registry under:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Perl
```

Delete or rename the key "PERL5DB". If the remote machine is a windows box and you are allowed to update the registry by double clicking a .reg file, use the "Setup & Information"

dialog to create a .reg file and execute it on the remote machine.

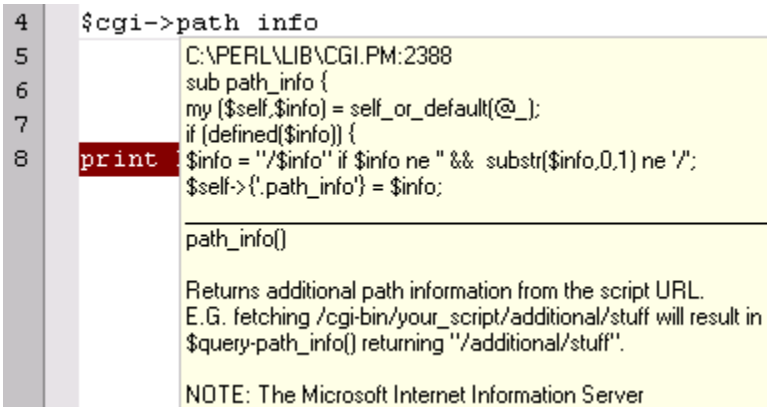
Automatically setting up a remote machine via remote session

In most cases you can setup the remote machine via FTP or SFTP by creating and sending files named `.perldb` and `perldb.ini` to the same folder that contains the script you want to debug. The files must must containt a line of text like the following:

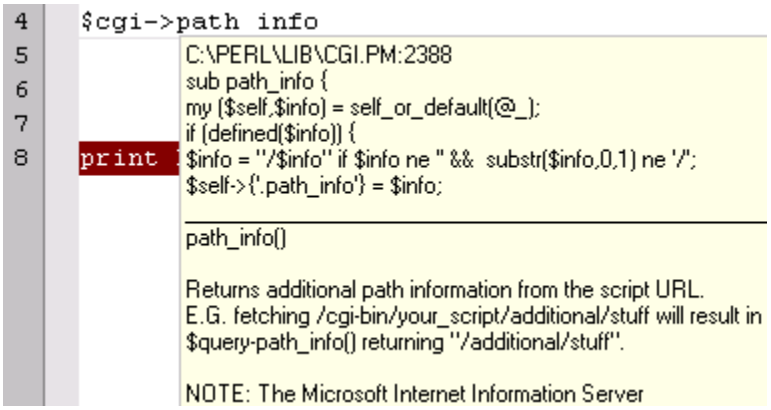
```
&parse_options("RemotePort=xx.xx.xx.xx:9010");
```

Where xx your IP adress. These files can be sent automatically using Menu Debug / Setup and information / Setting up via remote session, where you can select a remote session and path to upload to or delete from.

If the remote machine is connected via a windows network, OptiPerl can also try configuring it via a remote registry connection. See the corresponding tab in the Setup & information window.



See "Debugging via loopback" and "Debugging a script located on your web server" in the tutorials section for a complete example.



Important note when debugging on a remote machine (not important when debugging via loopback)**:**

If you have a dial-up internet connection with a dynamic IP address, before disconnecting from the internet, delete the configuration files sent, by selecting Debug / Setup & information / Setting up using remote session / Delete files. Or delete the files `.perldb` and

`perldb.ini` that were uploaded manually. If you don't, when you terminate your connection to the internet, and the temporary IP you had is assigned to another customer of your ISP, he/she could also create a debugging session with the scipt you were debugging. This also means ability to execute any perl function, and view/edit/delete all files on your web site.

Sendmail and date support

Sendmail is a common program found on most Unix servers, that CGI scripts use to send an e-mail. Most automatic responses and e-mail's you get are sent using sendmail.

Here is a common code snippet for sending mail:

```
open(MAIL,"|/bin/sendmail");
print MAIL <<MAILTEXT;
From: Nick Papadopoulos <nick\@pap.com>
To: $usersemail
Subject: Thank you for becoming a member $userfirstname

Thank you $userfirstname for your interest.
MAILTEXT
```

The external program `/bin/sendmail` might change between servers. Many hosts also create their own version of sendmail for security issues, and might name it like safemail, fastmail etc. Also the path might change like to `/usr/lib/sendmail` or other. If you want to use sendmail, you will have to check where your host has it installed (usually they will have a F.A.Q. page).

Sendmail is neat, but it is a huge task to debug under windows. Until OptiPerl you could not actually check if the e-mail's text is correct. This is because sendmail does not exist under windows.

However OptiPerl can help you debug scripts that use sendmail. It does not actually send the e-mail (and probably while testing you wouldn't want it to), but displays the text that would be sent to sendmail on UNIX.

To use, simply select Run / Sendmail & date support, and enter the path to sendmail your host uses, like for example:

```
/usr/lib/sendmail
/bin/fastmail
```

For the example above we would put:

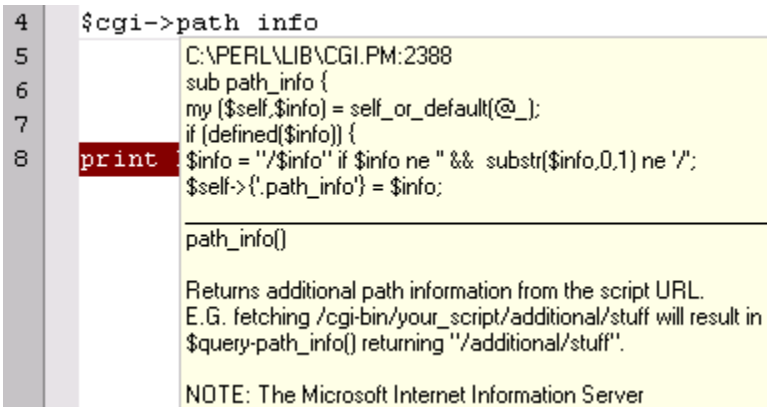
```
/bin/sendmail
```

Then tell optiperl on which drive to copy sendmail. The drive letter should be the same as the script you are debugging, and is usually the C drive.

You are ready. When you run or debug the script, a pop-up should appear with the text that would have been sent. The text is correct if:

- It has a From:, To:, Subject: headers
- It has a line between the headers and text

By pressing "send to MAPI" button on the sendmail pop-up, you can actually send the mail using your default mail program (it will go first to it's outbox before sending). OptiPerl will parse the fields and create the message.



No modifications are needed in your script for this to work.

Using many different sendmail path's

If you test many scripts on different servers, each server might have a different path to sendmail. This is fine. Enter each path in the Sendmail support dialog. Note that because files are copied and directories created, you only need to do this once for each possible path of sendmail. Even if you restart your computer it will still work. To remove sendmail, manually delete the folders c:\bin\ or c:\usr\lib\ (created depending on the sendmail path you entered) with all their contents.

Date support

OptiPerl also supports another common external program found in UNIX, date. Example:

```
print `/bin/date`;
```

This produces something like:

```
Sun Jul 12 16:55:51 EDT 2002
```

To support date, enter it's path (which again varies among servers) in the dialog in menu Run / Sendmail & Date support.

Working with projects

Often for a complete web page, two or more scripts are used, that might also share a common library. For example a script that produces an on-line poll might also have another script for the administrator of the poll.

A project in Optiperl is a set of perl scripts and/or html pages that all belong to the same web site. It is useful to organize many such scripts and html pages if needed in a single project so you can:

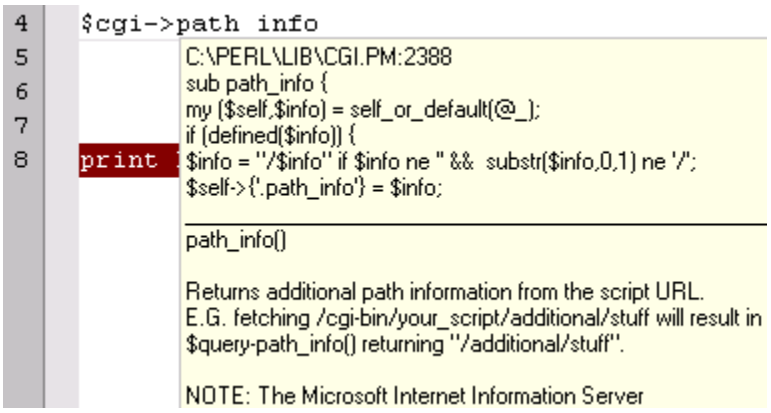
- Load the project in the project window and quickly access the script you want to edit,
- Publish all modified scripts to your web server,
- Save properties that have to do with all the scripts in the project.

Creating a new project

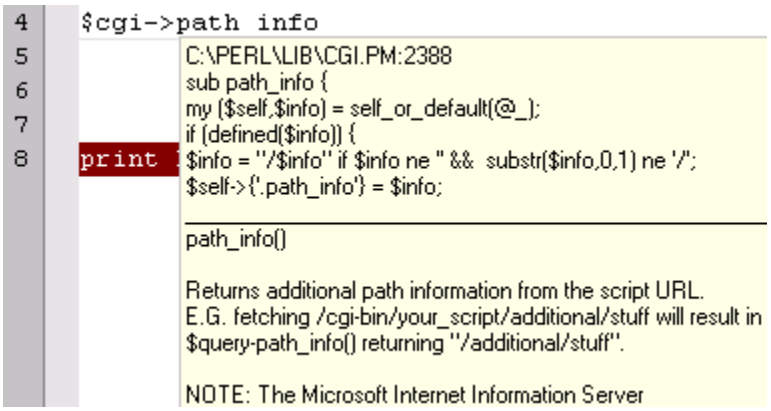
To create a project select from the project menu the item "New". You can add files to the project by selecting "Add" that will add the script that you are currently editing, or "Add Files" to see an open files dialog to add manually one or more files. You can also import an entire folder from "Import folder"

Each file is listed in the Project Window in a tree view. Double clicking a file will open it in the editor. Right-clicking will bring up a pop-up menu to edit the properties for each field.

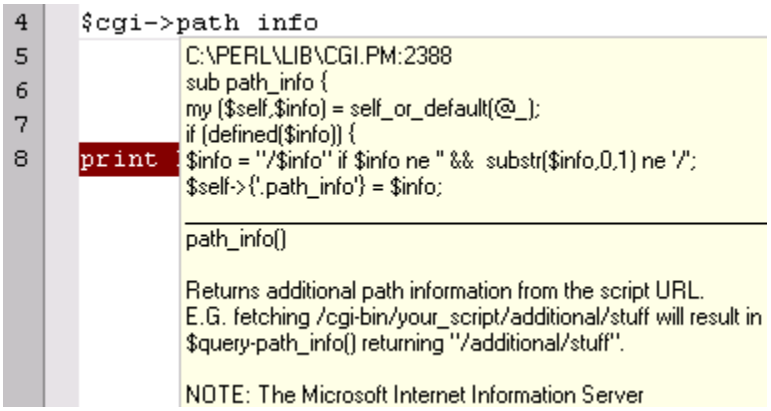
The tree view you see in the project manager should look like the remote folder you upload files to:



To make it look like your remote folder, you will need to follow the directory structure used. If you use any king of html authoring software, then probably you have your entire site offline. Select from Project Options / Local starting directory, the folder that contains all the files and folders of your web site. Add afterwards the scripts you are interested in. You may also add html files that may be necessary to call those scripts when testing so they are easy to access, plus other files used.

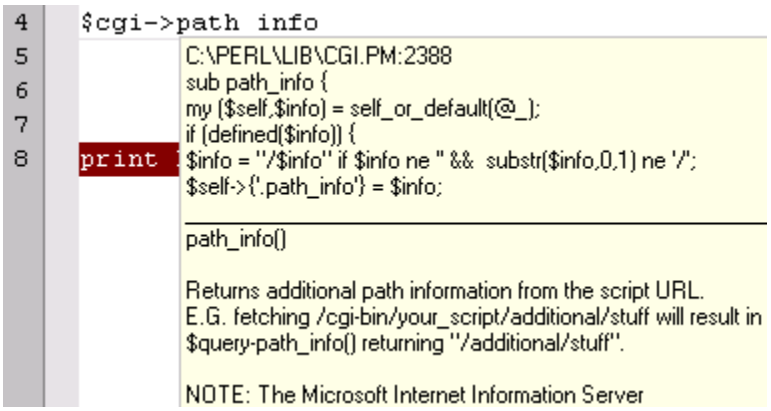


You don't need to add in the project manager every single file of your web site, if you use other software for publishing html files. However if you prefer OptiPerl to publish your entire site, that can be done also.



For larger sites, you may also want to create many projects, each managing a part of the web site. For example one for it's forums, another for customer support scripts, etc.

After specifying the starting folder, you should see all the files in a tree like view, following the directory structure of your server. To publish you will also need to select the corresponding path on the web server from Project Options / "Corresponds to directory on remote server". This is usually " / " but depending on your servers operating system and whether is serves many accounts, your remote path may be different.

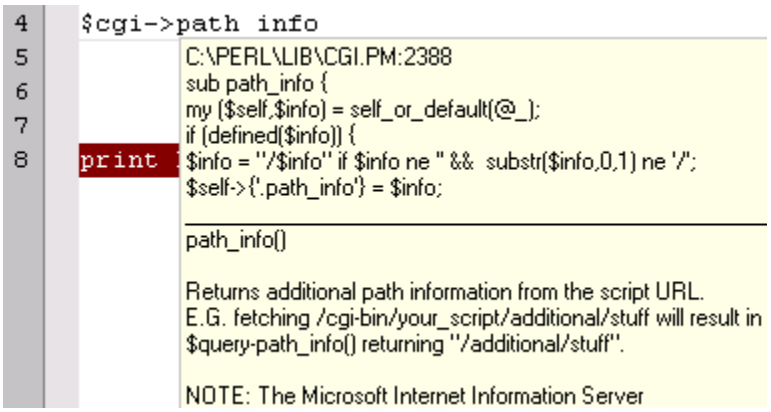


Read the section "Possible scenarios", for some possible real-world configurations.

Project manager window

The project window consists of 6 fields:

- File: The filename of an item in the project
- Mode: The unix permission that will apply when you publish the file.
- Publish: Whether the file will be uploaded to your server.
- Transfer: Transfer mode for the file. For perl scripts and html pages this is "Text"
- Status: After publishing, this field will show the status.



The trasfer field must be set as "text" to enable changing the shebang and converting the version, if enabled in the transfer setup.

The last 4 fields may be changed by right clicking on a file or folder. Note that perl scripts, and the folders that contain them should have a permission of 755. Also upload scripts with "text" transfer.

Publishing the Project

After the remote path and the remote session have been selected, pressing "Publish" will upload all files, and recreate the directory structure of folders do not exist on the remote server. If folders are created, then their permissions as specified are also set.

Publishing is optimized for speed; it will **not** check each remote path of the remote server for files that do not exist in the project and delete them. It will just upload files that were modified since the last publishing from OptiPerl.

If selected in the remote session setup, the uploaded script can also have it's shebang changed and it's version converted to SERVER.

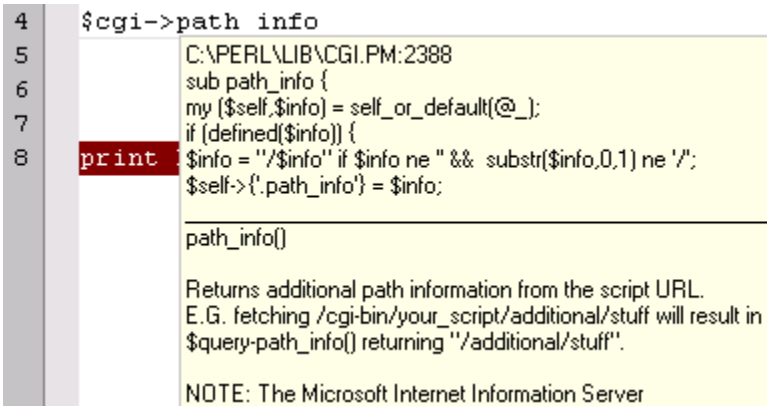
Overriding the destination path of a single local folder

In rare cases you might need to specify the destination path of a *single* folder. Do this by right clicking it and selecting "override destination path". Note that you will need to do the same thing for each of it's subfolders.

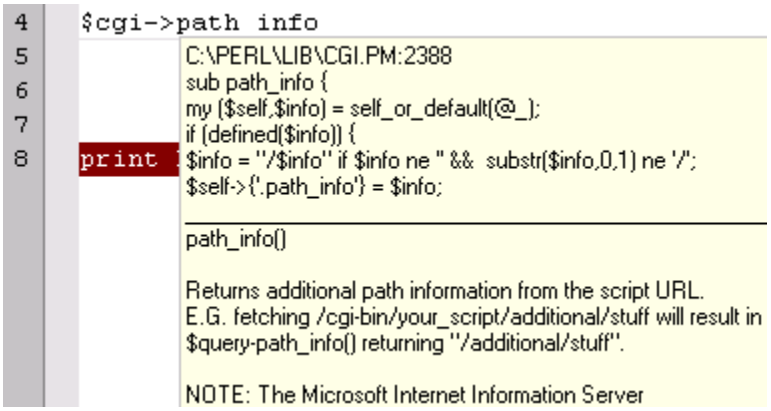
How publishing works

Most synchronization software needs to get a list of all the files of each path and compare their time stamp to the local copy. However because scripts will probably be uploaded very frequently, especially while developing, OptiPerl's publishing does not work this way and has

been optimized for speed. Before uploading it will make the necessary conversions to the script (the shebang and version) and extract the file's CRC value. If the CRC has been changed, then the upload will occur. Keep this in mind if you upload a script with a third-party FTP client; OptiPerl will not know that the file has been already uploaded.



Use publishing; The fact that the shebang gets changed automatically and the version gets converted to SERVER can speed up developing dramatically.



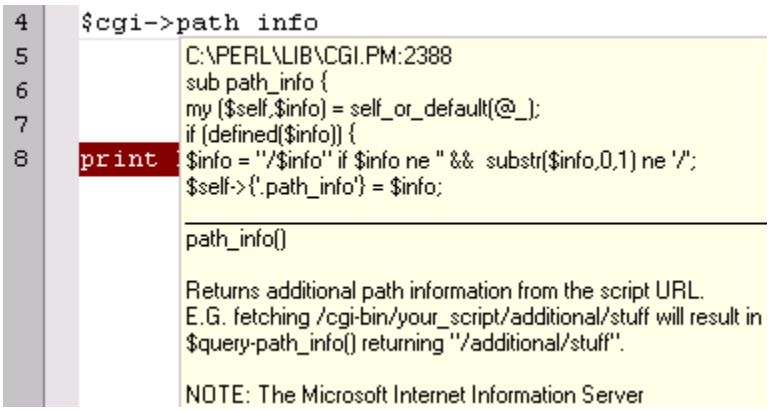
See also "Project Options"

Non-published folder

You can add files out of the local starting directory; They will all be bundled under a folder call "Not published" or in the root of the tree if the option "Display in root" is selected under Project Options / "Files not under starting folder". These files will never be published, unless you change the starting folder to include them also.

Overriding options of the environment per project

Many options selected from menu Tools / Options may need to be overridden at a per project basis. For example, you may want each project to set the root dir of the internal server to it's own setting. Enable this option from Project options / Override default settings, so the rest of the tabs can be enabled.



Before setting up a complicated project with scripts that use absolute paths, check out the section "Using absolute paths".

Project Options

Project Tab

Transfer session
Selected transfer session from the Remote transfer sessions setup window

Local starting directory
Root path of all files and subfolders of the project.

Corresponds to directory on Remote Server
Under what remote path should the project be published

Files not in starting directory
If you add files outside of the starting directory, whether to add them under the "non-published" node. This is purely cosmetic.

Default layout
Layout loaded when loading the project. Select "none" for no modification each time the project is opened.

Settings Tab

Override "settings" and "server" tab options
If selected, all options from the "settings" and "server" tabs will be enabled, and will override corresponding main options.

Run with Server - Host URL
Http address that will be called when the option "Run with Server" is set in the "Server" menu.

Default directories
When opening and saving files, the default folder of the dialog.

@INC
Search path for modules used by code explorer. Read more about how optiperl finds modules.

Parse Lib from project files
Parse all "use Lib (...)" from the project's files. Read more about how optiperl finds modules.

Server Tab - Internal Server

Internal Server Root Path
The root folder of the internal server. If enabled, the file being run must reside in this folder or a subfolder.

Value for PERLDB_OPTS
When the internal server is enabled, and you run a script while listening for a remote debugger, the internal server can run the script adding first in it's environment the setting needed to invoke the debugger. Change this only if you know what you are doing!

Server Tab - External Server

Access - Error Log
The access and Error log file of the external server.

Document Root
The document root path of the external server. This is usually the path that contains the html documents of the site, like htdocs or public_html

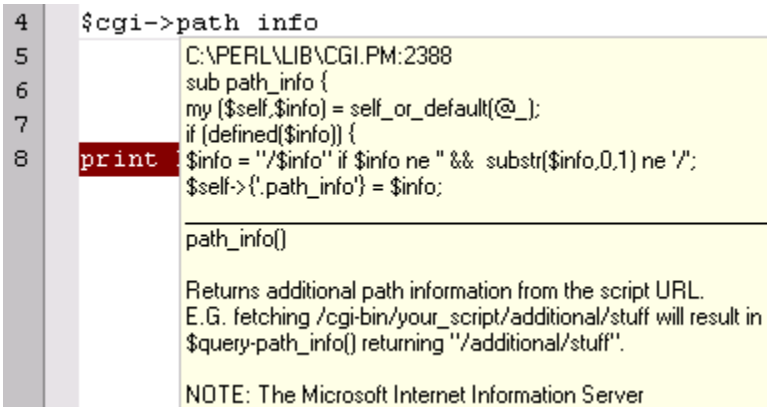
Data Tab

Data fields used for the %data0%..%data9% tags in user tools. Read more about user tools.

Read more about projects in "Working with projects".

Possible scenarios

Below you will find a list of common (and uncommon!) server configurations, and how to re-create and test them offline using OptiPerl. For the examples below we are using c:\webs\ as our local folder. You can of course change this to anything you want.



Scenario 1

Your remote server's path looks like this:

```
/index.html
/cgi-bin/
/cgi-bin/test.pl
```

When requesting

- `http://www.mysite.com/index.html` you get `/index.html`
- `http://www.mysite.com/cgi-bin/test.pl you get /cgi-bin/test.pl`

This is the easiest setup.

Your offline folder will look like

```
c:\webs\mysite\index.html
c:\webs\mysite\cgi-bin
c:\webs\mysite\cgi-bin\test.pl
```

Project Options

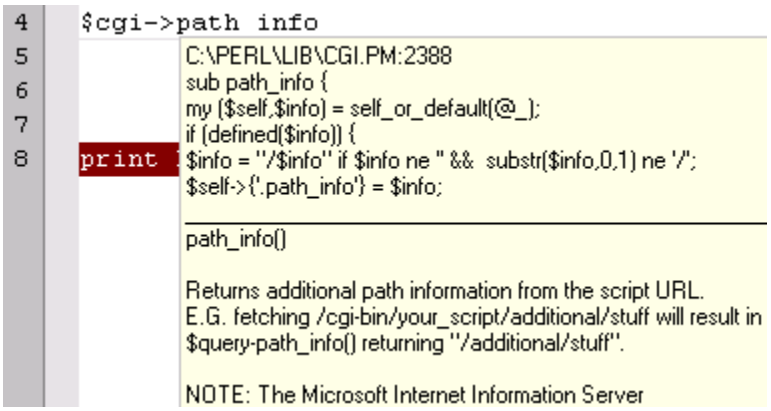
Local starting directory: `c:\webs\mysite\`
Corresponds to: `/`

Project Options / Internal server

Root path: `c:\webs\mysite`

Transfer Sessions / Remote running

Document Root: `/`
Links to: `http://www.mysite.com/`



Scenario 2

Your remote server is apache with mod_perl 😎 and looks like this:

/htdocs/index.html
/cgi-bin/test.cgi
/perl/fastscript.pl

When requesting

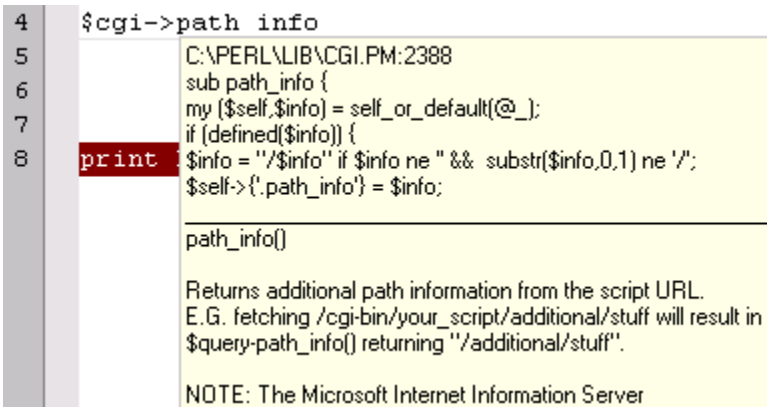
- `http://www.mysite.com/index.html` you get `/htdocs/index.html`
- `http://www.mysite.com/cgi-bin/test.cgi` you get `/cgi-bin/test.pl`
- `http://www.mysite.com/perl/fastscript.pl` you get `/perl/fastscript.pl`

Your offline folder will look like
`c:\webs\mysite\htdocs\index.html`
`c:\webs\mysite\cgi-bin\test.cgi`
`c:\webs\mysite\perl\fastscript.pl`

Project Options
Local starting directory: `c:\webs\mysite\`
Corresponds to: `/`

Project Options / Internal server
Root path: `c:\webs\mysite\htdocs`
Aliases: `/cgi-bin/=c:\webs\mysite\cgi-bin\; /perl/=c:\webs\mysite\perl\;`

FTP Sessions / Remote running
Document Root: `/htdocs/`
Links to: `http://www.mysite.com/`
Aliases: `/cgi-bin/=http://www.mysite.com/cgi-bin/;`
`/perl/=http://www.mysite.com/perl/;`



Scenario 3

Your remote server looks like this:

/users/students/disk1/george/public_html/index.html

Also after some begging of the sysadmin, he allows you to upload perl scripts to /usr/local/apache/cgi-bin/george/

When requesting

- `http://www.site.edu/~george/index.html` you get `/users/students/disk1/george/public_html/index.html`

- `http://www.site.edu/~george/cgi-bin/test.pl` you get `/usr/local/apache/cgi-bin/george/test.pl`

Setup Option A

Your offline folder will look like
`c:\webs\mysite\index.html`
`c:\webs\mysite\cgi-bin\test.cgi`

Project Options
Local starting directory: `c:\webs\mysite\`
Corresponds to: /users/students/disk1/george/public_html/
** read below about the cgi-bin folder when uploading the project.*

Project Options / Internal server
Root path: `c:\webs\mysite\`

Transfer Sessions / Remote running
Document Root: /users/students/disk1/george/public_html/
Links to: `http://www.site.edu/~george/`
Aliases: `/usr/local/apache/cgi-bin/george/=http://www.site.edu/~george/cgi-bin/;`

* Note that the cgi-bin must be uploaded to a completely different path, and not under the starting directory. So select the cgi path in the project manager, right click and select

"Override destination path". In the dialog box, enter `/usr/local/apache/cgi-bin/george/`. This needs to be done only once.

Also note that this setup will create problems if the cgi scripts needs to access files from the public_html folder. You will need to use the version converter to change the lines in the script for the Local and Server version. There is however another way to set this project up so this wont be a problem:

Setup Option B

Your offline folder will look like
`c:\webs\mysite\users\students\disk1\george\public_html\index.html`
`c:\webs\mysite\usr\local\apache\cgi-bin\george\test.pl`

Project Options
Local starting directory: `c:\webs\mysite\`
Corresponds to: /

Project Options / Internal server
Root path: `c:\webs\mysite\users\students\disk1\george\public_html\`
Aliases: `/cgi-bin/=c:\webs\mysite\usr\local\apache\cgi-bin\george;`

Transfer Sessions / Remote running
Document Root: /users/students/disk1/george/public_html/
Links to: `http://www.site.edu/~george/`
Aliases: `/usr/local/apache/cgi-bin/george/=http://www.site.edu/~george/cgi-bin/;`

Using this setup, if you need to access files of the public_html folder from the cgi-script and don't want to use the version converter, you can access them using relative paths. For example, for test.pl to access index.html, it would use:

`../../../../../users/students/disk1/george/public_html/index.html`

This would work on both window and unix.

Introduction & examples

Optiperl takes searching and replacing very seriously. It has many powerful tools to search and / or replace text in your project or entire folders, with or without prompting.

Optiperl has the following tools to search and replace:

- Standard find dialog at menu Search / Find
- Search again (for the next match after the cursor position) using the last pattern in the above dialog at menu Search / Find again
- Regular expression search / replace at menu Search / Advanced Search & Replace
- Regular expression search / replace in the project at menu Project / Search & Replace
- Regular expression search / replace in the File Explorer that can search in selected files or folders by pressing the "Find" button.

Examples using Regular expression search / replace

Adding a comment before each subroutine containing the subroutines name and normalizing the spacing between subroutines.

Find: `}[\n\s\t]+sub +(\w+)`
Replace: `}\n\n#####\n# Subroutine: \1\n#####\nsub \1`

After changing what a heavily used subroutine parameters should be, you need to add the parameter "1" on all the instances of calling the subroutine (the subroutine is called "mysub"):

Find: `([\s\t]*mysub\s*\([^)]+)\)`
Replace: `\1,"1")`

Standard search and replace dialog

With the standard find & replace dialog you can look and replace for simple text in the file being edited. While the dialog is open, you can also edit the script. Selecting the command at menu Search / Find again, will find the next match after the cursor position.

Case sensitive
Whether the case of the text to find will matter while matching

Whole words only
Whether the text to find will be a part of a word or must be a whole word

Regular expression
Enables very simple metacharacters for the find text, like

`$ ^ + ? .`

It also enables the replacement text to contain the metacharacters:

`\n` New line
`\t` Tab

Do not enter more complex regular expressions here. The Pattern search / replace dialog is better and has more functionality.

Prompt on replace
If you have enabled replace, then you will be prompted for each replacement.

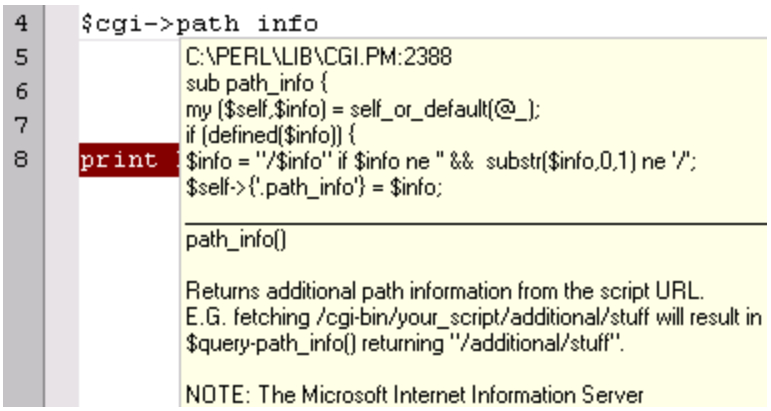
Direction
Search text either in a forward direction or backwards.

Scope
Whether to confine the search in the selected block or the entire file

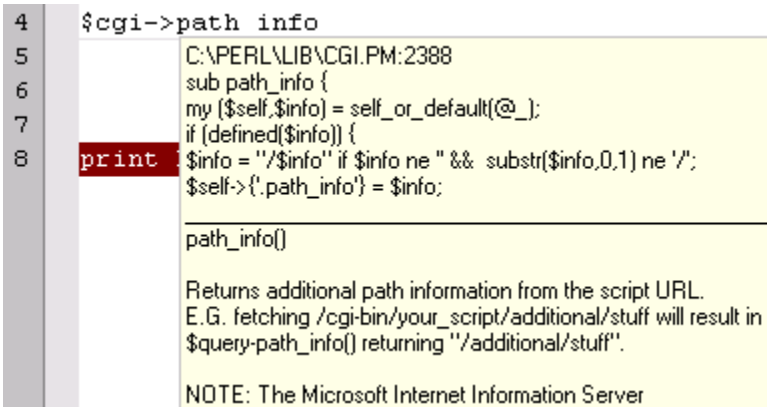
Origin
Begin the search from the cursor position or from either the selected block or the entire file (depending on scope)

Search / Replace in projects and files

The search / replace pattern dialog is a powerful tool that can search for standard regular expressions and replace them even with backreferences, new lines and tabs.



Regular expression can be unexpected sometimes. Note also that a pattern is applied to the entire file, not just line by line. So if you are replacing and do not have the option "open in editor" enabled, we recommend backing up the affected files first.



Read about what you can do at Examples.

Find box
Enter here a standard regular expression to search for.

Case sensitive
Whether the case of the pattern to find will matter while matching

Ungreedy
[from perlre.pod]
By default a quantified subpattern is "greedy", that is, it will match as many times as possible (given a particular starting location) while still allowing the rest of the pattern to match. If you want it to match the minimum number of times possible, select this option. Note that the meanings don't change, just the "greediness".

Replace box
Enter here text to replace with.

- Backreferences are noted with \1, \2 ... \9 etc.

- New line is \n
- Tab character is \t

Prompt on replace
If you have enabled replace, then you will be prompted for each replacement.

Open files in editor; Don't save
If you have deselected Prompt on replace, then the files replaced will not be saved, but opened in the editor for further review or canceling changes.

Detect & keep EOL
An important option that should be enabled at all times. If you want to search for new lines (\n) then depending on the file searched, the search pattern would have to be different. For example

- Files with window style line endings would require \r\n
- Files with UNIX style line endings would need \n
- Files with Mac style line endings would need \r

If you would where also replacing then you would also need the according pattern in the replace box.

However if you have this option enabled, then you do not have to worry about the EOL characters in the file you happen to be searching. In both the Find and Replace box, "\n" means end of line.

File mask box

The file mask box is shown only when the Pattern find dialog invoked to search in a project of files in File Explorer, and can limit the files to search using wildcards. Note that you cannot select specific files from a project to search on, only limit the files search using a wildcard. If you want to search in specific files in a project, then open the File Explorer and select the files you want.

Recursive search
If you have selected folders in the File explorer, then selecting this options will search in subfolders also of the folder.

Search in binary files
When selected then files that are binary will be searched also for the regular expression you entered. For example, try searching in all *.exe files in your windows folder for e-mails. The results will be shown in the code explorer that can handle display of binary files.

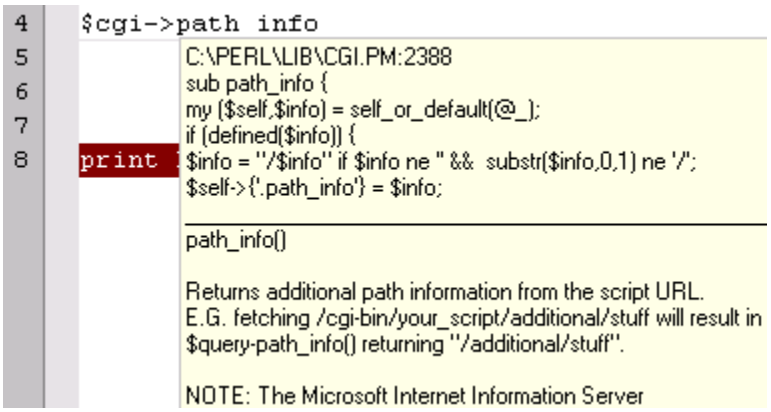
Search / Replace with code explorer

You can change the names of variables using the code explorer. Select a variable under the Scalars, Arrays or Hashes node, and left-click once to rename. Enter the new name. Afterwards a search and replace will be made automatically on the script and the code explorer will be updated again.

Search using the subroutine list

The subroutine list at menu Search / Sub list is a simple dialog containing all subroutines in the edited script, and is used for navigation. Entering a part of the subroutine's name will narrow the search in the sub list, where you can then select the sub using the up / down arrow keys and Enter. The editor will focus on the subroutines declaration.

You can also *copy* the subroutines name into the clipboard and exit by pressing Ctrl-Enter, *without* changing were the cursor position was. This is helpful if you were trying to remember how the subroutine was called.



Learn to use and get in the habit of using the subroutine list. It will increase your productivity by 100% guaranteed!

Backups

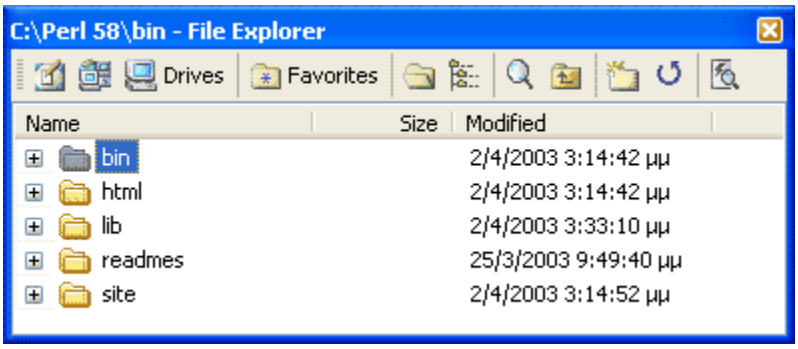
OptiPerl can create backups of files. Backups can be saved to a special folder, or compressed into a zip archive, or both. You can also name the resulting files using tags that can include date and time. For more information and examples, see the section backups in Options Dialog.

File Explorer

The file explorer can be used to browse a folder and open files. Right clicking on a folder or file will open a standard explorer menu.

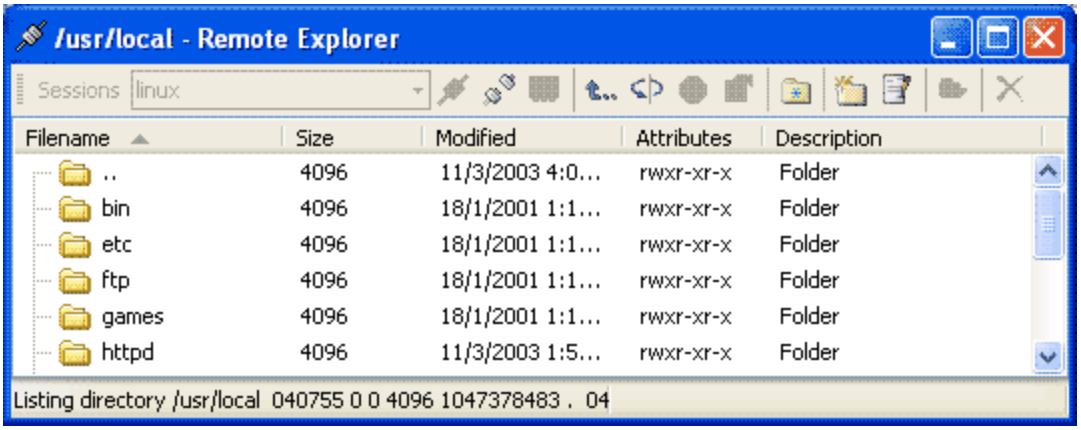
Double-clicking a file associated with OptiPerl will open it.

You can also select a set of folder and files and do a pattern search / replace on them. The results of the search or logging of the replacements will show in the code explorer window. To open the dialog, press the find button (rightmost button).



Remote Explorer

Like the file explore, the Remote explorer is an explore type viewer of the files in remote sesions (that have been selected from Tools / Remote sessions setup)



Also ability for basic file management is provided. You can also download files from this window. To open however in the editor, use the right click menu and select "Open in editor".

Code Librarian

With the code librarian you can store snippets of code, for use in your programs. You can:

- Organize the snippets of code using categories and sub categories utilizing the tree-like view.
- Share code between your group.

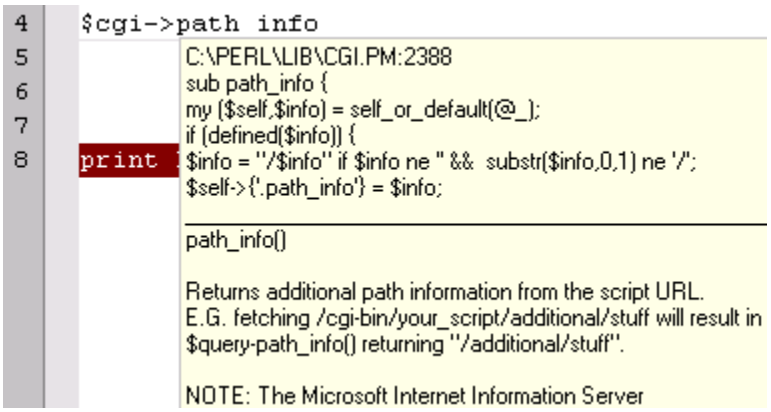
Starting from version 4, OptiPerl uses a standard ZIP format to store all code snippets. Actually code snippets represent files in the zip archive. However the extracting and packing is so integrated, it does not look like an actual zip file is being used!

When you select a code snippet in the tree structure (that actually is folders and subfolders in the archive), it is extracted and displayed in the editor. If you modify the text, it will be compressed again. This happens automatically when moving to another node, or just closing the dialog (this is why there is no "save" button).

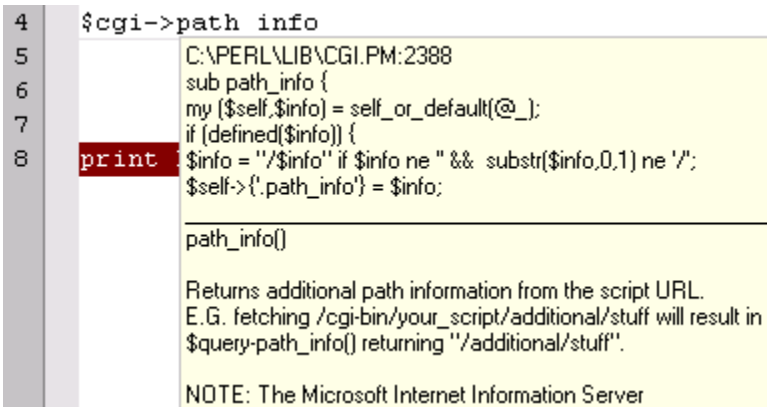
To add a new item, press "Add Item". A new item will be inserted below the focused item with the title "New snippet". You can change it's name by clicking the name once with the mouse.

"Add sub item" works like above, but creates a new tree-level under the focused item. Using then "Add Item" you can enter code snippets under the new category.

The tree structure fully implements drag and drop. You can drag files or folders and place them under other folder. To target another folder, release the mouse button on the folder you want. To target the root of code librarian, release over the top most item in the tree.

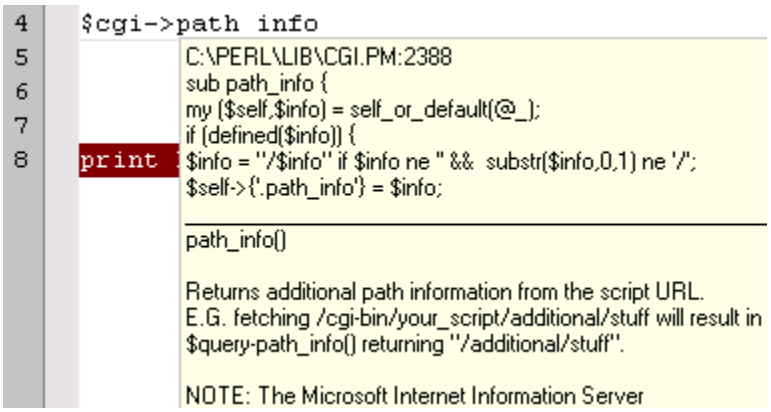


Note that because you are actually viewing a directory structure, all rules that apply to windows filenames should be followed: Don't use the characters < > / \ : * ? and don't use duplicate names.



Extensions are not shown in the tree view, but icons that represent the file type are used. Also from the file extension, the corresponding syntax parser will be used in the editor. To change the extension, rename the file adding the extension you want, for example `head.html` or `string.pl`. If you don't put a dot at the end, the extension will not change.

Code snippets do not have to be actual programs that run, but giving an extension helps add the correct icon and also enables the correct syntax parser to be used in code librarian's editor.



Code librarians snippets are saved in "`Code Librarian.zip`" in OptiPerl's Application Data folder.

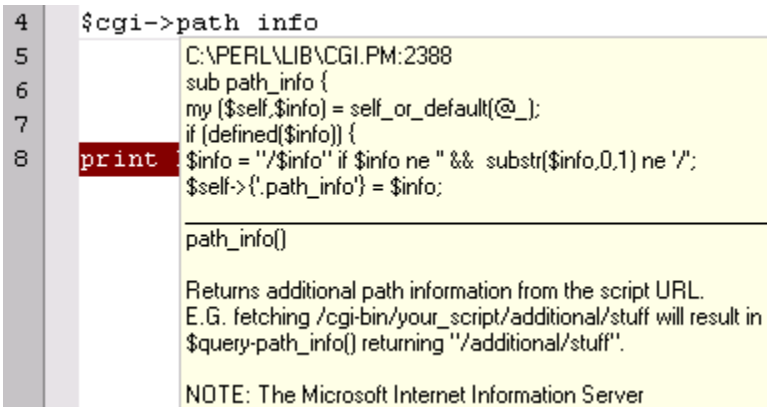
Opening Zip files in general

You can also open any zip file from Tools / Open zip file. The rules that apply to code librarian will also apply here, and you can actually open many zip files plus code librarian in separate windows to drag and drop files.

How to import to Code librarian

Nothing to it! Just collect your favorite files with examples of code, put them in a folder, and use your favorite Zip program (like for example WinZip (r))to add them to Code Librarian.zip

You can import the old OPL format of OptiPerl from the File menu of code librarian.

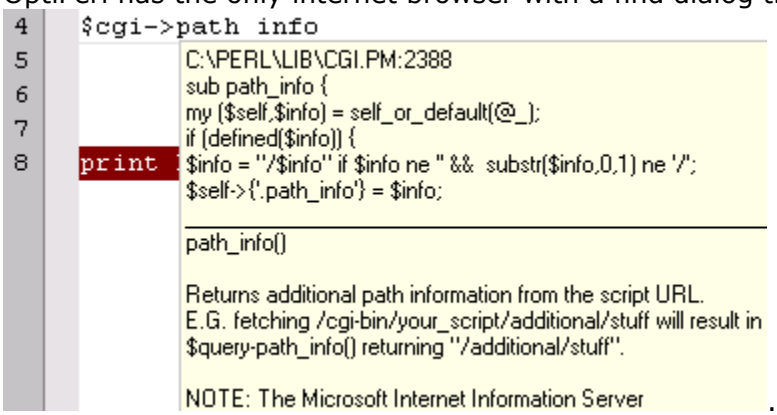


When opening zip files, remember that editing in the text box will actually replace the old file, by packing over it.

Pod Extractor

Pod extractor enables you to view POD files, or embedded pod information in a script. When selecting Pod Extractor, the window is opened with the filename of the script you are editing to be opened. Press "Extract" or select first a different file by pressing the Open File button.

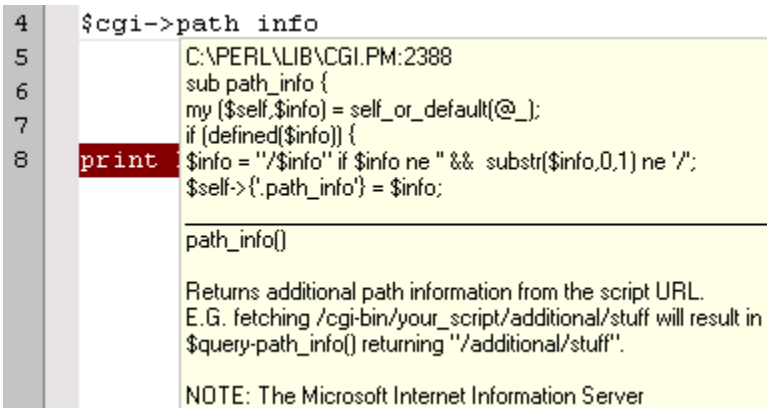
OptiPerl has the only internet browser with a find dialog that supports regular expressions



. To invoke the dialog, press "Ctrl-F"

Another useful feature is creating a list of all files in your perl directory that contain pod information. Select "Files" and then enter a folder to search in. Recommended is \perl or \perl\lib. This will list all the files in the combo box permanently so you can select them easily. You can refine the search any time later by pressing "Files" again.

Pod extractor can also be invoked by right clicking on a used or required module in Code explorer.



You can change the style sheet of the output by modifying the file "`Pod2Html.css`"

Encoder

With this dialog you can Encode and Decode strings with the following methods:

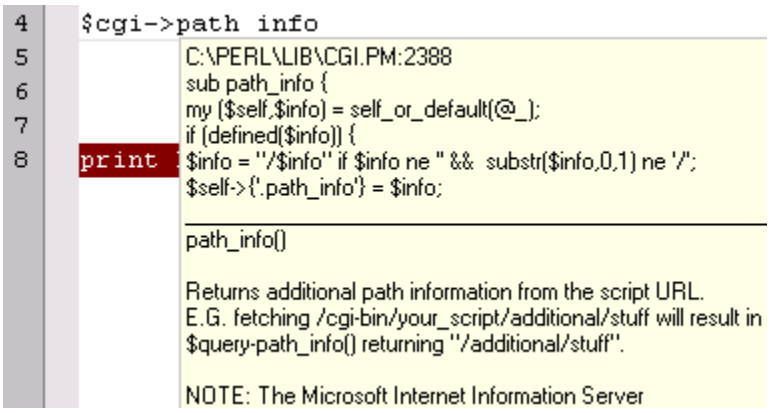
URL: Encode string characters using the hex escape syntax of RFC1738. Non-alphanumeric characters other than period, dash and underline are replaced by '%' followed by a 2-digit ASCII hex code.

Base64: base64 encoding compatible with Internet protocol RFC 1521 (the MIME standard). Data encoded in this manner is safe for Internet, database and mainframe storage and use. By definition, the resultant string is at least 1/3 longer than the original

UUEncoded: Encode binary Source string using Unix-to-Unix encoding. Encoding increases the overall length of string by 1/3. This encoding technique has never been formalized and as a result, a number of different variations exist.

Crypt: Encrypts a string exactly like the crypt function in perl. The salt used is random (you can only encode, decode will not function).

Hexadecimal - Binary: These convert decimal numbers to hex and binary. In the top editor enter a decimal number, and expect the conversion to be made in the bottom. By changing the bottom editor, the corresponding decimal will appear in the top editor.



How to work fast with the encoder: Select some text in the main or other editor, and the open the encoder, or press Ctrl-I (default shortcut). This will copy the selected text so it can be encoded. After selecting the method, press the replace button to make the replacement and return to the last editor.

Perl Printer

Perl printer is a two-panel window to convert raw text to print statements.

Enter the text to be outputted in the left editor, or open a file, and the text is converted to legal print statements.

You can select whether the print statements will be a "here-document" format, or select a start and end command and quote for each line.

Regular Expression Tester

With the Regular Expression Tester, you can enter regular expression and test quickly the way they match, substitute and transliterate text.

There is also an Explain box, that uses Jeff Pinyan's YAPE::Regex::Explain module to analyze the pattern of the regular expression. This module is included with OptiPerl's distribution.

Enter some text in the edit box and some input for processing in the left panel. Updating takes place automatically.

In the explain list box, moving through the tokens also highlights in the edit box the corresponding part of the regular expression that is explained.

Here are some examples that you can enter. After entering, type some corresponding text in the left panel to get the results.

```
/(ftp|http):\/\/([_a-z\d\-]+(\.[_a-z\d\-]+)+)(([_a-z\d\-\\\.\/]+[_a-z\d\-\\
\/])+)*/
```

Check for URL addresses.

```
/\b([_a-z0-9-]+(\.[_a-z0-9-]+)*)@([_a-z0-9-]+(\.[_a-z0-9-]+)*)\.([a-z]{2,3})
\b/
```

Very simple check for e-mail address, not recommended.

```
/(\d{3,4})[- ]?(\d{4})[- ]?(\d{4})[- ]?(\d{4})/
```

Credit card number check

```
/\b([01]?\d\d?|2[0-4]\d|25[0-5])\.([01]?\d\d?|2[0-4]\d|25[0-5])\.([01]?\d\d?|
2[0-4]\d|25[0-5])\.([01]?\d\d?|2[0-4]\d|25[0-5])\b/
```

Check for IP addresses like 187.224.98.4 or 1.2.3.4, consist of four bytes separated by dots.

```
/\b\w{3,}a\b/
```

Word at least 4 letters long, ending with an a

```
/\A-?(?(?=[1-9])\d*)\z/
```

Signed or unsigned number, not starting with 0.

```
tr/A-Z/a-z/
```

Convert all uppercase to lowercase

```
m/(\d{3})-(\d{3})-(\d{4})/
```

Check input if it is a valid U.S. telephone number.

```
s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg
```

This will unpack a URL Encoded String.

Important: You must include a starting and ending slash in the pattern you type. Entering a raw pattern does not work.

Regular Expression Tester is a program in it self! It is designed to quickly test implementations of regular expressions without endless rounds of editing-fixing-running.

Perl Tidy interface

Perl tidy is an interface to Steven L. Hancock's perltidy (included with OptiPerl's distribution). His excellent perl script can be used to reformat perl code to make it more readable. The bottom part of the window shows a preview of your script, after the process. By pressing save, you can save your changes.

It is highly recommended to have the "Backup" option enabled before pressing save.

File Compare

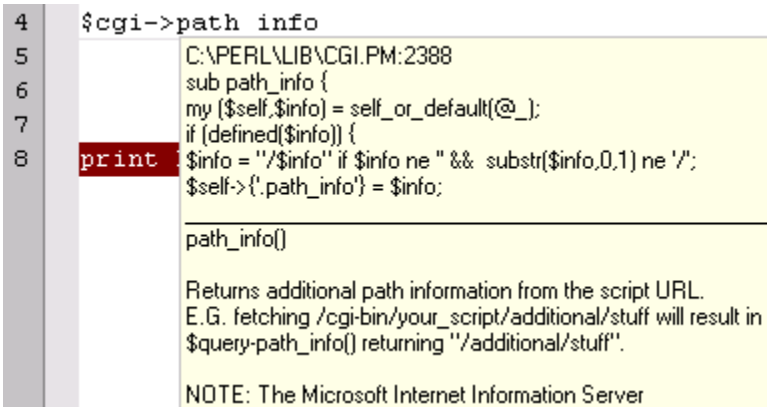
The file compare tools is handy to view two file's differences. Press the buttons to select the files you want and then the compare button to start. The first file by default will be the active file in the editor.

Auto view

The auto view window is a file viewer that updates each time the target file is modified. It can be used to monitor a file that your script uses. For log files however you can use the items Server / Error & Access logs.

About plug-ins

Every programmer will need at some point to automate tasks using his own personal standards. OptiPerl includes an OLE automation interface so you can control the editor using a high level language (like perl) using Object Linking & Embedding.



In the folder you installed optiperl, check out the "plug-ins" folder. This is where all perl and other plug-ins are stored. Many examples are included showing how to use plug-ins. Also check out the web page:

http://www.xarka.com/optiperl/plugins/

Using Plug-Ins

Below is a basic template for a plug-in:

```
#Name: Plug-in Name
#Description: Plug-in Description
#Icon: %opti%Tools.icl,148
#Button: Editor / Main menu
use Win32::OLE;
sub Initialization
{
 $PlugID = $_[0];
 #Get plug id. Will be used later
 $optiperl =  Win32::OLE->new('OptiPerl.Application');
 #get reference to application
 $optiperl->OutputAddLine("Hello World!");
 $optiperl->MessageBox("My Plug-In","Hello World!",0);
 $optiperl->EndPlugIn($PlugID);
}
sub Finalization
{
}
if (! defined $valid_plugin)
{
 #enter here the subroutines you want to test.
 Initialization;
}
```

The first 4 lines with comments are required and parsed for a definition of the plug-in. The lines are name, description, an icon that will be used as a button, and where to add the

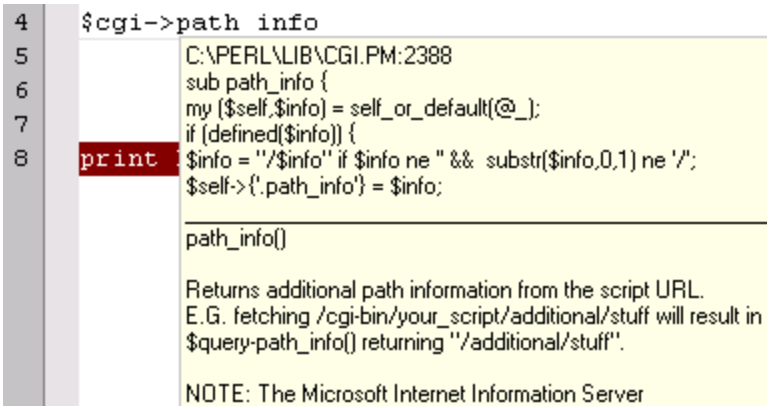
button. Here `Editor / Main menu` means that the button will be placed in the editor window, in the main menu (the toolbar over the editor window).

Also note the **`Initialization`** and **`Finalization`** subroutines. These are required in every plug-in. Because all code is event-driven, there is no need to place code outside of subroutines. The last line is for debugging purposes only; the value **`$`**`valid_plugin` is "magic" and defined only when the plug-in is run via the "Start/Stop" plug-in menu.

To run this plug-in, save in the \plug-in folder, and press once Tools / Start Stop Plug-ins / Update.

Plug-ins using Tk

Special considerations have been made when using Tk to create a plugIn. See the example below as a template for creating a window from Tk and docking it into optiperl.



For a more complex example, see the `PlugInDemo.pl` in the \plug-ins folder.

```
#Name: PlugIn Tk Demo
#Description: PlugIn Tk Demo
#Icon: %opti%Tools.icl,146
#Extensions: NewProcess
use Tk;
use Win32::OLE;

sub Initialization {
 $plug_id = $_[0];
 #The first parameter is a handle to the plug-in. We will need this
 #for future commands

 $optiperl = Win32::OLE->new('OptiPerl.Application');
 #Initialize OLE object

 $main = MainWindow->new;
 #Initialize TK toplevel window

 $main->Label(-text => 'OptiPerl Plug-in Demo')->pack;
 #Add a label

 $main->OnDestroy(sub{Finalization()});
 #IMPORTANT: Add an event handler when TK destroys itself to call
 #the Finalization subroutine.

 $win = $optiperl->RequestWindow($plug_id);
 #Request a dockable window in OptiPerl

 if (defined $win) {
  $win->Show;
  #Show the window

  $win->{Title} = "Plug-in Demo";
  #set the windows title

  ProcessEvents();
```

```
  #"Magic" subroutine in plugins to force TK to process
  #all windows events to create the window.

  $optiperl->DockWindow($plug_id, hex $main->winfo("id"), $win);
  #Dock TK's toplevel window into OptiPerl

  # The following code creates our menu system and toolbars

  my $btn = $optiperl->CreateToolItem($plug_id,"button");
  # set the buttons options.
  # SetOptions(Caption, Hint, Image, Shortcut, On Click subroutine name)
  $btn->SetOptions(
      "Button Test","A Button Test","%opti%tools.icl,117",
      "Alt+F5","ButtonTestEvent");
  # Note that setoptions is a fast way to set all the settings of
  # a tool item. We could also do:
  # $btn->{Enabled}=1;
  # $btn->{Hint}="test";
  # $btn->{Caption}="caption";
  # $btn->{Image}="cool.dll,100";
  # $btn->{Visible}=1;
  # $btn->{Shortcut}="Ctrl+Shift+F1";
  # $btn->{OnClick}="SubName";
  $win->MainBarLinks->Add($btn,0);
  #add the item to the menu of the window
  $win->PopUpLinks->Add($btn,0);
  #add the item to the pop-up menu (when the arrow is pressed)
  $optiperl->ToolBarLinks($plug_id)->Add($btn,0);
  #add the menu to the main toolbar assigned to this plugin
  #(the one created next to the main menu)
  $optiperl->UpdateToolBars($plug_id);
  #Necessary! An error will occur if the above is not called after
  #messing around with the toolbars.
  $optiperl->ToolBarVisible($plug_id,1);
  $win->ReDraw;
 }
}

sub ButtonTestEvent {
 my $handle = $optiperl->Handle;
 my $string = $optiperl->InputBox('Plug in','Enter Text:','Default');
 $optiperl->OutputAddLine($string);
 $main->Label(-text => $string)->pack;
}

sub Finalization {
 $optiperl->DestroyWindow($plug_id,$win);
 $win->Hide;
 $optiperl->EndPlugIn($plug_id);
}

if (! defined $valid_plugin)
{
 sub ProcessEvents {};
 Initialization;
 MainLoop;
}
```

Application Class
Methods

CloseDocument
Declaration: CloseDocument

CreateToolItem
Declaration: CreateToolItem(Process as long,ClassName as String) as Variant

DestroyWindow
Declaration: DestroyWindow(Process as long,Window as IWindow)

DockWindow
Declaration: DockWindow(Process as long,Handle as long,Parent as IWindow)

EndPlugIn
Declaration: EndPlugIn(Process as long)

ExecuteAction
Declaration: ExecuteAction(Action as String)

GetColor
Declaration: GetColor(Name as String) as String

GetOpt
Declaration: GetOpt(Name as String) as Variant

GetWindowHandle
Declaration: GetWindowHandle(Window as IWindow) as long

GrabWindow
Declaration: GrabWindow(Process as long,Enable as Boolean,Handle as long)

InputBox
Declaration: InputBox(Caption as String,Prompt as String,Default as String) as String

MessageBox
Declaration: MessageBox(Caption as String,Prompt as String,Flags as long) as long

NewDocument
Declaration: NewDocument(Filename as String) as Variant

OpenDocument
Declaration: OpenDocument(Filename as String) as Variant

OutputAddLine
Declaration: OutputAddLine(Text as String)

OutputClear
Declaration: OutputClear

ProcessMessages
Declaration: ProcessMessages

QuickSave
Declaration: QuickSave

RequestWindow
Declaration: RequestWindow(Process as long) as Variant

SetColor
Declaration: SetColor(Name as String,Value as String)

SetOpt
Declaration: SetOpt(Name as String,Value as Variant)

StatusBarRestore
Declaration: StatusBarRestore

StatusBarText
Declaration: StatusBarText(Text as String)

ToolBarVisible
Declaration: ToolBarVisible(Process as long,Visible as Boolean)

UpdateOptions
Declaration: UpdateOptions(Everything as Boolean)

UpdateToolBars
Declaration: UpdateToolBars(Process as long)

Properties

ActiveDocument
Declaration: ActiveDocument as Variant

CodeExplorer
Declaration: CodeExplorer as Variant

DocumentCount
Declaration: DocumentCount as long

Documents
Declaration: Documents(Index as long) as Variant

EditorControl
Declaration: EditorControl as Variant

FocusedControl
Declaration: FocusedControl as Variant

Handle
Declaration: Handle as long

Project

Declaration: Project as Variant

ToolBarLinks
Declaration: ToolBarLinks(Process as long) as Variant

Windows
Declaration: Windows(Name as String) as Variant

Document Class

Methods

Add
Declaration: Add(Text as String)

CursorDown
Declaration: CursorDown

CursorLeft
Declaration: CursorLeft

CursorRight
Declaration: CursorRight

CursorUp
Declaration: CursorUp

Delete
Declaration: Delete(Index as long)

Insert
Declaration: Insert(Index as long,Text as String)

TempHightlightLine
Declaration: TempHightlightLine(Index as long)

Properties

ColorData
Declaration: ColorData(Index as long) as String

CursorPosX
Declaration: CursorPosX as long

CursorPosY
Declaration: CursorPosY as long

Filename
Declaration: Filename as String

LineCount
Declaration: LineCount as long

Lines
Declaration: Lines(Index as long) as String

Modified
Declaration: Modified as Boolean

Selection
Declaration: Selection as String

Node Class
Properties

Caption
Declaration: Caption as String

Hint
Declaration: Hint as String

ID
Declaration: ID as long

Line
Declaration: Line as long

Path
Declaration: Path as String

Project Class

Methods

AddFile
Declaration: AddFile(Filename as String) as Variant

AddFolder
Declaration: AddFolder(Folder as String,WildCard as String)

GetOpt
Declaration: GetOpt(Name as String) as String

NewProject
Declaration: NewProject(Filename as String)

OpenProject
Declaration: OpenProject(Filename as String)

RemoveFile
Declaration: RemoveFile(Filename as String)

SaveProject
Declaration: SaveProject

SetOpt
Declaration: SetOpt(Name as String,Value as String)

UpdateOptions
Declaration: UpdateOptions

Properties

Count
Declaration: Count as long

Filename
Declaration: Filename as String

Items
Declaration: Items(Index as long) as Variant

Modified
Declaration: Modified as Boolean

SelectedItem
Declaration: SelectedItem as Variant

ProjectItem Class

Properties

Filename
Declaration: Filename as String

Mode
Declaration: Mode as long

Publish
Declaration: Publish as Boolean

PublishTo
Declaration: PublishTo as String

Text
Declaration: Text as Boolean

ToolItem Class

Methods

SetOptions
Declaration: SetOptions(Caption as String,Hint as String,Image as String,Shortcut as String,OnClick as String)

Properties

Caption
Declaration: Caption as String

Enabled
Declaration: Enabled as Boolean

Hint
Declaration: Hint as String

Image
Declaration: Image as String

OnClick
Declaration: OnClick as String

Shortcut
Declaration: Shortcut as String

ToolLinks
Declaration: ToolLinks as Variant

Visible
Declaration: Visible as Boolean

ToolLinks Class

Methods

Add
Declaration: Add(ToolItem as Variant,BeginGroup as Boolean)

AssignLinks
Declaration: AssignLinks(SourceLinks as Variant)

Properties

Count
Declaration: Count as long

Items
Declaration: Items(Index as long) as Variant

TreeView Class

Methods

AddNode
Declaration: AddNode(ParentNode as INode) as Variant

DeleteChildren
Declaration: DeleteChildren(Node as INode)

DeleteNode
Declaration: DeleteNode(Node as INode)

GetFirst
Declaration: GetFirst as Variant

GetFirstChild
Declaration: GetFirstChild(Node as INode) as Variant

GetNext
Declaration: GetNext(Node as INode,Sibling as Boolean) as Variant

Node
Declaration: Node(ID as long) as Variant

Properties

MainNode
Declaration: MainNode(Index as long) as Variant

RootNode
Declaration: RootNode as Variant

Window Class

Methods

Hide
Declaration: Hide

Redraw
Declaration: Redraw

Show
Declaration: Show

Properties

Handle
Declaration: Handle as long

MainBarLinks
Declaration: MainBarLinks as Variant

PopUpLinks
Declaration: PopUpLinks as Variant

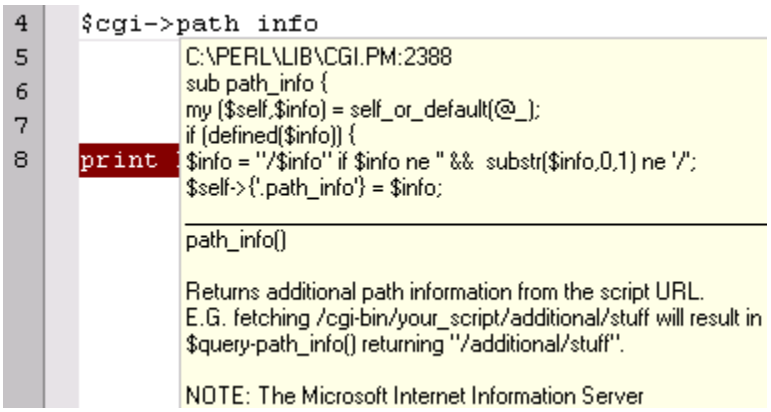
Title
Declaration: Title as String

About user tools

User tools are external programs that can be setup to run using custom parameters. If console programs are used (including perl scripts), you can also send and receive to standard input & output.

Parameters used can be changed dynamically, by replacing special tags listed in the next section.

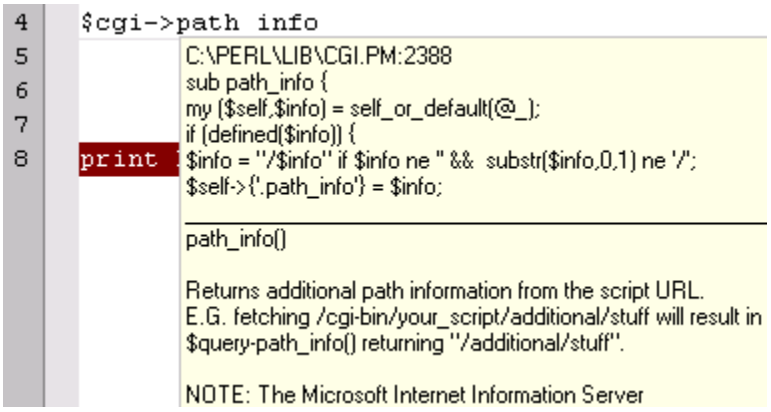
You can also move your tools to toolbars or the menu, assign shortcuts and even a corresponding graphic.



For more advanced manipulating of optiperl read about plug-ins instead.

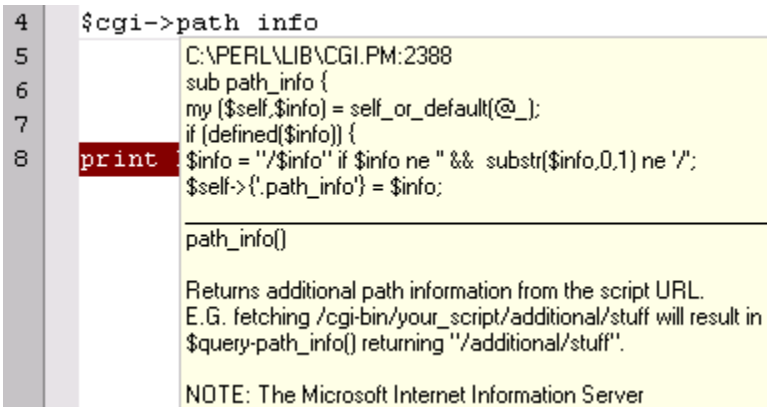
Why you would want to use tools - Some examples

Examples of non perl tools:



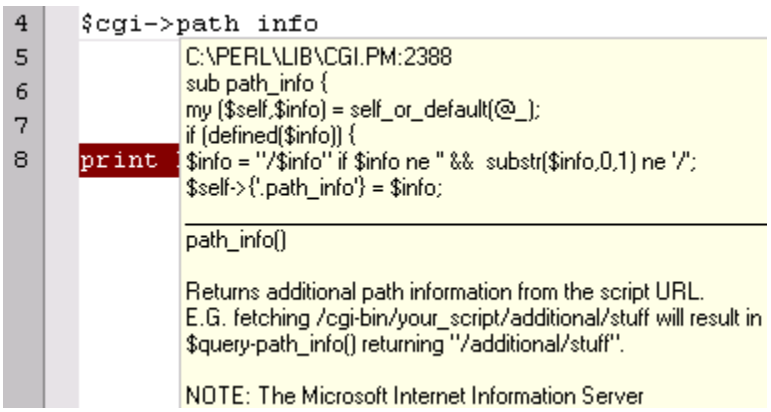
Opening an explorer window to c:\perl

Program: `Explorer.exe`
Parameters: `c:\perl`



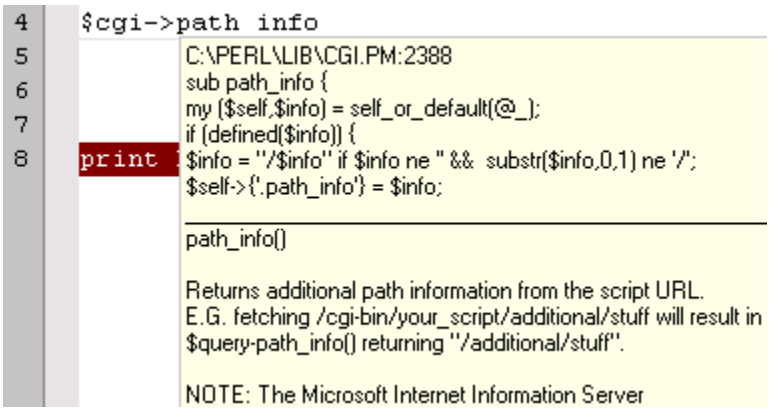
Opening perl's main help file:

Program: `Explorer.exe`
Parameters: `c:\perl\html\index.html`



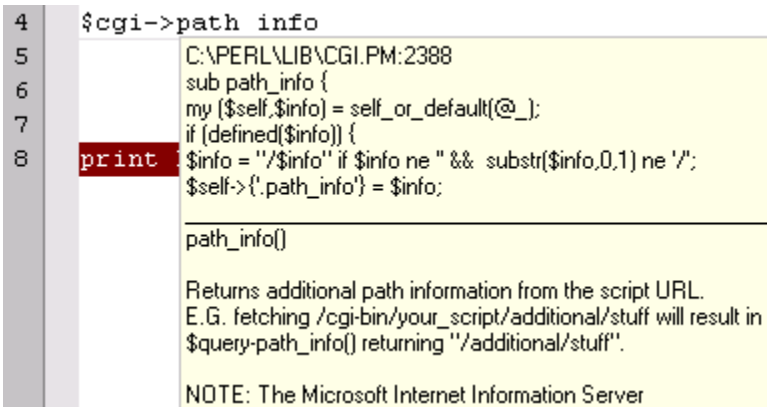
Starting and stopping apache

Program: `c:\perl\bin\apache.exe`
Parameters: `-f "c:/perl/conf/httpd.conf"`



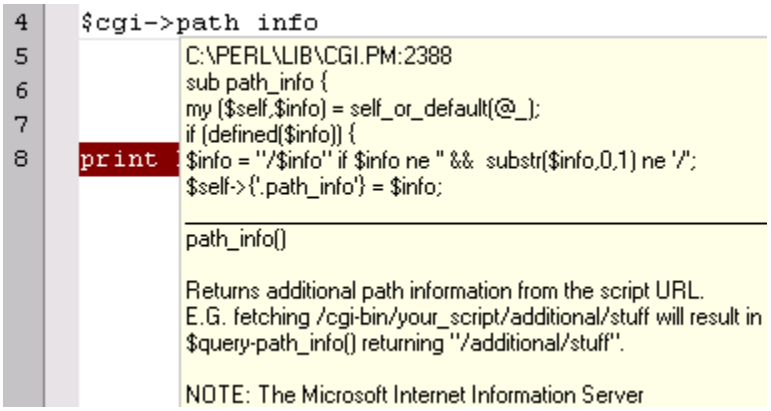
Opening a console window at the folder of the active script in the editor

Program: `cd`
Parameters: `"%folder%"`



Creating a new email in Microsoft (r) Outlook (this will need MS-Office (r) installed) with the active file in the editor as an attachment.

Program: `c:\program files\Microsoft Office\Office\Outlook.exe`
Parameters: `/a "%pathsn%"`



Using pod2html

Program: c:\perl\bin\pod2html.bat

a) First method

Parameters: `--infile=%PathSn% --outfile="%folder%\%FileNoExt%.html"`

Notice the use of double quotes to make sure that paths with spaces are not interpreted from windows as multiple parameters.

The above will extract the pod text in your script in an html file. The file will be created in the same folder.

You could also set the above up so that the name of the output file is queried first:

```
-infile=%Pathsn% -outfile="%Querybox%"
```

b) Second method

Parameters: `-infile=%PathSn% %n<pod - %filenoext%.html>%%getfile%`

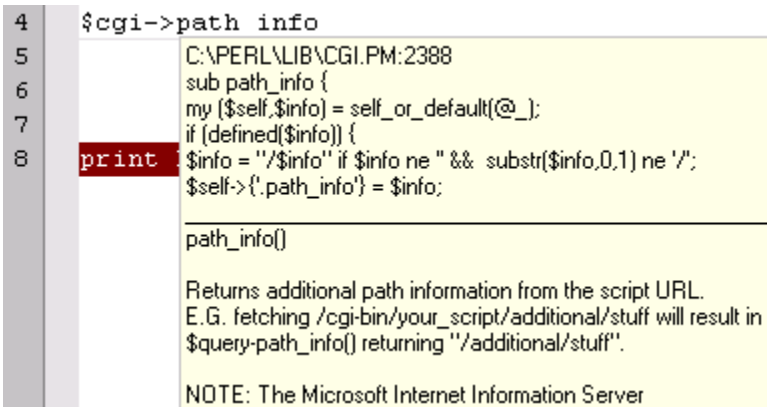
If you don't specify the -outfile in pod2html then the output is sent to standard output. The

parameters `%n<pod - %filenoext%.html>% and %getfile%` are "magic" which are deleted when running. They command optiperl:

`%n<pod - %fileNoExt%>%`
To open a new file in the editor named "`pod - filename`" (without the extension)

`%getfile%`
To get all output and insert in the editor. This will capture all the output from pod2html.

Examples of perl tools:



Adding a tab character if front of each selected line

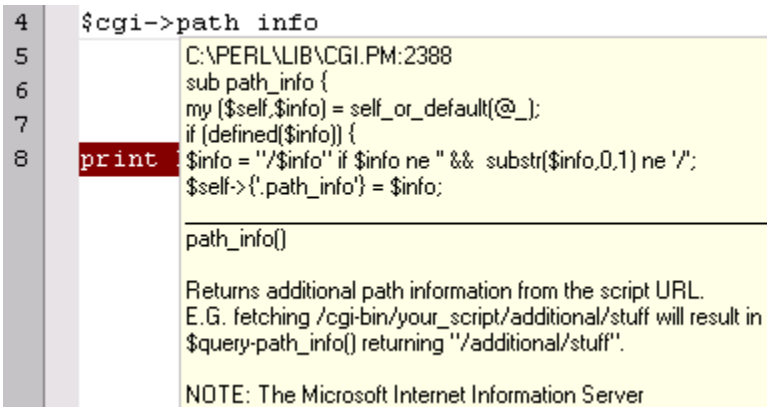
Program: `TabIndentSelection.pl`
Parameters: `%SENDSELECTION%%GETSELECTION%`

Enter the following for `TabIndentSelection.pl`:

```
# Get input from OptiPerl
@lines = <STDIN>;

# Do now what you want with @lines and the rest of the variables
foreach (@lines) {
 $_ = "\t" . $_;
}

#Send it back
print @lines;
```



Also see an implementation for CVS.

Configuring user tools

You can add your own external programs under the tools menu, and associate them with the script you are editing. From this window you can set up these programs.

To add items, use the following fields:

- Name is the display name in the Tools Menu
- Program is the external program you want to run
- Parameters are the command line parameters for the external program.

You can also select an image for each tool from menu "Tool settings".

In the Parameters field, you can add some tags to associate the script you are editing. Right click and select the "Build parameters" item in the popup menu to invoke a parameters dialog.

From the parameters dialog, you can select additionally:

- A starting path for the program when executed
- If it's a console script, whether the console window will be left open after it terminates.

Pausing for each page in console output

If you are running in the console and the output is large, you might need to pause on each page. To do this, after pressing "Run in Console", enter in the dialog box:

```
%PathSN% | more
```

This way after each page of output you will be prompted to press a key for the next page.

Tag list

The parameter list dialog is used when running your script in the console or selecting parameters in the external tools.
The edit box shows the parameter going to be used, while the list view some predefined tags and their value (at the time the dialog was invoked). The hint of the parameter edit box previews the parameters after processing the tags.

`%file%`
The filename of the script.

`%filenoext%`
This is the filename of your script excluding the extension.

`%folder%`
The folder in which the script is, excluding the trailing backslash.

`%path%`
This will be replaced with the full path to the script you are editing.

`%pathsn%`
Full path to script as a dos short name (no spaces).

`%querybox%`
Prompts user to enter some text.

`%ARGV%`
The arguments entered in Run / Arguments. Needed for console scripts.

`%selection%`
The text selected in the editor. If the text spans many lines then the EOL characters are deleted.

`%url%`
Sends the same url that is called in the internal browser when running the script.

`%word%`
The current word under the cursor.

`%get%`
The active get method text

`%post%`
The active get method text

`%pathinfo%`
The active pathinfo text

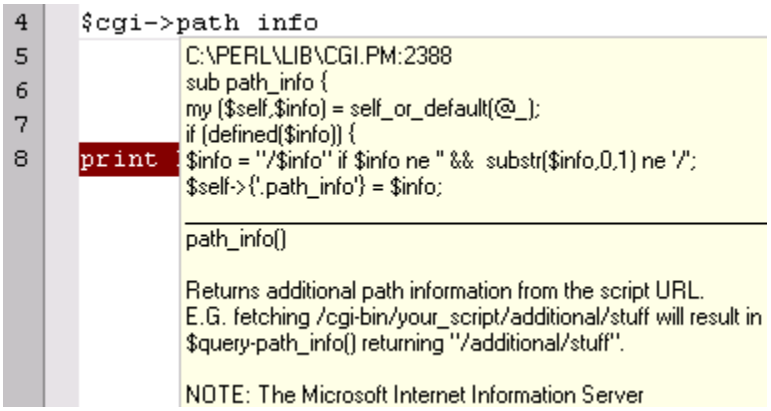
`%cookie%`
The active cookie text

`%data0% ... %data9%`
These are replaced with the corresponding fields of the active project's options. Check how these are used for an implementation of CVS.

`%f<filename>%`
Gets replaced with the text in the file "filename", for example `%f<c:\test\testparams.txt>%`

`%n<filename>%`
Creates a new file in the editor with the name "filename". This is used mostly with the special tags below.

`%o<FullPathToFile>%`
Opens a file in the editor. This is the only tag that when used, does not require the "Program" path to contain something to be executed. For example, you can create a tool with the "Program" field empty, and in "Parameters" something like `%o<c:\myfile.txt>`. When ran, this will just open the file in the editor.



The text you enter in the above 3 commands is also evaluated. For example, you can enter:

`%n<%fileNoExt%-text>%`
If you ran a tool containing this parameter on test.cgi, then a new tab in the editor would pop-up with the name `"test-text"`

Tags that modify OptiPerl's options

These tags do not actually get replaced with text; they only change the behavior with the script.

`%toggle<OptionName>%`
Toggles any boolean option value of optiperl. See the file settings.ini for possible values. For example: `%toggle<WordWrap>%`

`%set<OptionName=Value>%`
Toggles any options value of optiperl. For example: `%set<WordWrap=true>%` `%set<BlockIndent=3>%`

Special tags

If optiperl detects one of the following tags, then OptiPerl opens a pipe with the script. These tags do not actually get replaced with text; they only change the behavior with the script.

%GETFILE%
Replaces the text of the active file in the editor with the text sent from the script.

%SENDFILE%
Sends the text of the active file to the script. The script should read it from <STDIN>

%GETSELECTION%
Replaces the text of the selection in editor with the text sent from the script.

%SENDSELECTION%
Sends the selected text in the editor to the script. The script should read it from <STDIN>

%GETPROJECT%
OptiPerl replaces all files of the current project with the filenames sent from the script. The filenames must be absolute and delimited with a \n (newline).

%SENDPROJECT%
Sends all the filenames of the active project to the script. The filenames are absolute and are delimited with a \n (newline).

%GETLINE%
Navigates and positions the cursor on a line of a file, opening it in the editor. The format optiperl will expect from <STDOUT> is:

```
line<tab>filename
```

so print from your script something like:

```
$line\t$file
```

If this parameter is used together with %GETFILE% then make sure this is the first line printed.

%SENDLINE%

Sends to the script the active filename and caret position, in the same format as above. It will be the first line sent, and you can read it in a perl script by using:

```
($linenum,$activefilename) = split(/\t/,<STDIN>);
```

Adding to the menu

When you create a new tool, it is added in menu Tools / User tools. It can also be found in the customization dialog (menu Tools / Customize toolbars) under the "user tools" category, so you can drag & drop in anyplace in the menu or toolbars.

You can also assign a shortcut to the tool, from menu Tools / Customize shortcuts.

Resetting the main menu

If you reset the main menu from the Customize toolbars dialog, then all the user tools will appear under the "User tools" menu, and deleted from other positions you may have set them to.

Storage file

The tools are saved in file tools.csv in OptiPerl's Application data folder. It is comma separated and each value is quoted, so you can open in any database.

Each line has the following format:

```
"Tool name","executable","parameters","starting folder","index of graphic
(0..109)","leave console open after run (0..1)","Unique ID"
```

The unique ID can be any unique string with without spaces, and is used to identify the tool in the main menu. If however the string starts with "CVS_" then when resetting the toolbars, the default will be for it to go to the CVS menu.

Using absolute paths

An important rule for scripts to run both offline and remotely, is to use relative paths to the files they use, or use the **subst** command demonstrated below.

Assume your remote directory structure looks like this:

```
/htdocs/index.html
/cgi-bin/form.cgi
/data/data.txt
```

You have your site offline in windows looking like this:

```
c:\webs\mysite\htdocs\index.html
c:\webs\mysite\cgi-bin\form.cgi
c:\webs\mysite\data\data.txt
```

On your remote server you would access the file data.txt from the script form.cgi, using something like:

```
open(FILE, "/data/data.txt")
```

This however would not work offline! the folder c:\data does not exist.

<u>You have 3 solutions to this problem:</u>

1) Use relative paths to the file:

```
open(FILE, "../data/data.txt")
```

2) Use version converter remarks:

```
#@Local#?#@Server
#? open(FILE, "/data/data.txt")
open(FILE, "c:/webs/mysite/data/data.txt") #?
```

After the upload this would become:

```
#@Server#?#@Local
open(FILE, "/data/data.txt") #?
#? open(FILE, "c:/webs/mysite/data/data.txt")
```

3) Use the subst command

This is a technique that will allow absolute paths without any changes. It is very easy to do and you can also create a user tool in OptiPerl to do this automatically.

i) Open a MS-DOS prompt or Console window
ii) Enter:

```
subst z: c:\webs\mysite
```
(don't add the ending backslash)

where z: a drive letter that is not being used. You can also leave it as z, or if you manage many web sites, assign them to large letters like x:, y:, z: etc.

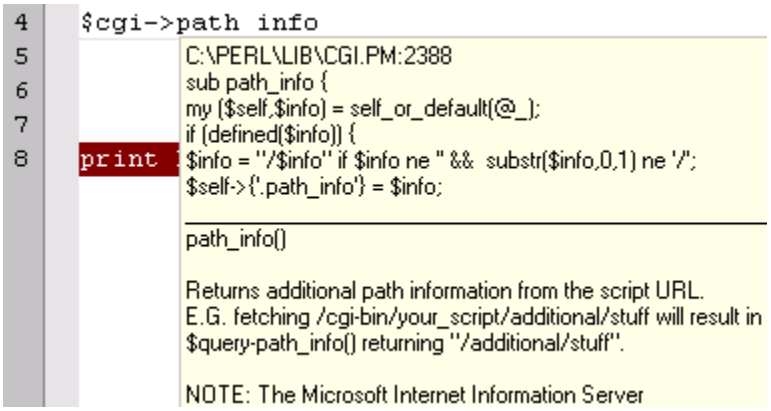
iii) You are done! Now open the file form.cgi from optiperl, but select drive z: to open from. Notice that the file is at `z:\cgi-bin\form.cgi`

If you run form.cgi, even absolute paths will work! Also don't forget that this is **not** a second copy of `c:\webs\mysite\cgi-bin\form.cgi`. If you save the file in the z: drive it will also change in the original folder.

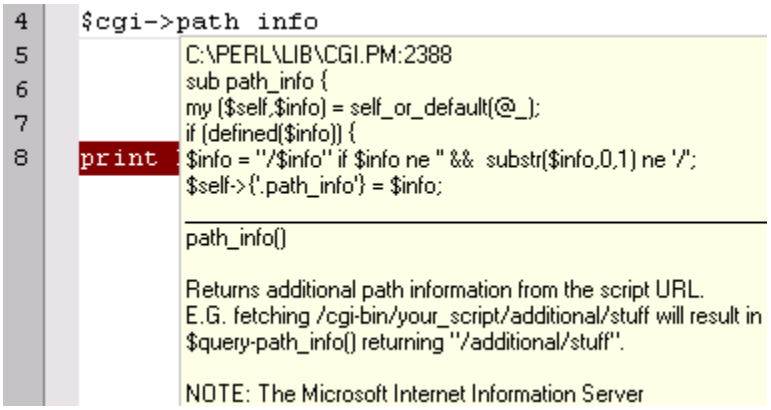
To delete the z: drive, restart windows, or enter the command:

```
subst z: /d
```

The files will not be deleted of course; they are still in the original path.



You will also need to change the internal or external webroot from `c:\webs\mysite\htdocs` (and accordingly any of it's aliases) to `z:\htdocs`. Remember that the file must be run from the z: drive.



If you are using sendmail and date support, don't forget to setup again, by selecting "drive to copy files" as the new drive z:. Note that this will create folders under c:\webs\mysite, but this should not be a problem.

Common setups of web hosts

Unless you have a dedicated server, your website will probably be hosted on a machine with many other accounts. This means that depending on your web host, an absolute path to your website might look something like:

```
/usr/local/customer/virtualfp/www.mysite.com/
```

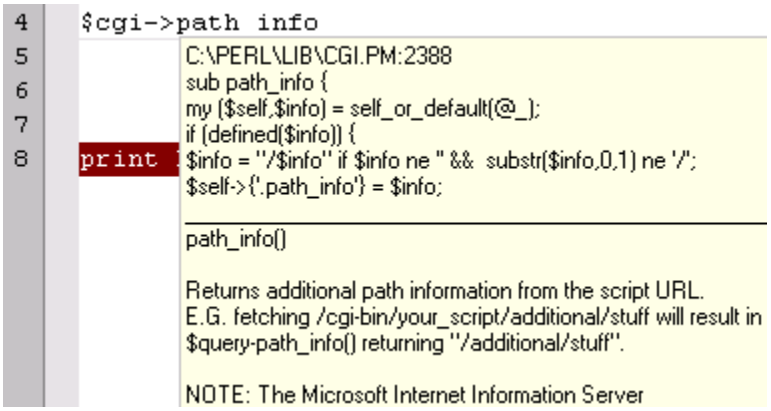
```
/usr/local/customer/virtualfp/www.mysite.com/cgi-bin

/www.mysite.com/htdocs
/www.mysite.com/cgi-bin

/users/students/disk1/george/public_html/
```

You will need to check with the CGI directory help section of your web host to learn where your directory is. In any case, if you want to use absolute paths, you will need to recreate the structure in your hard disk. For example:

```
x:\usr\local\customer\virtualfp\www.mysite.com
y:\www.mysite.com\htdocs
z:\users\students\disk1\george\public_html
```



Don't get mixed up however if your remote root path is simply "/", this does not mean that your files are also at root level. Sometimes web hosts might set-up an alias when you log in your FTP or SFTP account, so that '/' is really one of the above for your account only. But still when programming, you will need to use the real path to your files.

Linking to a project

You might want each project to be associated with it's own drive letter. Have every project set it's drive in one of the Project options / data0..9 fields. For example enter in Data5 `z:` , and enter in Data6 the real starting path of the project (like for the above example `c:\webs\mysite`). Add a custom tool:

Name: `Subst project`
Program: `Subst`
Parameters: `%data5% %data6%`

You can also create a tool to unlink after use:

Name: `UnSubst project`
Program: `Subst`
Parameters: `%data5% /d`

You can also replace all the files in the project with the files under z:, plus settings for the internal and external server. Note however that when first loading the project, if the subst command has not been run yet, you will get warnings about the files not existing in drive z.

Ignore the warning, and execute the subst command.

Automating when managing many webs

A better procedure would be a create a .bat file that makes all the substitutions, and run once per window session (you can even put this file in the Start menu / Startup folder so it gets executed automatically whenever windows starts).

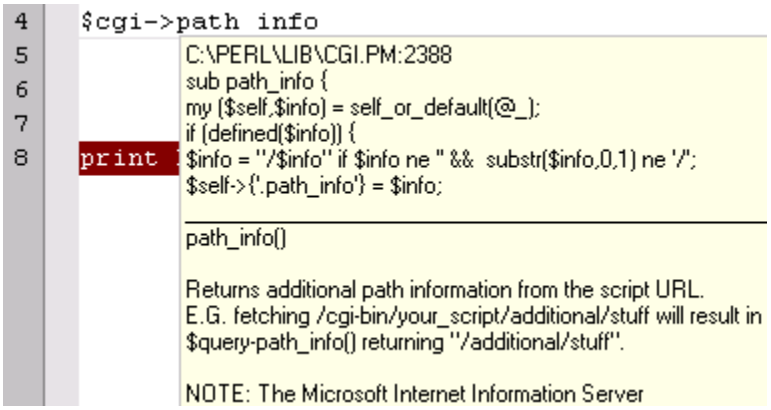
Create `Webs.bat` with the following:

```
subst x: c:\webs\ruth
subst y: c:\webs\george
subst z: c:\webs\harry
```

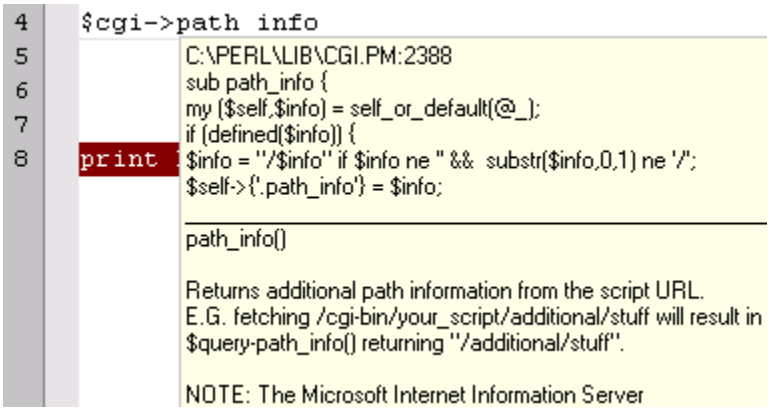
Working on the same project on different machines

If you are working on a project on different computers, chances are that the drive lettering may change, thus invalidating project set-ups. However by using the subst command, you can set your projects to point to the virtual drive on both computers, and then change how you call subst between computers.

Other tips

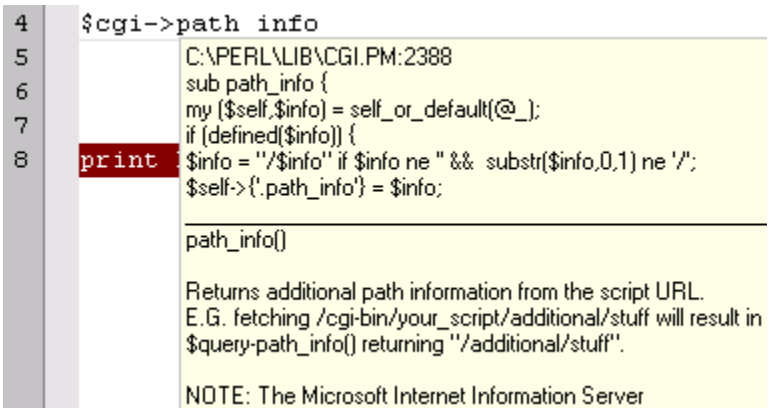


Using this technique might be necessary if you are trying to run both offline and remotely, complicated perl projects that share many scripts and data folders for storage, like on-line discussion boards, search engines etc.



A fourth method of working with absolute paths, is very similar like the above, but your will need a second hard disk.

Repartition it with 15-20 partitions of 50-100 mb each assigning to each a drive letter. A major drawback however it that a web site might "grow" larger then the partition that holds it.



Another option is for a group of developers sharing a network. Unfortunately the subst command will not work with UNC paths like \\main\c\webs. But you can have the same result the following way:

i) Go go to explorer (not internet explorer), menu Tools / Map network drive. Enter a drive letter (like z:) and as folder the UNC path to the remote folder that is the web root, for example: `\\maincomputer\customers\webs\GeorgeSite`. Select also "reconnect at log-in" so you don't have to do this every time windows starts.

ii) Again the new drive letter **is not** a local copy. Saving to files in z: will also save them via the network to the remote machine.

Using real domain names

Many scripts require you tell them the domain name they are running on. For most cases when offline, `http://127.0.0.1/` or `http://localhost/` will work. However when you upload, this must change to the real domain name. There are two solutions to this problem:

1) Use the version converter. Usually a script will read the domain name into a variable, that it will use for the rest of the program. Modify to something like this:

```
#@Local#?#@Server
#? $domain = 'http://www.mysite.com/';
$domain = 'http://localhost/'; #?
```

After the upload this would become:

```
#@Server#?#@Local
$domain = 'http://www.mysite.com/'; #?
#? $domain = 'http://localhost/';
```

2) Modify the computers **hosts** file.

The `hosts` file (without extension) is a text file located:

- Windows NT/2000/XP: `c:\winnt\system32\drivers\etc\hosts`
- Windows 98: `c:\windows\hosts`

If it does not exists you can create it, by adding just one line like the below.

It's format looks like this:

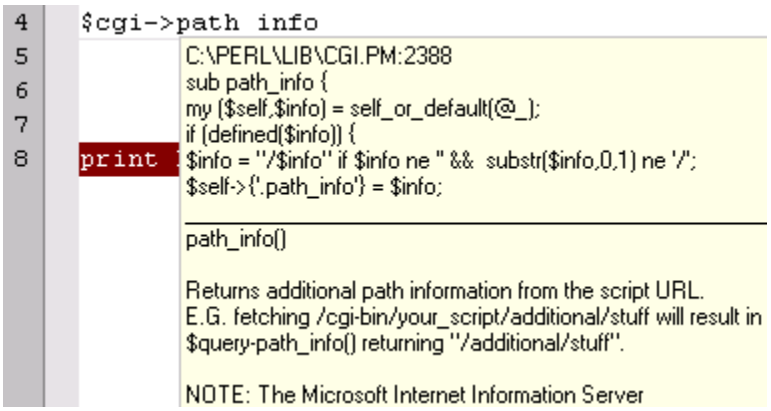
```
127.0.0.1   localhost
```

What this file does is map IP addresses to host names. Usually it contains only one entry, the above, that tells windows that `http://localhost/` is the same as `http://127.0.0.1/`

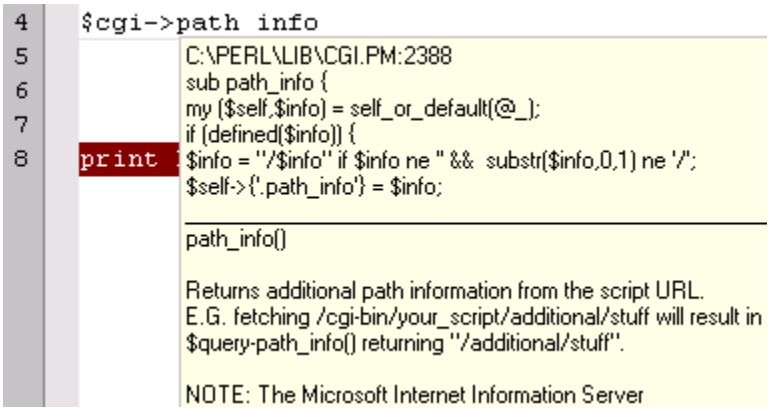
So what we can do is add another line:

```
127.0.0.1   www.mysite.com
```

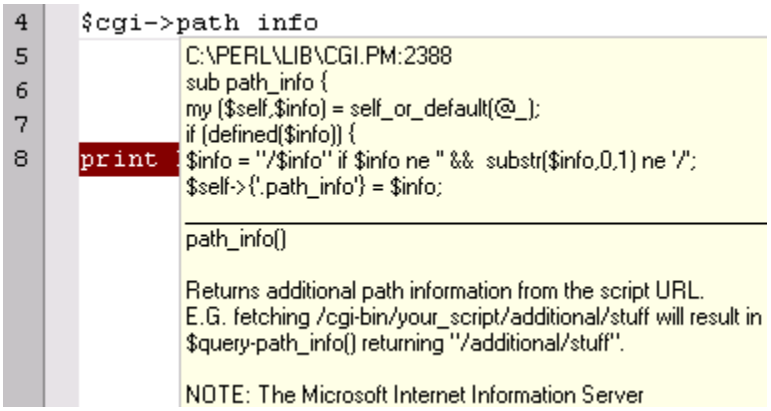
That's what is needed! Now when http://www.mysite.com/ is called, the server at http://127.0.0.1/ will be invoked. This can be optiperl's internal server, or any other external server you are using.



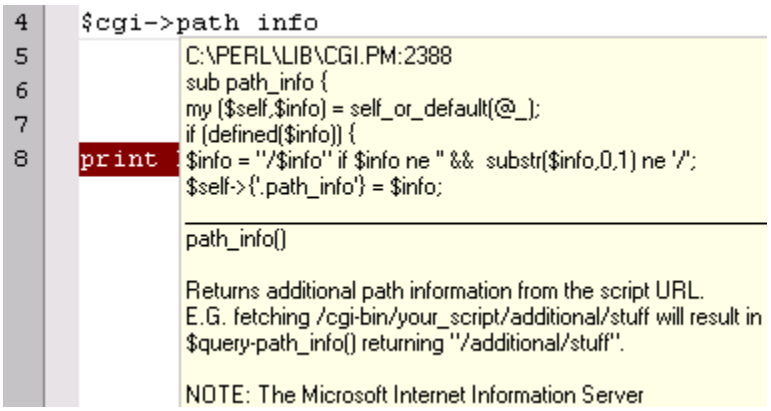
You can also change the Host from Options / Run or Project options so OptiPerl also can use the new domain in it's requests.



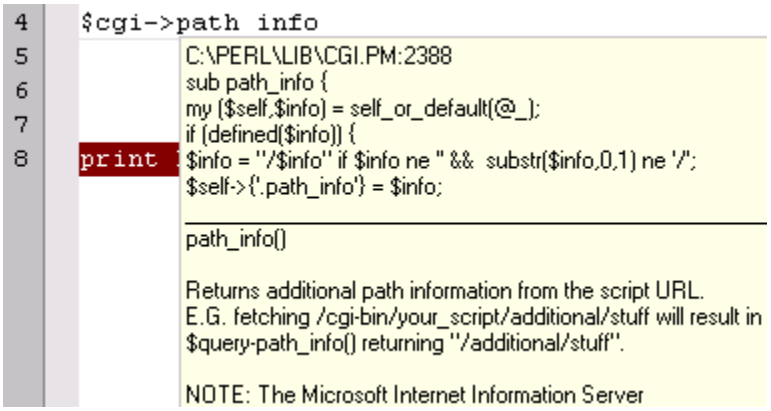
The *first* time you call this mapped domain, you might get a window "Connect to" dialog prompting to connect to the internet; Don't connect, just press "Cancel".



But this line must be removed, when you then connect to the internet or else the real www.mysite.com will never be called when you want to!



Note also that the mapped domain does not have to only be used for a http protocol. You can map any domain for that is used for any kind of port. For example, if you are testing a script that uses mySQL, and you connect to your server's mySQL using `mysql.mysite.com`, then you can map `mysql.mysite.com` in the hosts file so that your local mySQL server can be used.



IMPORTANT: Because of the above, if your have mapped a url like www.mysite.com, but your FTP address is also www.mysite.com (and not something like ftp.mysite.com) then while the url is mapped, your will not be able to connect to your real FTP site. This will happen becase for your computer www.mysite.com will mean 127.0.0.1, which also means itself. It will try to connect to an unavailable FTP server.

Automating when testing web sites

You can create a set of tools in OptiPerl to add and remove this line and integrate with the active project.

Add in one of the project's options data fields, like data7 the value `www.mysite.com`

Create a user tool:

Name: `Map local IP to project`
Program: `MapIP.pl`
Parameters: `%data7% "<path to hosts>"`

To unmap afterwards, create the following user tool:

Name: `UnMap local IP to project`

Program: `UnMapIP.pl`
Parameters: `%data7% "<path to hosts>"`

MapIP.pl and UnMapIP.pl are included in OptiPerl's installation. The <path to hosts> is replaced when OptiPerl is installed depending on your windows version. They add and remove the line:

```
127.0.0.1   site
```

where "site" is the first argument sent to the script.

500 errors

Even after testing a script offline using the internal or an external server, you may still get a 500 error after running on the remote host. This can be caused from a number of reasons:

- The shebang (the first line of your script) is not correct. You may need to check where the path to perl is located.
- Modules used in the script. Many web service providers don't have all the modules found under `\perl\lib` and `\perl\site\lib` installed. If this happens and the module is written only in perl, perhaps you can upload the module yourself in your cgi-bin folder recreating the folder it resides relative to your script. If not, you will need to ask the provider to install the module.

For example `MIME::Lite` in `\perl\site\lib\MIME` can be uploaded to your cgi-bin folder, by creating `/cgi-bin/MIME/` and uploading there `\perl\site\lib\MIME\Lite.pm`.

Other issues if you are uploading with a third-party program, or the extensions used for scripts are not common (pl plx pm cgi etc.)

- Windows line endings. If you uploaded the file with windows line endings with a binary mode, probably it will not work. From the project manager, check if the transfer is "Text". If uploading via "Save to remote location", and the filename is not pl,plx,cgi,pm then modify the file `DefaultModes.xml` in OptiPerl's installation folder.

- File and folder permissions. Make sure that the script was uploaded with a permission of 755, and the folders containing it have 755 permissions. The default permissions are set in the `DefaultModes.xml` file.

Still not fixed?

If the problem still persists and it is not a fault of the above, then make sure you add the next line in your script:

```
use CGI::Carp qw(fatalsToBrowser);
```

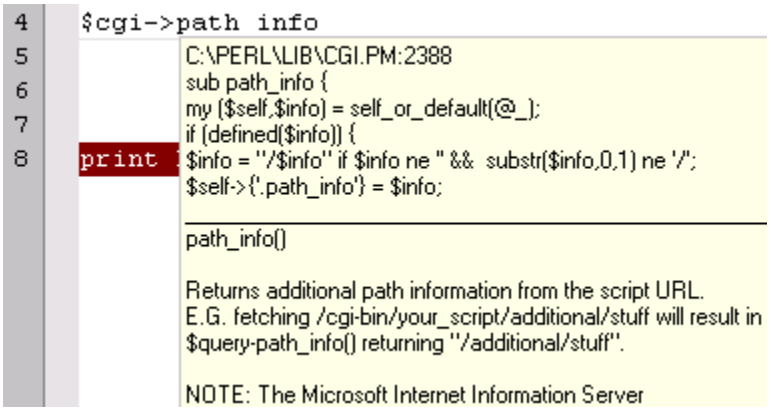
This will provide a detailed explanation about the error.

CVS

CVS is a version control system that records the history of your source files. It is freely available at http://www.cvshome.org/.

OptiPerl has been modified in ways to work better with CVS. A project in CVS can be linked to a project in OptiPerl, and you can easily checkout the latest version and commit new changes. OptiPerl however is not a GUI for CVS; it cannot help with uncommon tasks of CVS like first importing a project or administrative tasks. But for more common tasks, in day to day programming, it can help save a lot of time.

When you first install OptiPerl, the CVS menu is under menu Tools/User Tools/CVS. Beneath we will explain however how to implement those commands from scratch, for better understanding. Also most configurations of CVS might need some changes in the tools parameters.



Linking a project in OptiPerl to a CVS project

- Add in **Data0** of OptiPerl's project options the repository of CVS, for example:

```
c:\cvsroot
```

or for remote servers something like:

```
:pserver:faun.example.org:/usr/local/cvsroot
```

You can skip this if you use only one repository and it is set in the CVSROOT environment variable. Also remove all references to "`-d %data0`" below.

- Add in **Data1** the module name, for example:

```
mysite/cgi-bin
```

(notice the use of the slash (/))

- Add in **Data2** the folder in which you will checkout. You must setup the starting directory for checkout correctly, since especially for scripts being tested with web servers, it is not easy to change their location each time (unless of course this does not matter for you).

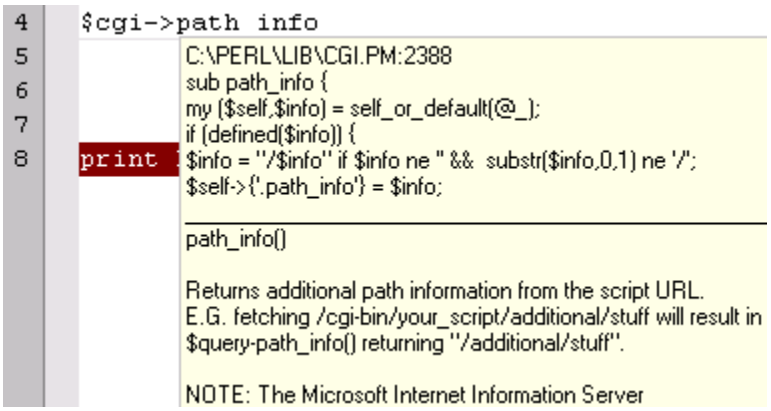
If your project's files are in `c:\webs\mysite\cgi-bin` and you first imported from this folder using

```
cvs import mysite/cgi-bin vendor start
```

then enter here `c:\webs`

- Add in **Data3** the top level folder of the project. For example

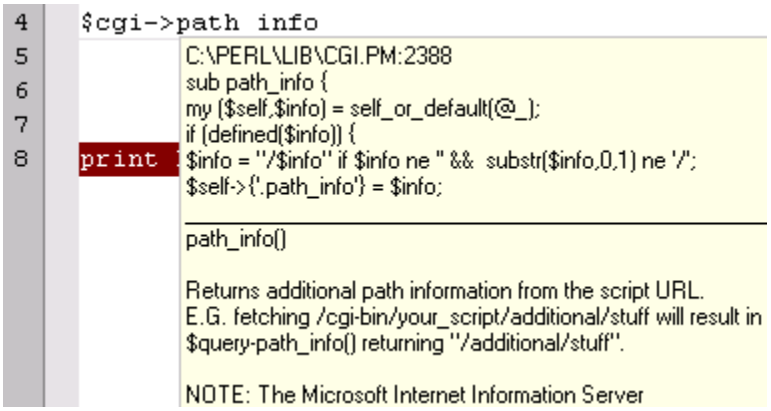
```
c:\webs\mysite\cgi-bin
```



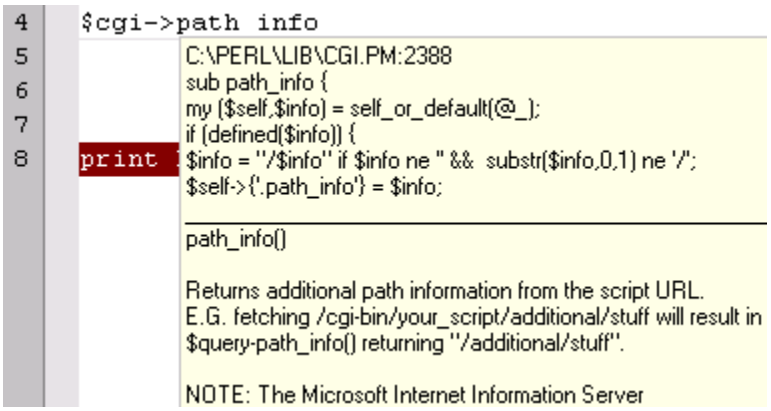
Logging in (for remote server)

Create a custom tool:

Name: `CVS Login`
Program: `CVS.exe`
Parameters: `-d %data0% login`



For this tool and all the next, from the "Build parameters" of "configure tools" (right click in the grid), select "Leave window open". This will keep the console window open after running, so you can verify the results.

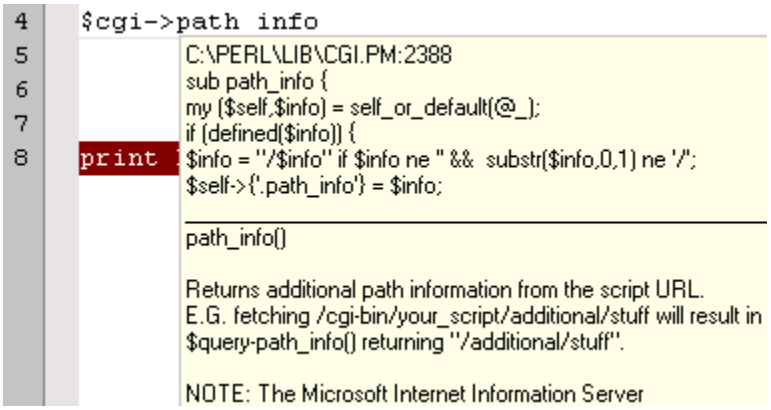


Checkout

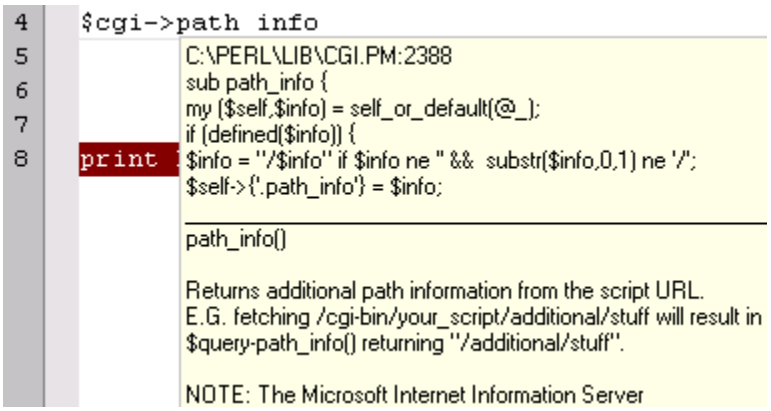
Create a custom tool:

Name: `CVS Checkout`
Program: `CVS.exe`
Parameters: `-d %data0% checkout -P %data1%`
Starting path: `%data2%`

We had setup %data2% with the folder above the top-level folder of the project. Remember that the top-level directory created is always added to the directory where checkout is invoked.



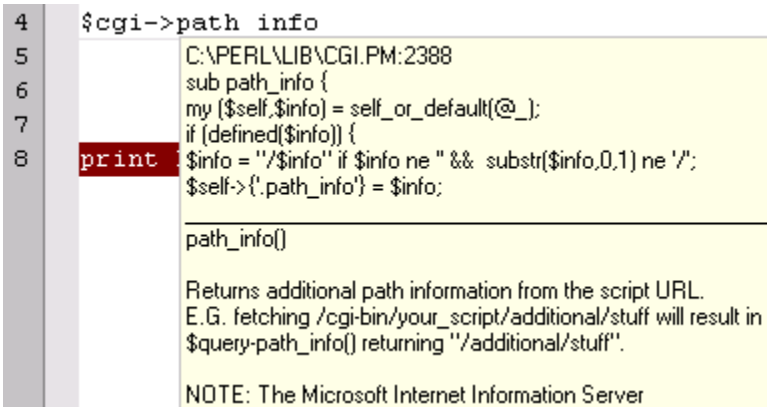
Note that this tool, and some of the next, requires you also enter the starting path. You can enter this value from the "Build parameters" dialog. The tags in the starting path are also evaluated by OptiPerl and replaced.



Updating

To bring the work tree in sync with repository (if you are part of a developing group) create a custom tool:

Name: `CVS Update`
Program: `CVS.exe`
Parameters: `-d %data0% update -d -C`
Starting path: `%data3%`

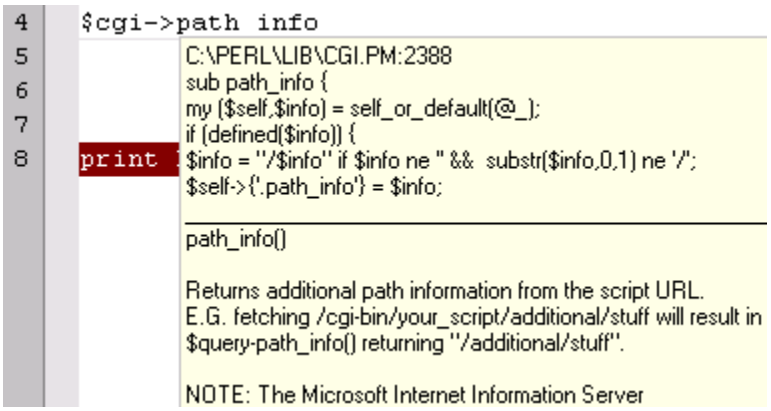


Commit

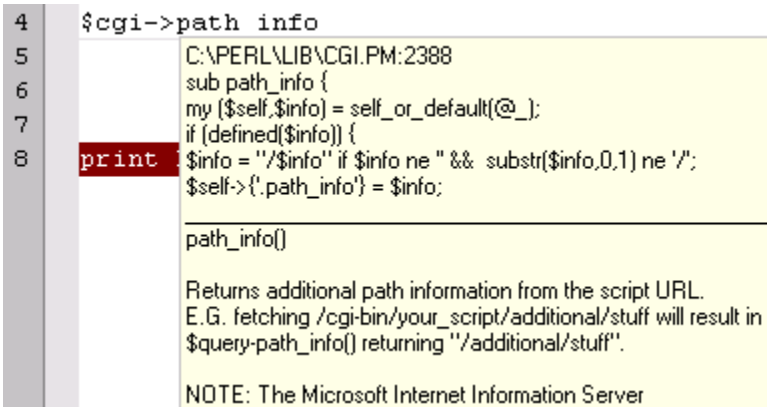
To commit changes, enter the following:

Name: `CVS Commit`
Program: `CVS.exe`
Parameters: `-d %data0% commit -m"%querybox%"`
Starting path: `%data3%`

The `%querybox%` will bring up an input box before running, so you can enter a string.



Make sure you save changes in the editor before committing.



Viewing changes with diff *(the fun part)*

The following will compare the active file in the editor, and display changes in a new page of the editor:

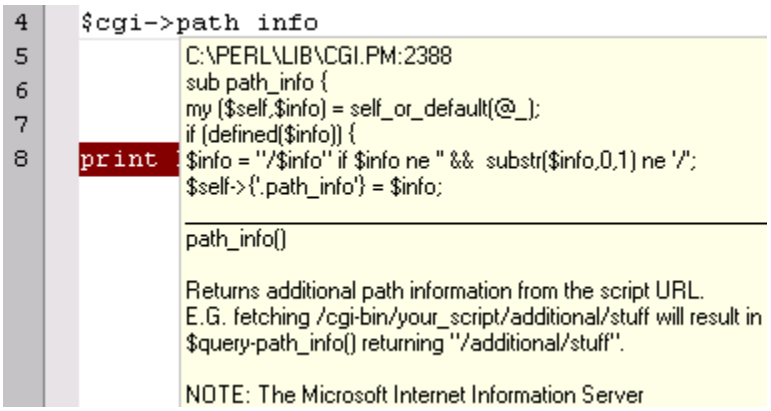
Name: `CVS Diff`
Program: `CVS.exe`
Parameters: `-d %data0% diff -c %PathSN%%n<diff - %fileNoExt%>%%getfile%`
Starting path: `%folder%`

`%PathSN%` is the full short path to the file, without spaces.

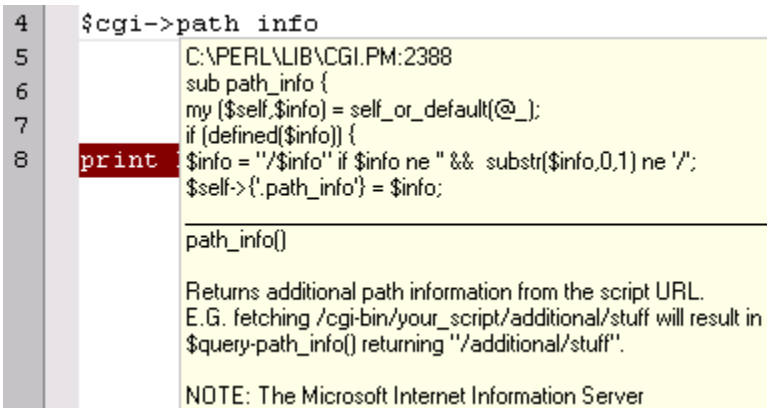
The parameters `%n<diff - %filenoext%>% %getfile%` are "magic" which are deleted when running. They command optiperl:

`%n<diff - %fileNoExt%>%`
To open a new file in the editor named "`diff - filename`" (without the extension)

`%getfile%`
To get all output and insert in the editor.



Especially with diff, but with other of CVS commands too, you may want to enter each time you run more parameters. Use the `%querybox%` for this.



Jumping to the line referenced from diff output *(the fun part II)*

If you try the above, on a new tab you will get the results of diff. We will explain here, how to jump to the line of the changed file, depending on where you have the cursor on the diff output file (made with the -c parameter). For example, if you have the cursor on:

```
.....................
diff -c -r1.2 vegetables3.pl
*** vegetables3.pl      30 Jul 2002 17:32:51 -0000    1.2
--- vegetables3.pl      30 Jul 2002 17:48:52 -0000

.................

--- 1,6 ----
  #!perl                          <- Cursor on this line
+    $a=1;
```

You will go to the file vegetables3.pl on line 1. To do this, create a tool:

Name: `CVS Goto line from diff`
Program: `c:\tools\CVSGotoline.pl` (wherever you save this file)
Parameters: `%SENDFILE%%GETLINE%%SENDLINE%`

Now create CVSGotoline.pl with the following code:

```
# Get the active line. The format used to send
# is linenumber<tab>activefilename
# linenumber is 0 indexed.
($linenum,$activefilename) = split(/\t/,<STDIN>);

# Get input from OptiPerl.
@lines = <STDIN>;

# the first line of the output should contain
# the file used for diff
die unless ($lines[0]=~/Index: (\S+)/);
$difffile = $1;

# Now starting from the the active line in the
# editor and going backwards, find something
# like "--- 10,30"
while ($linenum >= 0) {
 if ($lines[$linenum]=~/^--- (\d+),(\d+)/)
 {
  # print now the line we want to go to and the
  # file we want to go to:
  print $1 - 1 . "\t$difffile";
  exit;
 }
 $linenum--;
}
```

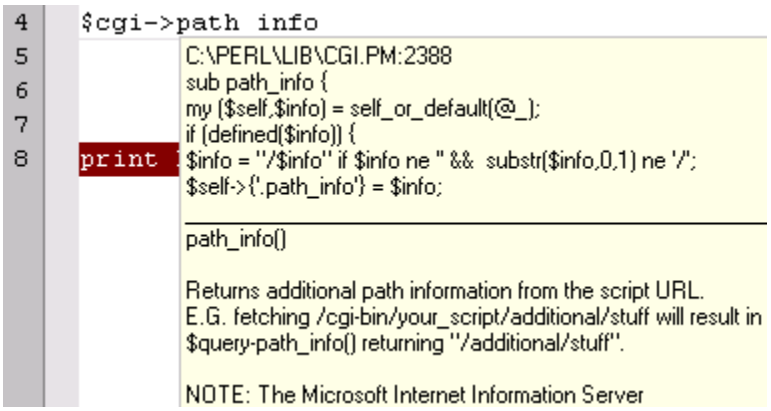
Notice the use of `%SENDLINE%` in the parameters. This instructs OptiPerl to send to the script as the first line something like:

```
14<tab>filename
```

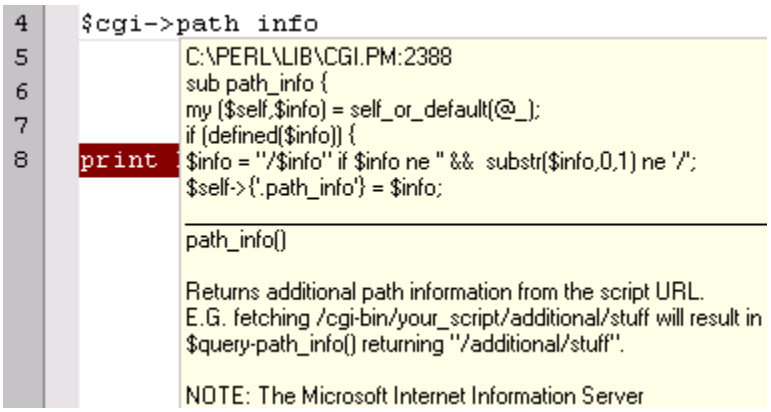
With 14 being the line under the cursor when called and filename the full path to the active filename.

With `%GETLINE%` OptiPerl expects as the first line of output the same format, so it will navigate there.

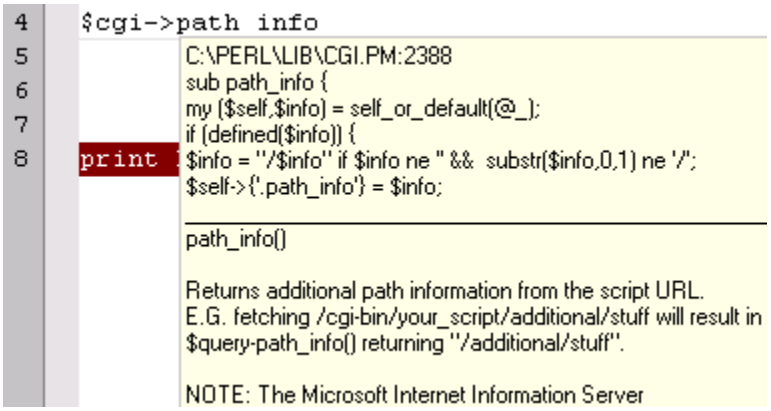
Other issues



For the implementations above, we have used Data0..Data3 of the project options. You may of course use other numbers, just make sure that the tools parameters correspond afterwards. Also keep corresponding place holders between all projects that use CVS, so the tools can be commonly used between projects.



When adding custom commands, the default is to add them in the tools menu. If you add many tools, this will make the menu huge! Don't allow this to happen, create a menu called CVS and put your tools there. You can also assign shortcuts and icons to your tools.



You got the feel of it! If you use CVS a lot, probably you have personal preferences on how to use it's commands, and probably you can set them up using OptiPerl's tools. If you are administrator of a multi-user version of OptiPerl, you can also copy the Tools.csv in OptiPerl's application data folder to each user, or append it's contents, changing the username for each if needed.

We did not hard-code CVS integration into OptiPerl because probably this would not be wise,

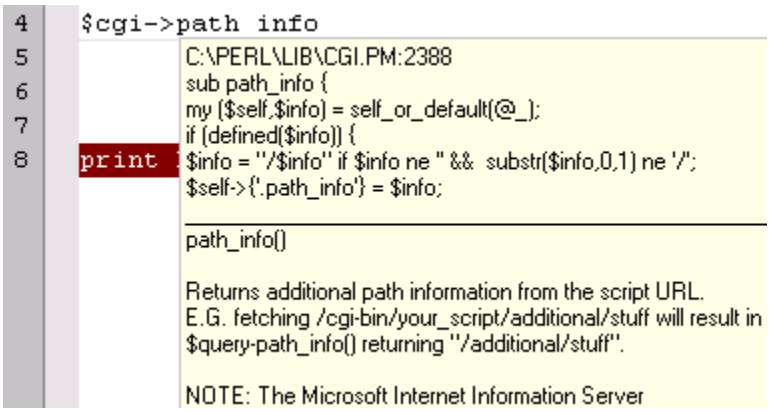
instead it can be built using OptiPerl's custom tools. This is because people who use CVS are probably more mature programmers that have strong personal preferences on using it. Matching this would be impossible, so it's better to give the framework and let programmers use CVS exactly as they want.

mySQL

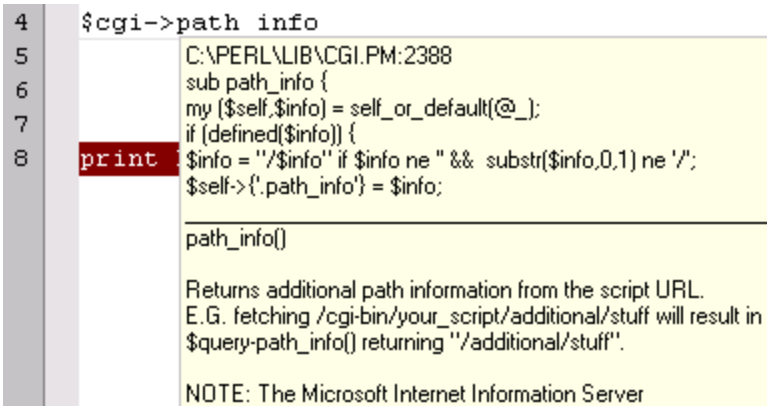
mySQL is a Open Source SQL database. Many perl scripts use mySQL and many web hosts support it.

To develop scripts using mySQL, you don't have to be connected to the internet and access the database on your web host. You can download the mySQL server and install it on your own computer for testing purposes.

mySQL is available at http://www.mysql.com/.



Most web hosts that provide virtual hosting, have one mySQL server for all their customers and each customer is assigned a database. If your web hosts has only given you a database name and a password, this is the case for you also. We recommend emulating this offline, by creating a database in your local mySQL server, with the same name as the one provided to you.



Notice the following code as an example:

```
use DBI;
my $dbh = DBI->
connect('DBI:mysql:username:mysql.mysite.com','username','password',
{RaiseError=>1});

my $insert_handle =
            $dbh->prepare_cached('INSERT INTO table1 VALUES (?)');

$insert_handle->execute('data');
$dbh->disconnect;
```

If you are running mySQL server offline, and have not mapped your remote mySQL server to localhost, you would use the following to connect to it:

```
my $dbh = DBI->
connect('DBI:mysql:username:localhost','username','password',
{RaiseError=>1});
```

So before uploading you would have to change the above to the first code. This can be done using the version converter.

m4 macro processor

You can add a command in the Tools menu to process a script using GNU's m4 macro processor (available at http://www.gnu.org/software/m4/m4.html. Using "configure tools", add the following:

```
Program: c:\m4\bin\m4.exe
Parameters: -P %PathSN% %n<m4 - %file%>%%getfile%
```

This will process the file open in the editor and send all the output into a new tab of the editor, named after the file. You can then edit and save the new file.

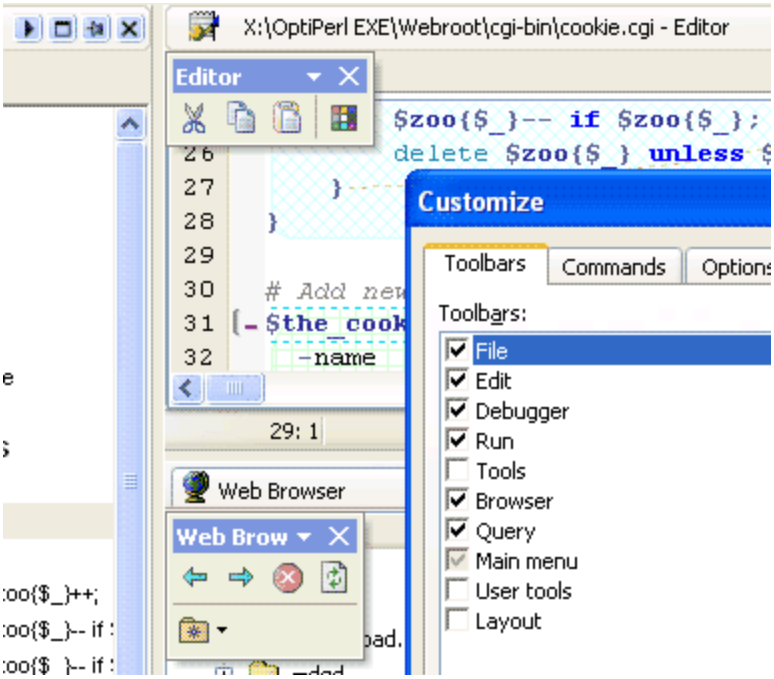
The parameters `%n<m4 - %file%>% %getfile%` are "magic" which are deleted when running. They command optiperl:

`%n<m4 - %file%>%`
To open a new file in the editor named "`m4 - filename`"

`%getfile%`
To get all output and insert in the editor.

Customizing menus & toolbars

OptiPerl's menus and toolbars can be completely customized. Also new toolbars and menus can be created. The menus and toolbars can also be placed wherever possible.

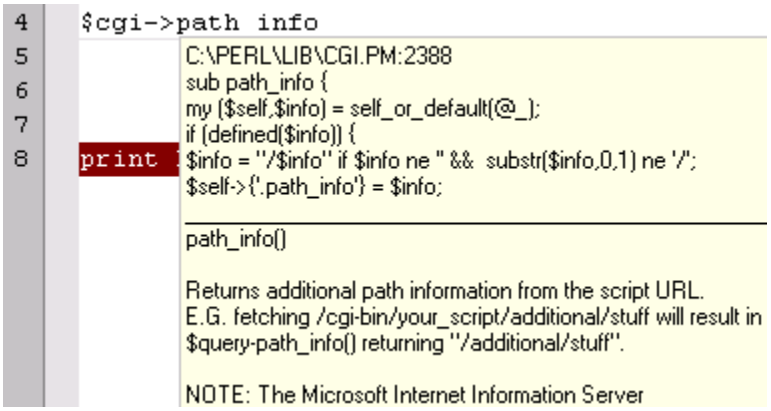


To customize do the following:

- Select Tools / Edit Toolbars
- Right Click on a toolbar or menu to quickly select the visible toolbars or select the Customize Toolbars dialog as above.

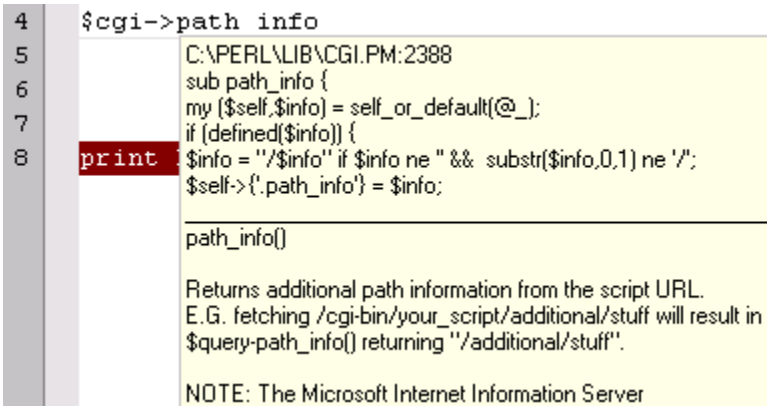
The customize toolbar window

- Toolbars: Shows a list of defined toolbars. You can create your own by pressing the "new" button.
- Commands: Shows a list of all of OptiPerl's commands, plus your own custom tools (in the My Tools category). You can drag and drop all the commands on the toolbars or menus.
- Options: More options that affect the menus.



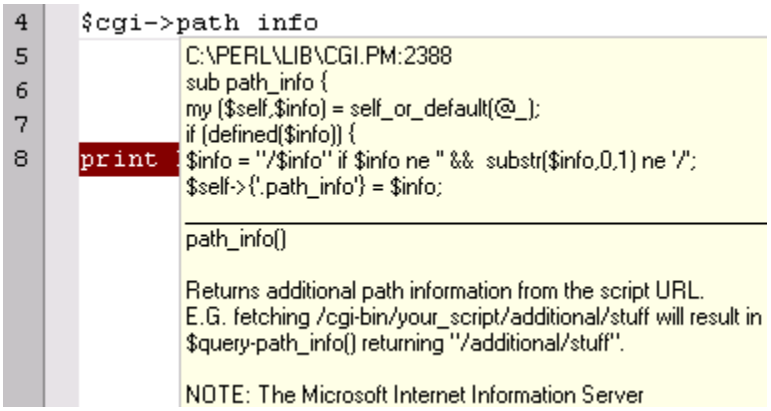
To create your own menu:

From the Commands tab, go to the category "Custom Menus" and drag & drop a new menu to a toolbar. Add commands to it. To rename it after adding it, right click on it and enter the new name in the menu dialog.



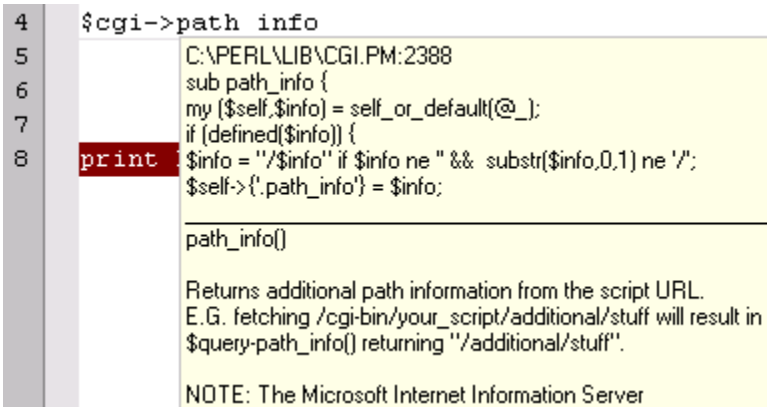
To add a separator line between commands on a toolbar or menu:

Go the command, right click it, and select "Begin a group"



To edit the right click pop-up menu of the editor:

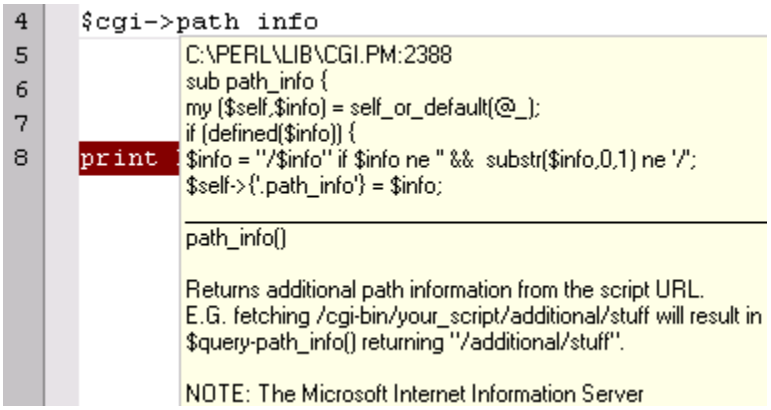
Go to Commands / Sub Menus and drag the Editor Popup sub menu to somewhere temporarily (like next to the main menu). Edit it and then return it back to the customize dialog.



To edit the left click pop-up menu

of OptiPerl's tray icon in the Tray Bar

Same as above. Select the menu Main menus / Tray Icon



You can also move the toolbars on the desktop to create floating menus. However they will really stay on top only if you have selected "single window interface" from the Layouts.

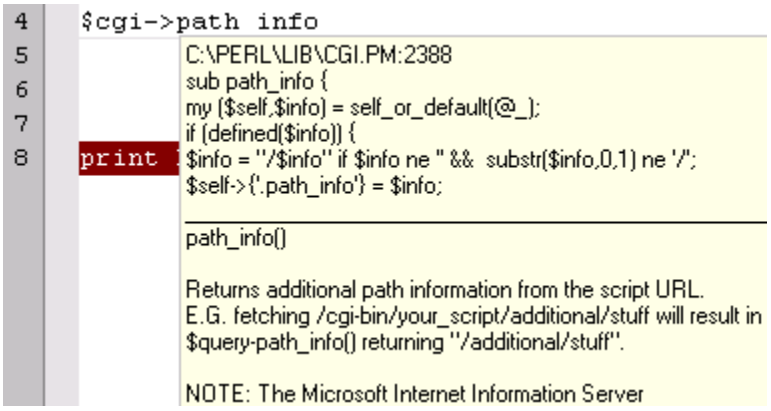
Layouts

Layouts affect the positioning and sizes of windows in OptiPerl's desktop, and whether or not a single window interface will be used.

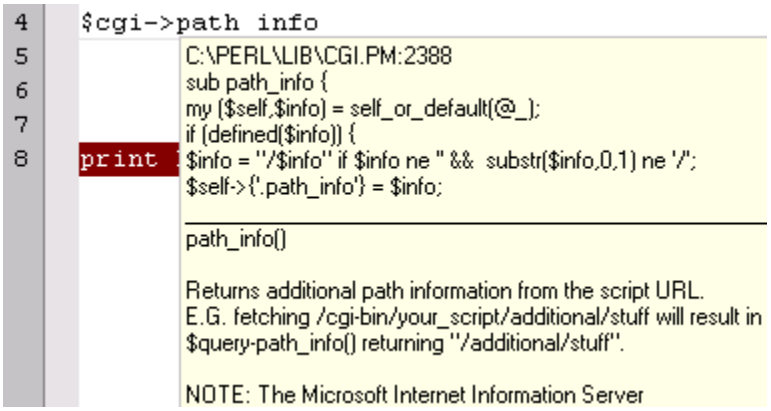
You can save as many desktop layouts you want in OptiPerl. The control of the layouts, plus a listing of the saved layouts is in the Windows Menu.

OptiPerl has 4 special layouts used for specific purposes:

- Run: Invoked when selecting "Run"
- Debug: Invoked when debugging.
- Edit: Invoked when pressing the button "Load edit layout" of the Layout toolbar. This toolbar appears only when you have started running or debugging. The edit layout is also loaded when you select "stop debugger".

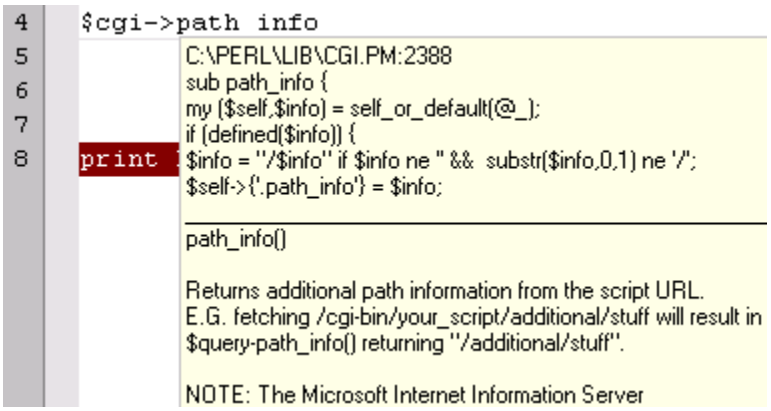


You can customize these layouts. Select each one, modify the windows to your like and select "save layout".



You must enable the feature Options / Environment / Enable Standard layouts for the above behavior.

- Last Session. Whenever you exit OptiPerl, the layout is saved as "Last Session". If you also exited which this layout selected, then the next time you open OptiPerl, the last session will be restored.



The "enable standard layouts" is selected by default. However you might feel more comfortable and more productive if instead of having a crowded desktop, have common windows open in the background and viewing them only when needed by pressing a shortcut key. For example the Web browser window takes up alot of space; instead have it in the background and whenever you need it press Ctrl-F12 to view it (or assign your own shortcut).

Multiple monitor support

If you are using a computer with more than one monitor, Optiperl can support this. When layouts are saved, the position of windows in all of the monitors are saved, so they are restored correctly. Also most pop-up windows (like find & replace, go to line, etc) will pop-up in the same monitor that the text is being edited in.

Customizing shortcuts

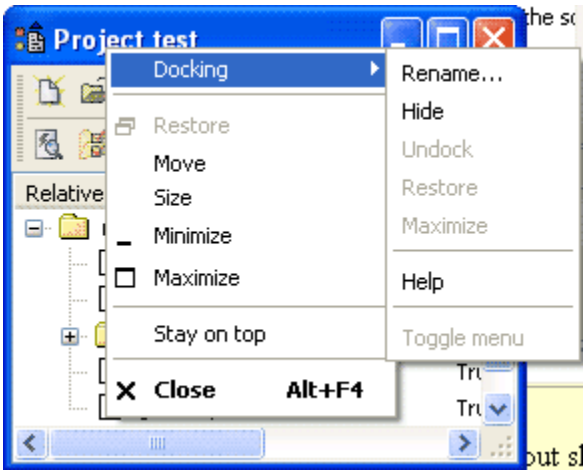
Almost all of OptiPerl's commands (except some standard keys of the editor like arrow keys, home and end), even custom tools you create can have a shortcut assigned to. This can be done from Tools Menu / Edit Shortcuts.

To select or change a shortcut, select the action you want, and press the shortcut in the edit box of the window. Afterwards, press the "set" button. If the shortcut is already used, then you be prompted to select a different shortcut.

About OptiPerl's windows

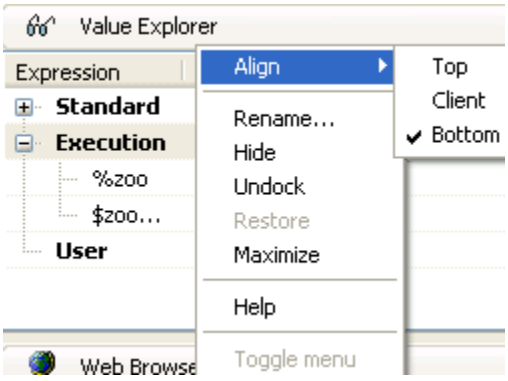
In OptiPerl you may have many windows open at the same time, that can be docked to each other, docked in a tabbed interface or floating. More control over the behavior is set in options dialog / Windows.

Floating windows

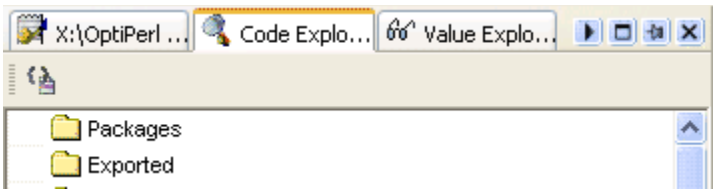


To undock a window, drag and release it's title bar. You can also force floating by holding down the control button while dragging. On a floating window, by right clicking on it's title bar, you can select "stay on top" to force it to always remain on top of other windows. Also, if selected in the options dialog, all floating windows add a button in Windows task bar.

Docked windows



To dock a window, drag it next to another window. You can also drag it in the middle of another window, to create a tabbed interface with it:



Docked windows have by default two right-click context menus. It's window menu (above) is shown when right-clicking while holding down the control button. If the window has a context menu, then this is accessed by right clicking.

How OptiPerl finds modules

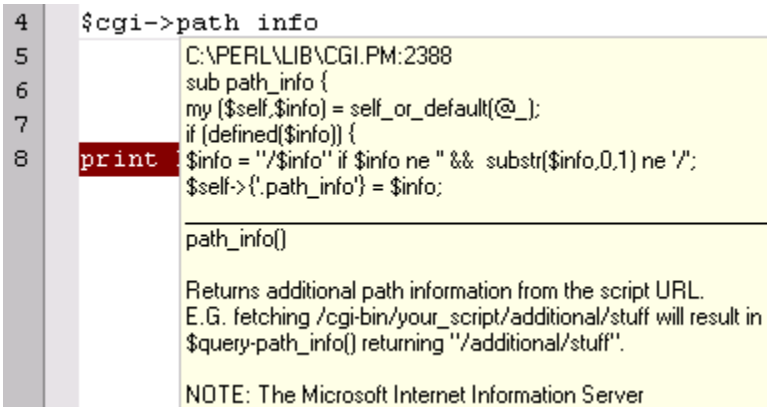
OptiPerl needs to know the library path for your modules, so it can find them in Code Explorer and to use Code Completion. Here is how to setup this search path:

- Options Dialog / Perl. Add a search folder in the @INC edit box, or just press the "Guess" button for a standard set of folders.
- Project Options / @INC. If the files in your project have their own library path, the above may be overridden here.

Overriding the search path is necessary if you have a more complex project that has a main module with a statement like:

```
use lib ('./Data'    ,
         './Sources',
         './',
        );
```

You should add these folders as absolute paths in the search path of the project options. You can also press the "Import Lib" button; this will parse statements like the above from all the files in the project, and automatically create the search paths.



OptiPerl's @INC path does not affect Perl's PERL5LIB path when actually running or debugging scripts. It is used only within OptiPerl, for Code Explorer and Code Completion. You cannot use this search path to tell perl where to find modules when running or debugging. To do this, use code like above.

How OptiPerl saves files

When using a classic word processor, an edited file is saved only when you press save. This however can not be done in OptiPerl. OptiPerl needs to save the file very often, even if you don't tell it too, because it is the only way to run it, debug it and run tools on it. However you may notice that the behavior of a classic word processor is emulated completely.

When optiperl opens a file, it keeps two copies of it in memory. One is to display the editor, another is the original state the file was when first opened, and only changes when you select "Save". If the file was saved by optiperl so you could run or debug it, and then you close it, selecting "cancel" on saving the changes, what actually happens is that the file gets saved; but with it's original contents (if of course this is needed).

All the above is completely transparent to you; But have this in mind if you notice that the file has been actually saved after running it, even though you did not press save. If you cancel saving changes afterwards, the file will be restored.

Perl Information

Shown here is the information about the build of Perl you are using. The "Detailed Information" button toggles between the information extracted with the –v and –V switch.

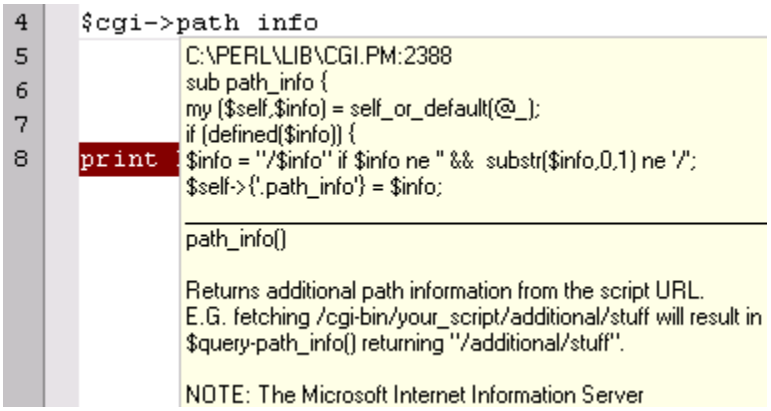
Perl Core and Module Documentation

Included with OptiPerl is all of Perl's Core and Module documentation, organized with topics and keywords. You can also search for custom queries. Open the help manually from Help/Perl Documentation, or search for the keyword under the cursor in the main editor by pressing F1.

Check for Update

Here you can check if a new version of OptiPerl has been released.

If you are a registered customer, you are entitled to the registered new version, so login to your update page and download it. If you have lost your password, then go to the "Customers" section of Xarka Software's homepage at http://www.xarka.com/contact/registered.html



OptiPerl's customers have been getting free updates since version 2.

Custom skins

You can add your own background in the menus, editor and code explorer. Just convert your favorites pictures to windows bitmap format, rename them as following and place them in the path where you installed optiperl (for example c:\program files\optiperl).

- `OptiEditor.bmp` for the editor.
- `OptiExplorer.bmp` for the code explorer.

This works only in the registered versions.

About Options

OptiPerl's Options dialog has over 230 options (not including 5 text styles with 33 color syntax elements each) to customize every aspect of OptiPerl. These options are saved in the file `Settings.ini` in the Application Data Folder. OptiPerl does not use the Windows registry for anything except the users registration data.

Perl
Perl

Path To Perl
The path to perl executable. This should look something like `c:\perl\bin\perl.exe`.

@INC
Search directories for perl modules, separated with a colon. Used by optiperl to search for modules for the code explorer and code completion features. Changing this does not affect the PERL5LIB path.

Perl DLL
Path to perl DLL. Should look something like `c:\perl\bin\perl58.dll`. This is used for embedded perl in the regular expression tester and plug-ins.

Find Again
Try to get perl's path from the registry.

Guess
Create a default search path based on the path to perl.

Run Timeout
How long to wait for a script to execute until it is terminated.

Default perl extension
When saving a new perl script, select the default extension for it.

Windows
Docking Style

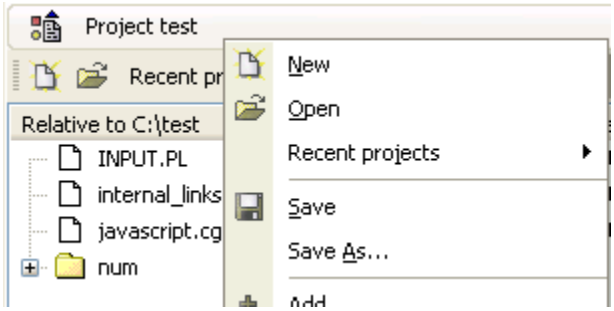
Visual style to apply to windows. The style "Themed" applies only to Windows XP and uses the selected theme.

Window Options

Allow docking windows with mouse only while Control is down
Normaly dragging a window over a container will grab and dock it by default, unless holding down at the same time the Control button. Selecting this option will reverse this behavior, e.g. docking will be enabled only if the Control button is down.

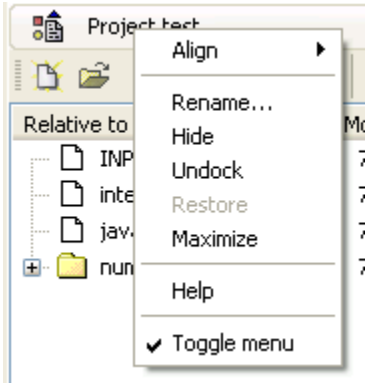
Show buttons in docked captions
Enables control buttons in docked windows.

Right-clicking on tabs displays context menu (Control-right click for window menu)
When right clicking on a docked window caption, if the window has a context menu, then it will be displayed, else a menu with window commands will be displayed. Unselecting this option reverses this behavior.



Context menu

However if the Control button is down when right clicking, then the window menu will be displayed.



Window menu

Show floating windows in task bar (requires restart)
Enables displaying of all the floating windows in the windows taskbar for easy selection.

Docked caption height
Height of docked windows captions.

Environment

Environment Settings

Allow only one instance of OptiPerl to run
Don't allow a second instance of OptiPerl to be loaded (recommended). This way double clicking a script in explorer will open it in the active OptiPerl session.

Show tips at startup
Show "Tip of the day" when OptiPerl starts.

Show tray bar icon
Enable the tray bar icon, which also has a pop-up menu when clicking the icon. The default menu contains the minimized windows of OptiPerl's desktop. This menu corresponds to the "Tray icon" menu, so you can customize it.

Enable extended keyboard support
Supports keyboards with extended keys like Microsoft (R) Natural or Internet keyboard.

Update registration
Update the license information for OptiPerl.

Reset Message Dialogs
Show again all the dialogs that had a "Don't show again" checkbox.

Recent Lists

Maximum items on recent lists
Maximum items on recent lists. These are the combo edits found in OptiPerl that list recent values, for example in the Find/Replace dialog, Run in Console etc.

Clear Recent lists
Clear recent list of files and folders used within OptiPerl.

Registered File Types

Status
Shows if associations with OptiPerl are OK.

Check automatically when OptiPerl starts
If selected, OptiPerl will check if its file types are associated with it.

Restore associations
Restore all associations that existed before OptiPerl's installation.

Associate again
Associate extensions with OptiPerl.

Code completion

Code completion - HTML code completion
Enable Code Completion features.

Enable hints
Enables hints in the editor when hovering the mouse over a statement. When debugging, the hints can be disabled by deselecting Debug Menu / Auto-Evaluation.

Use editor font for hints
If selected, the font used for hints will be the same as the editor's font.

Layouts

Enable standard layouts
Enable standard layouts (edit, debug and run). Layout will automatically change when selecting Run and Debug. Uses any layouts found named "Edit", "Debug", and "Run".

Code Librarian

Prompt before saving changes
Whether Code Librarian will prompt to save changes whenever you edit the active snippet. If deselected, changing between snippets will save any edits made without prompting.

Code Explorer

Select here a font type and style for the code explorer window.

Debugger

Check syntax before entering debugger
Always check syntax before entering debugger.

Increase gutter width before entering debugger
Automatically increase the gutter width to display better breakpoints, run line and bookmarks.

Alternative buffering
Select if you cannot start the debugger.

Live evaluation delay
Delay in milliseconds for live evaluation hint to appear (increase on slower systems).

Remote debugger - Listen at port
Port to listen for remote debugger. This is applied when selecting Debug/Listen for remote debugger. This number must match the PERLDB_OPTS environment variable on the remote machine.

Error Testing

Error testing warning level
The warning level when selecting "Test for errors".

Live syntax checking

Highlight errors & warnings in editor
Highlight lines with errors and/or warnings in the editor. Will also show the error text in the status box when the cursor is on the offending line. You can also select the color and the style of the lines drawn.

Show errors in code explorer
Show error lines in the code explorer under the "Errors" node.

Show warnings in code explorer
Show lines with warnings in the code explorer under the "Warnings" node.

Disable on paths
Don't do automatic syntax checking on the paths entered. The paths can include the standard perl libraries which are known to be error-free. If you are editing files over a slow network, or you are not allowed to use a large bandwidth very often, you can also enter here the mapped network drives and UNC paths. Example:

```
c:\perl\lib; c:\perl\site\lib; Y:\; \\computer1\C\;
```

Note also that all the subfolders will not be checked either.

Read more about Error Testing.

Running

When Running with a server

Host URL
Http address that will be called when the option "Run with Server" is set in the "Server" menu.

When Runnign remote files

Upload automatically before running
Upload the remote script before running in the browser (unless it has not been modified).

Replace the 'Host' field in headers sent when running
Will replace the "Host" field in request headers of the browser when running the remote file. The value for host is extracted from the URL address in the "links to" field of the corresponding remote transfer session. This is required in most cases. See "Running remote files" for more information.

Other options

Bring up browser after running a file
If checked, the internal browser will get focus after running a file.

Select secondary browser
Select here the executable of a third-party internet browser. If selected it will be added in the "Run" menu.

Keep transfer sessions open
Allows only one log-in per transfer session. This means that when working on a remote file, each time you upload, log-in will not be needed so the transfer will be faster and safer. You can control which sessions are open from the Remote Explorer window.

Session keep-alive interval
Interval in seconds between sending commands to the remove server, to keep alive the session.

Web servers

Internal Server Settings

Root folder
The root folder of the internal server. If enabled, the file being run must reside in this folder or a subfolder.

Associations
Shows currently associated extensions.

Aliases
Aliases used for internal server, column separated. For example:
`/cgi-bin/=c:\webroot\cgi-bin;`
Read more about Aliases.

When the internal server runs a script, add this value for PERLDB_OPTS
When the internal server is enabled, and you run a script while listening for a remote debugger, the internal server can run the script adding first in it's environment the setting needed to invoke the debugger. Change this only if you know what you are doing!

External Server Settings

Get from httpd.conf file
Will import an apache style httpd.conf file.

Access Log File
The access log file of the external server.

Error Log File
The error log file of the external server.

Executable
The filename only (without a path) of the server executable. Used by OptiPerl to check if it is loaded.

Consider external server loaded
If you are running an external server and OptiPerl cannot find it then check this option, so OptiPerl will manually consider the server loaded.

Document root
The document root path of the external server. This is usually the path that contains the html documents of the site.

Aliases
Aliases used for external server, column separated. For example:
`/cgi-bin/=c:\webroot\cgi-bin;`
Read more about Aliases.

Backups

Create backups when saving files
Enables backups. Disabling this option will not create any backups.

Enable zip compression
Allows saving backups in zip archives (see below for implementation)

Filename scheme:
Full path (relative to file or absolute) to file or zip archive that will contain the resulting backups. Note that backup files are always ovewritten in the resulting folder or zip archive without warning.

The path, zip archive (if enabled above) and filename can include the following tags:

%F	Filename (without extension)
%E	Extension
%P	Project filename
%Y	Year (4-digits)
%M	Month (2-digits)
%D	Day (2-digits)
%H	Hour (2-digits)
%N	Minute (2-digits)
%S	Second (2-digits)
%J	Path where project is saved (without ending path delimiter)
%0	Project Data 0
...	.....
%9	Project Data 9

Filename scheme for files in projects
Just like above, however this scheme is applied when a project is open and the file being edited belongs to it.

Examples

Assume you are editing file `c:\webs\test\script.pl`

1) Backups.zip\%F.%Y-%M-%D.%E

With this file scheme, each time you save your file, the zip archive Backups.zip will be created or opened in
`c:\webs\test` and in it will be written files like `script.2003-03-01.pl`

2) **Backups\%F.%Y-%M-%D.%E**

A folder named Backups will be created in `c:\webs\test\backups`, and files like `script.2003-03-01.pl` will be added

3) **..\Backups\%F.%Y-%M-%D.%E**

Like above, but the backups folder will be c:\webs\backups

4) **c:\Backups\%F.%Y-%M-%D.%E**

Like above, but the backups folder will be c:\backups

5) **c:\Archive\Work\%P\Archive-%Y.zip\%F.%Y-%M-%D.%E**

Assuming we have a project named MyProject, this will create in the folder `c:\Archive\Work\MyProject\` the zip file named `Archive-2003.zip` and write the files like `script.2003-03-01.pl in it.`

Editor
Editor Settings

Startup editor
Default startup of the editor.

Font
Font name, color and size of the editor. This affects all editor windows in OptiPerl.

Selection
Font color and background color when selecting text.

Fixed width Font
Whether only monospaced (courier like) fonts will be used. What is important with monospace fonts is that column widths are always the same in every line of the editor.

Font smoothing
Whether the windows default for font smoothing will be applied to the editor's font. Leave unselected on slower computers, and enable only on very fast computers, especially if you have "ClearType" font smoothing enabled in Windows XP. This affects dramatically the redrawing speed of the editor.

Bracket Highlighting

Auto highlight brackets
Moving the cursor and typing brackets will automatically highlight the corresponding bracket.

Highlight brackets with mouse
Allow highlighting corresponding brackets when moving the mouse while the Control key is pressed.

Code folding

Code folding enables hiding sections of your script enclosed by brackets, parenthesis, Here-Document and POD statements. Read more about code folding.

Tabs

Use tab character
Whether a real tab character will be used when pressing tab and saving. If not used, then spaces will be used for padding. This also affects how indent and outdent block commands pad lines.

Smart tab
When tab is pressed, the cursor is moved to the same column as the next non-whitespace character of the preceding line. This cannot be used at the same time with "use tab character", and if enabled the tab stop column numbers will not be obeyed. For example:

```
if (1)      {
   |        |
```

If you were on the second line on column 1, each tab press would go to the | representing the first non-whitespace character of the first line.

Cursor always on tabs
Pressing right and left arrows will move on tabs in the editor.

Tab stops
Selects the columns that the cursor will position onto each time tab is pressed. The difference of the last two numbers is used for calculating tabs if the cursor is on a column after the largest number. Also note that these numbers are not used if "smart tabs" are enabled.

Block indent
Contains the indent that is applied to the text edited when you select Indent/Outdent.

Visualise Tabs
Tab lines are vertical lines denoting the tabs in the whitespace in front of each line of your script. You can select here their appearance. To be visible, you must also have the option "Use tab character" enabled.

Lines

Auto indent mode
Add the indent from the previous line when pressing enter for a new line.

Backspace unindents
Backspace will delete all spaces of an indent.

Whitespace
Controls how whitespace (spaces & tabs) at the ends of lines is handled.

Never trim: Leaves whitespace as is.
Trim only in non-empty lines: Trims whitespace only in lines that are not empty (already have visible characters).
Trim always: Trims whitespace always.

Editor behavior

Show line numbers
Show line numbers of file.

Show line numbers on gutter
Show line numbers on gutter. The above must also be selected for this to work.

Group undo
Will undo actions at a line-by-line basis (else character by character).

Cursor beyond EOF
Allow cursor to move beyond the end of the script.

Cursor beyond EOL
Allow cursor to move beyond the end of each line.

Selection beyond EOL
Allow selections beyond the end of each line.

Enable triple Home & End keys
Enable triple HOME and END keys.

Word wrap
Enable word wrap. Cannot be used with Code Folding.

Persistent blocks
Selected blocks will not be unselected with cursor movements.

Overwrite blocks
Selected blocks will be overwritten when a character is pressed.

Double click line
Double clicking on a line will select it.

Find text at cursor
Allow the default text in the "Find" dialog to be the text under the cursor.

Show special symbols
Show EOL and EOF characters.

Overwrite cursor as block
Selected blocks will be overwritten when a character is pressed.

Disable dragging
Disable dragging selected blocks.

Highlight URLs
Will highlight text in the editor that looks like URLs. Clicking will navigate to them.

Display file extensions in tabs
Displays the extensions of the files opened in the editor. Useful if you often open many files that share the same name with a different extension.

Close tab on double click
Enables closing a file in the editor by double-clicking on it's tab.

Multiline tabs
Whether the tabs of the files in the editor will be displayed in many rows, if all cannot fit in one row.

Delimiters
Characters that set bounds to the selection when double clicking on a word.

Editor appearence

Editor margin
The editor margin is a vertical line positioned after the number of characters in the "position" edit box. When the option "Word Wrap" is selected, then the lines also wrap here, unless the option "Word wrap on margin" is unchecked. In this case words are wrapped at window width.

Line Highlight
Line highlight is a box or line that notes the line being edited.

Editor gutter
The editor gutter is the right-most part of the editor, that usually contains the line numbering and icons denoting bookmarks and debugging symbols. Because it is heavily used when debugging, you have the option to widen it automatically when debugging (see the debugging section).

Line Separator
Line separators are horizontal lines separating each line of your text.

Editor defaults

Default Script Folder
The default folder to load perl scripts from.

Default Perl Template
Default template for perl scripts.

Default Html Folder
The default folder to load html documents from.

Default HTML Template
Default template for html documents.

Default line ending format
Default line ending for saving files.

Printer

Margins
Page margin when printing.

Scale font to fit x lines per page
How many lines per page to print. This affects the font size used.

Font
Font used when printing

Syntax coding
Enable syntax highlighting when printing (overrides setting on Syntax Coding page).

Print line coding
Print line coding (overrides setting in Line Coding page).

Print box coding
Print box coding (overrides setting in Box Coding page).

Convert all patterns to solid
Override background patterns you have selected in box and line coding to a solid coloring. This may be necessary when printing at a low DPI or draft.

Override line widths with x
Override the settings you have for line widths with another setting. 1 or 2 pixel line widths may not show up well when printing, so you may need to select this setting and enter a larger number here.

Compatible mode
If you have enabled "Print box coding" and some of the text is not printed on paper, then select this option. Deselect it for a small improvement in the boxes printed.

Read more about Printing.

Syntax coding
Syntax Coding

Use syntax coding
Enable / disable syntax coding

Color code declared Identifiers
Color code with style "Perl Declared Identifier", variables and subroutines that have been declared in the script.

Variable differentiation in strings
Enables syntax coding variables even when used in strings.

Active text style
Active text style is the syntax coding style currently loaded. This may also be changed from the sub menu "Color coding" when editing.

Text style pages
On the text style pages you can select the font properties for all kinds of elements in scripts and programs.

Background color
Note that the background color selector also has a check mark. This is needed to override the default color of the editor if you want to.

Text style name
Name of the style. This changes the text in the sub menu "Color coding"

Reset style
Resets style to the default.

Copy from first style
Copies all properties from the first style.

Previewing changes on styles
While creating your own styles, we recommend pressing the Apply button, and moving the Options dialog so you can view the editor in the background.

Read more about Syntax Coding.

Box & Line coding

Line Coding

Line coding can be used only for brackets. From the page "Lines" you can select a line style for each level of brackets. Read more about Box & Line coding.

Box Coding

Box coding can be used only for brackets, parenthesis, Here-Document and Pod statements.

- Box style of POD and Here-Documents can be selected from the Level coding page.
- Box styles for each bracket { } level can be selected from the "Bracket Boxing" page.
- Box styles for each parenthesis ( ) level can be selected from the "Parenthesis Boxing" page.

Read more about Box & Line coding.

Improving performance

To improve your experience with OptiPerl, you may want to change some settings depending on your computer's speed and memory. In general, all of OptiPerl's code that affect the editor, like syntax parsing, box & line code, code folding, are optimized for speed in expense of memory, so a system with over 64 MB of RAM is recommended. However all the graphics are drawn lightning fast; many techniques have been used to increase performance.

Options affected by processor speed

The following options are affected from your computer's speed and graphics card. If you notice an unappealing delay in the redrawing of the editor, you may want to disable one or more of them. We recommend disabling in the order given for a smaller trade-off.

1) Disable Font Smoothing in Options / Editor / Font box

Many resources are used for font smoothing (anti-aliasing), especially if you have Windows XP and have selected the "ClearType" font smoothing method in the display properties. So the first thing you should do to greatly increase performance is to disable it. You might need to restart OptiPerl for this to be applied. The default in OptiPerl is disabled under Win 2000 / XP and enabled under Win 98 / Me / Nt since in these versions this does not affect the speed too much.

2) Box & Line coding from Options / Level coding

These also need a lot of resources. We recommend disabling if your processor is under 500 mz.

In most cases doing 1 and 2 above will make the editor extremely fast, even on very slow processors. However if your computer is extremely slow (< 300mz), then also do the following:

- Disable monospace fonts from Options / Editor / Font box
- Remove code folding from Options / Editor / Code folding
- Remove Tab lines from Options / Editor / Tab handling
- Remove Other "cosmetic" options in Options / Editor / Visual
- Remove Live syntax checking from Options / Error testing
- For very slow processors, remove color syntax coding, from Options / Syntax coding

A good test to see how fast the editor redraws, is to open a large script, and then scroll through it by holding down the Page Up or Down buttons.

More performance options

Under Options / Environment II / Performance options, there are some options that also affect performance:

- *Error check delay*. Delay time in milliseconds of keyboard inactivity before the automatic error checking initiates (default 1000 ms = 1 second). Note that while perl is loading and checking the code, you can still edit, since the checking takes place in a low priority thread.

- *Explorer check delay*. Delay time in milliseconds of keyboard inactivity before the code is parsed to be used in code explorer, box & line coding and code folding (def. 100 ms =

1/10 second). Note that while the code is being parsed, you can still edit, since updating takes place in a low priority thread.

- *Line & Box lookahead*. On each redraw of the editor, how many lines of code forward or backwards of the viewable code should be checked for long boxes or lines. On fast computers, over 800mz, you can increase these numbers to arbitrary large numbers. Note that due to a limitation on windows 98/me systems these numbers have no affect after 3000 lines. On windows nt/2000/xp, this is not a limitation. You can put an opening bracket on line 1 of your code, and if the closing bracket is on line 5000, then a huge line will be drawn from line 1 to line 5000. The defaults are 100 lines for Line coding and 1000 lines for box coding.

Options affected by amount of computers memory

The following options, found in Performance options dialog, have to do with the amount of RAM installed. Increase or decrease to change the memory footprint of Optiperl. We recommend increasing if you have over 256mb installed:

- *Max search results.* Maximum results logged in code explorer after a regular expression search or replace (def. 20000 items).
- *Max undo levels.* Maximum undo levels when editing (def. 100).

Application data folder

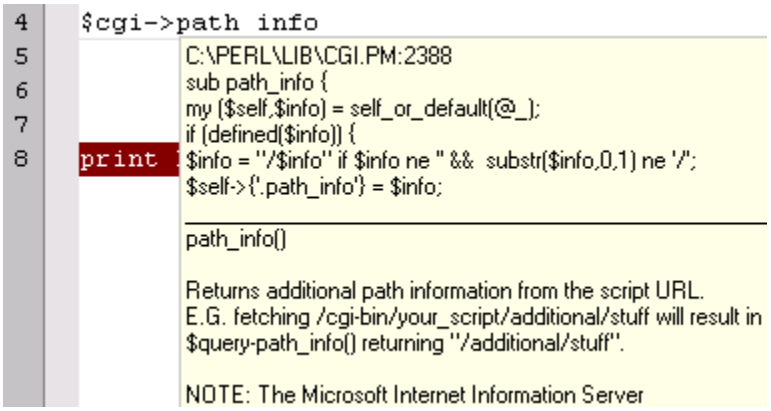
OptiPerl uses a folder to store its settings and cached files. The application data folder has 3 sub-folders:

- **Sessions**: Saves a cached copy of remote files. The files are saved in sub folders named after the session of the remote transfer.
- **Debug**: Saves cached copies of scripts that are Remotely debugged. The scripts are saved in folders named after the IP of the remote machine, for example Remote\127.0.0.1\test.pl
- **Settings**: Stores files with the settings of OptiPerl.

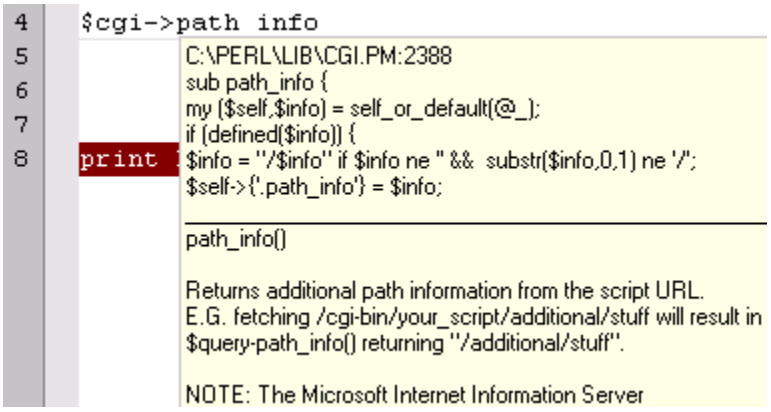
The application data folder is usually at:

Windows 98/Me: `C:\windows\All Users\Application Data\OptiPerl\`
Windows NT: `c:\WinNT\Profiles\{Active User}\Application Data\OptiPerl\`
Windows 2000/XP: `C:\Documents and Settings\{Active User}\Application Data\OptiPerl\`

For security reasons, we recommend setting the Encrypted properly of the folder in Windows 2000 and XP.



You can change this folder from Options / Environment II. If you access this folder a lot, you can move it to a simpler to access location, like in the root or in "My Documents".



In multi user versions of OptiPerl, each user of OptiPerl can have his own Application Data folder.

Other Settings

OptiPerl in it's installation folder has among other the following files that control settings not found in OptiPerl's dialogs:

- `Parsers.xml`: Control's settings that affect the associations of file extensions and default and non-default language parsers. Used for syntax coding.
- `Server.ini`: Settings for the internal server.
- `Default Modes.txt`: Affects default transfer modes and permissions when adding a file to the Project Manager.
- `Default Watches.txt`: Standard watches in the watches window.
- `Buttons.bmp`: The actual bitmaps of the custom buttons added in the Custom Tools.
- `Startpage.htm`: The html code to the page in OptiPerl's browser when it starts.
- `File Dialog Filters.txt`: Contains the "Files of type" filtering items in the editors open and save dialog. Needs to have an even number on non empty lines, and each pair of lines contains the caption of the filter and the filter itself (as a set of wild cards).
- `Code Librarian.zip`: Is copied to the Application Data Folder when creating a user's session. Used for multi-user versions of OptiPerl. This is NOT where your code snippets are stored. They are stored in the `Application Data Folder\Settings\Code Librarian.zip`
- `Internal Functions.txt`: List of internal functions of perl used for syntax highlighting.
- `Reserved words.txt`: List of reserver words of perl used for syntax highlighting.
- `Heredoc Ignore.txt`: Here-document headers to ignore in syntax highlighting. Enter here common headers used when autoloading perl code so the code will not be parsed as string.

Edit these files only if you know what you are doing, and make sure you keep backups! These files generally do not need to ever be changed.

Also read about the Application Data folder.

No help available for this item.

Transfer session type.

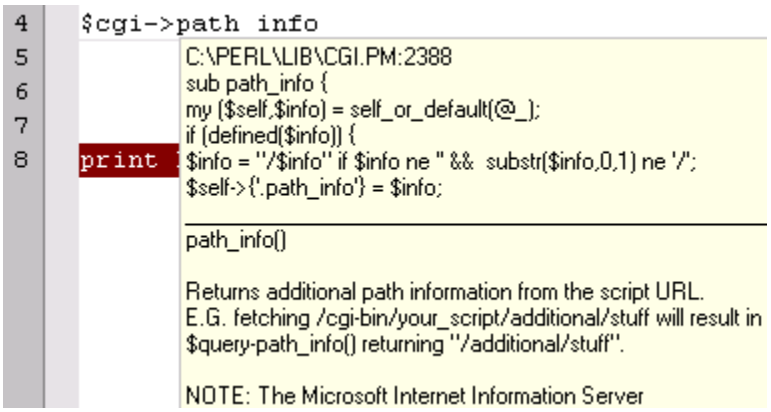
Account sent to the FTP server with the ACCT command. Not used normally.

Specify additional linking from absolute paths of the server to url addresses. For example:
`/usr/local/apache/cgi-bin/=http://www.mysite.com/cgi-bin/;`

Whether to convert the version to SERVER when uploading and to LOCAL when downloading. Read more about the version converter.

Whether to change the shebang when uploading.

A database navigator:



"+"	Adds a new record
"-"	Deletes a record
"Up arrow"	edits the selected record
"v"	Saves changes from a previous edit
"x"	Cancels changes from a previous edit

Displays remote sessions used.

User notes about the session.

Passive transfers.

Password for the remote session. This will only be saved if you select "Save passwords".

Port of the FTP Server. Usually 21.

Firewall password.

Firewall port.

Firewall server.

Firewall type.

Firewall username.

Whether to save the two passwords fields above.

Server of the remote machine. Should look something like:
`ftp.mysite.com`

Name of transfer Session. The session identifies the transfer settings in the "Open remote files" dialog, "Remote Explorer" and "Project options".

Shebang line this remote server uses. Used when uploading CGI scripts. It should look something like:
`#!/usr/local/bin/perl`

Document root of the server. Usually the path were all html documents are stored and can be accessed from a URL address. For example,
`/usr/local/apache/`

Read more about remote running.

How the document root links to a http address. Used when running a remote file. For example, `http://www.mysite.com/`

Read more about remote running.

Username for the remote session.

No help available for this item.

If many files are open in the editor, enable multiple rows with the tabs of the files.

Controls how whitespace (spaces & tabs) at the ends of lines is handled.

Never trim: Leaves whitespace as is.
Trim only in non-empty lines: Trims whitespace only in lines that are not empty (already have visible characters).
Trim always: Trims whitespace always.

When word wrap is enabled, the text will wrap at the position above. If disabled, the text will wrap at window width.

Interval in seconds between sending commands to the remove server, to keep alive the session.

Enables closing a file in the editor by double-clicking on its tab.

Enables backups. Disabling this option will not create any backups. For more information, read about backups.

Allows saving backups in zip archives. For more information, read about backups.

Full path (relative to file or absolute) to file or zip archive that will contain the resulting backups. Note that backup files are always ovewritten in the resulting folder or zip archive without warning. For more information, read about backups.

Examples
Assume you are editing file c:\webs\test\script.pl

```
1) Backups.zip\%F.%Y-%M-%D.%E
```

With this file scheme, each time you save your file, the zip archive Backups.zip will be created or opened in c:\webs\test and in it will be written files like script.2003-03-01.pl

```
2) Backups\%F.%Y-%M-%D.%E
```

A folder named Backups will be created in c:\webs\test\backups, and files like script.2003-03-01.pl will be added

```
3)  ..\Backups\%F.%Y-%M-%D.%E
```

Like above, but the backups folder will be c:\webs\backups

```
4) c:\Backups\%F.%Y-%M-%D.%E
```

Like above, but the backups folder will be c:\backups

```
5) c:\Archive\Work\%P\Archive-%Y.zip\%F.%Y-%M-%D.%E
```

Assuming we have a project named MyProject, this will create in the folder c:\Archive\Work\MyProject\ the zip file named Archive-2003.zip and write the files like script.2003-03-01.pl in it.

Full path (relative to file or absolute) to file or zip archive that will contain the resulting backups. Note that backup files are always ovewritten in the resulting folder or zip archive without warning. This scheme is applied when a project is open and the file being edited belongs to it. For more information, read about backups.

Examples
Assume you are editing file c:\webs\test\script.pl

```
1) Backups.zip\%F.%Y-%M-%D.%E
```

With this file scheme, each time you save your file, the zip archive Backups.zip will be created or opened in c:\webs\test and in it will be written files like script.2003-03-01.pl

```
2) Backups\%F.%Y-%M-%D.%E
```

A folder named Backups will be created in c:\webs\test\backups, and files like script.2003-03-01.pl will be added

```
3)  ..\Backups\%F.%Y-%M-%D.%E
```

Like above, but the backups folder will be c:\webs\backups

```
4) c:\Backups\%F.%Y-%M-%D.%E
```

Like above, but the backups folder will be c:\backups

```
5) c:\Archive\Work\%P\Archive-%Y.zip\%F.%Y-%M-%D.%E
```

Assuming we have a project named MyProject, this will create in the folder c:\Archive\Work\MyProject\ the zip file named Archive-2003.zip and write the files like script.2003-03-01.pl in it.

Based on the data entered above, this box displays a description of the job that would be done if the currently open file was saved. For more information, read about backups.

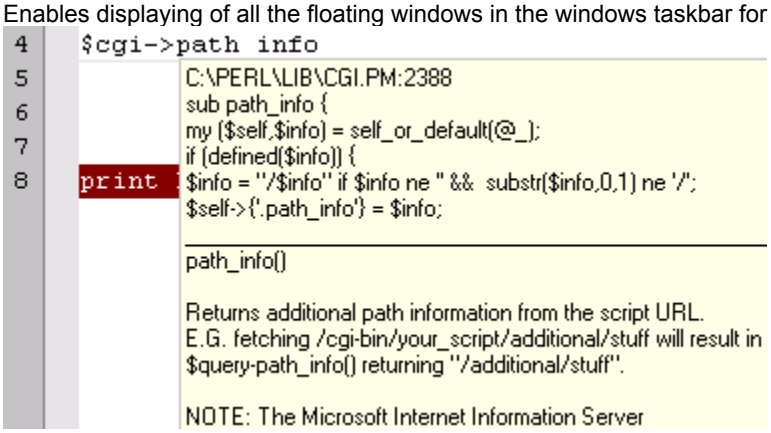
Allows only one log-in per transfer session. This means that when working on a remote file, each time you upload, log-in will not be needed so the transfer will be faster and safer. You can control which sessions are open from the Remote Explorer window.

Normaly dragging a window over a container will grab and dock it by default, unless holding down at the same time the Control button. Selecting this option will reverse this behavior, e.g. docking will be enabled only if the Control button is down.

Enables control buttons in docked windows.

When right clicking on a docked window caption, if the window has a context menu, then it will be displayed, else a menu with window commands will be displayed. Unselecting this option reverses this behavior.

Enables displaying of all the floating windows in the windows taskbar for easy selection.



Height of docked windows captions.

Visual style to apply to windows. The style "Themed" applies only to Windows XP and uses the selected theme.

Path to perl DLL. Should look something like c:\perl\bin\perl58.dll. This is used for embedded perl in the regular expression tester and plug-ins.

The access log file of the external server. This is the file that will be viewed when selecting Server / View access logs.

Closes the open file when double-clicking on its tab in the editor

Enables syntax coding variables even when used in strings

Selected active text style. This may also be changed from the sub menu "Color coding".

Whether the windows default for font smoothing will be applied to the editor's font. Leave unselected on slower computers, and enable only on very fast computers, especially if you have "ClearType" font smoothing enabled in Windows XP. This affects dramatically the redrawing speed of the editor.

Sets the Application Data folder.

Shows if associations with OptiPerl are OK.

Shows currently associated extensions.

Add the indent from the previous line when pressing enter for a new line.

Background color.

Backspace will delete all spaces of an indent.

Allow cursor to move beyond the end of each line.

Contains the indent that is applied to the text edited when you select Indent/Outdent.

Background pattern of this level.

Color of this level.

Curving of the rectangle at this level.

Box visible in this level.

Box brackets.

Enable box background coding.

Box here documents.

Box parenthesis.

Box embedded pod.

Background pattern for boxes.

Color for boxes.

If checked, the browser will get focus after running a file.

Add new association.

Apply style.

Clear recent list of files and folders used within OptiPerl.

Copy color settings from the 1st style.

Default associations (pl, pm, plx, cgi).

Edit association.

Edit this style.

Try to get perl's path from the registry.

Create a default search path based on the path to perl.

Remove association.

Show again all the dialogs that had a "Don't show again" checkbox.

Reset all styles to their defaults.

Reset this style to its default.

Restore all associations that existed before OptiPerl's installation.

Associate extensions with OptiPerl.

Update the license information for OptiPerl.

If selected, OptiPerl will check if its file types are associated with it.

Override the background of the editor with the selected color.

Always check syntax before entering debugger.

Whether Code Librarian will prompt to save changes whenever you edit the active snippet. If deselected, changing between snippets will save any edits made without prompting.

Enable Code Completion features. Read more about code completion.

Enable hints in the editor. Read more about code completion.

Enable HTML code completion features. Read more about code completion.

Font used in the Code explorer window

Font size used in the Code explorer window

The folder that stores common settings of OptiPerl, which should be shared between users, like common user tools. You can also specify a UNC path.

Allow cursor to move beyond the end of the script.

Pressing right and left arrows will move on tabs in the editor.

Double clicking on a line will select it.

Select if you cannot start the debugger.

Default template for html documents.

The default folder to load html documents from.

Default line ending for files when creating a new file. Read more about Creating Files.

Default template for perl scripts.

The default folder to load perl scripts from.

Select here a default extension when saving Perl scripts

Disable dragging selected blocks.

Displays the extensions of the files opened in the editor. Useful if you often open many files that share the same name with a different extension.

Allow highlighting corresponding brackets when moving the mouse while the Control key is pressed.

Moving the cursor and typing brackets will automatically highlight the corresponding bracket.

Background color of the editor.

Characters that set bounds to the selection when double clicking on a word.

Text style name.

List of code elements.

Enable triple HOME and END keys. The sequence for HOME is first column, first line on screen, first line of file.

The error log file of the external server. This is the file that will be viewed when selecting Server / View error logs.

Aliases used for external server, column separated. For example:
`/cgi-bin/=c:\webroot\cgi-bin;`

Read more about Aliases.

The filename only (without a path) of the server executable. Used by OptiPerl to check if it is loaded.

The document root path of the external server. This usually is the path that contains the html documents of the site.

Allow the default text in the "Find" dialog to be the text under the cursor.

Enable bracket folding.

When loading a script, have folded by default.

Enable code folding. Cannot be used at the same time with word wrap.

Color of the folding column.

Enable folding "here-document" text.

Whether the last line of folded blocks will be visible or not. If enabled then the ending line that terminates the block will also be hidden when the block is folded.

Enable parenthesis folding.

Enable embedded pod folding.

Font for editing.

Font size used for editing.

Font color.

Will import an apache style httpd.conf file.

Will undo actions at a line by line basis (else character by character).

Gutter color.

Pattern for the gutter.

Allow the gutter to be visible.

Gutter width.

Highlights all URL's in the editor. Clicking will also navigate to them.

If selected, the font used for hints will be the same as the editor's font.

Http address that will be called when the option "Run with Server" is set in the "Server" menu.

Automatically increase the gutter width to display better breakpoints, run line and bookmarks.

Aliases used for internal server, column separated. For example:
`/cgi-bin/=c:\webroot\cgi-bin;`

Read more about Aliases.

Supports keyboards with extended keys like Microsoft (R) Natural or Internet keyboard.

Enable standard layouts (edit, debug and run). Layout will automatically change when selecting Run and Debug. Uses any layouts found named "Edit", "Debug", and "Run". Read more about layouts.

Line highlight background color.

Line highlight background pattern.

Line highlight color.

Line highlight pen style.

Shows line highlight. This affects the active line in the editor.

Line highlight width.

Color of line for this level.

Pen style for this level.

Line visible for this level.

Line width for this level.

Enable line coding for brackets.

Show line separators

Line separator color.

Line separator pen style.

Line separator width.

Delay in milliseconds for live evaluation hint to appear (increase on slower systems).

Margin line color.

Character position of the margin line.

Margin line style.

Allow the margin line to be visible.

Margin line width.

Don't allow a second instance of OptiPerl to be loaded (recommended). This way double clicking a script in explorer will open it in the active OptiPerl session.

Affects cursor movement within blocks.

Selected blocks will be overwritten when a character is pressed.

The path to perl executable. This should look something like c:\perl\bin\perl.exe. You will need to fill in this path if you get a "Cannot find perl" error.

Options for fine-tuning performance.

When the internal server is enabled, and you run a script while listening for a remote debugger, the internal server can run the script adding first in its environment the setting needed to invoke the debugger. Change this only if you know what you are doing!

Search directories for perl modules, separated with a colon. Used by optiperl to search for modules for the code explorer and code completion features. Changing this does not affect the PERL5LIB path. Read more about how OptiPerl finds modules.

Selected blocks will not be unselected with cursor movements.

Opens a script in the editor to help preview changes.

Print box coding.

If you have enabled "Print Box Coding" and some of the text is not printed on paper, then select this option. Deselect it for a small improvement in the boxes printed.

Font used when printing.

Print line coding.

How many lines per page to print. This affects the font size used.

Page margin when printing.

Override background patterns you have selected in box and line coding to a solid coloring. This may be necessary when printing at a low DPI or draft.

Override the settings you have for line widths with another setting. 1 or 2 pixel line widths may not show up well when printing, so you may need to select this setting and enter a larger number here.

Enable syntax highlighting when printing.

Maximum items on dialogs that contain combo boxes with previous selections, like the find/replace dialog and other.

Port to listen for remote debugger. This is applied when selecting Debug/Listen for remote debugger. This number must match the PERLDB_OPTS environment variable on the remote machine.

The root folder of the internal server. If enabled, the file being run must reside in this folder or a subfolder.

Will replace the "Host" field in request headers of the browser when running the remote file. The value for host is extracted from the URL address in the "links to" field of the corresponding transfer session. This is required in most cases. See "Running remote files" for more information.

Upload the remote script before running in the browser (unless it has not been modified).

How long to wait for a script to execute until it is terminated.

Request a file using the http protocol. If unchecked then the file is loaded as a file url.

Color code with style "Perl Declared Identifier", variables and subroutines that have been declared in the script.

Select here the executable of a third-party internet browser. If selected it will be added in the "Run" menu.

Background color when selecting text in the editor

Font color when selecting text in the editor

Allow selections beyond the end of each line.

Highlight lines with errors in the editor.
Will also show the error text in the status box when the cursor is on the offending line.

Read more about Error Testing.

Highlight lines with warnings in the editor.
Will also show the error text in the status box when the cursor is on the offending line.

Read more about Error Testing.

Color of the lines drawn that highlight errors or warnings.

Style of the lines drawn that highlight errors or warnings.

Show error lines in the code explorer under the "Errors" node.

Read more about Error Testing.

Show lines with warnings in the code explorer under the "Warnings" node.

Read more about Error Testing.

Show line numbers in the editor.

Show line numbers on gutter. The setting "show line numbers" must also be enabled to view the lines.

Show EOL and EOF characters.

Show "Tip of the day" when OptiPerl starts.

Don't do automatic syntax checking on the paths entered. The paths can include the standard perl libraries which are known to be error-free. If you are editing files over a slow network, or you are not allowed to use a large bandwidth very often, you can also enter here the mapped network drives and UNC paths. Example:

```
c:\perl\lib; c:\perl\site\lib; Y:\; \\computer1\C\;
```

Note also that all the subfolders will not be checked either.

When tab is pressed, the cursor is moved to the same column as the next non-whitespace character of the preceding line. This cannot be used at the same time with "use tab character", and if enabled the tab stop column numbers will not be obeyed. For example:

```
if (1)      {
    |       |
```

If you were on the second line on column 1, each tab press would go to the | representing the first non-whitespace character of the first line.

Default startup of the editor.

Enable syntax highlighting.

Whether a real tab character will be used when pressing tab and saving. If not used, then spaces will be used for padding. This also affects how indent and outdent block commands pad lines.

Color of the tab lines.

Selects the columns that the cursor will position onto each time tab is pressed. The difference of the last two numbers is used for calculating tabs if the cursor is on a column after the largest number.

Pen style of the tab lines.

Show tab lines in the editor.

Width of the tab lines.

Enable tainting checks.

The folder that stores file templates for File / New / Choose from template. You can also specify a UNC path.

Enable the tray bar icon that also has a pop-up menu when clicking the icon. The default menu contains the minimized windows of OptiPerl's desktop. This menu corresponds to the "Tray icon" menu, so you can customize it.

If you are running an external server and OptiPerl cannot find it then check this option, so OptiPerl will manually consider the server loaded.

Allow only fixed width (monospace) fonts. This is recommended.

Displays categories of OptiPerl's options. Click on an arrow symbol to expand categories.

The warning level when selecting "Test for errors".

Enable word wrap. Not recommended because code folding cannot work at the same time.

No help available for this item.

The installation directory of OptiPerl is where you installed it. For example `c:\program files\OptiPerl\`
This folder contains except the executable of OptiPerl, other files. See Other Settings.

No help available for this item.

Press to setup your remote sessions.

If this option is enabled, each time you open this project, the files that where opened the last time you were using this project will be restored.

Data fields replaced when running user tools.

How to show files that are out of the starting folder and are not published. If you try to add files that are out of the "Local starting folder" above, these cannot be published, but will be displayed in the project either under a folder called "not published" or in the root of the project.

Internal server root path. This overrides the setting in the "Server" menu when this project is loaded.

Layout loaded when loading project. Select "none" for no modification each time the project is opened. Read more about layouts.

Parse all "use Lib (...)" from the project's files. Read more about how optiperl finds modules.

Starting local path. This folder should contain all the files and subfolders that you want to publish. Read more about projects.

If selected, all options on the default tab will be used instead of the corresponding in the main options dialog (under menu Tools).

Starting remote path. This is the path on the remote server that corresponds to the local path above. For example:

```
/
/cgi-bin/
```

Read more about projects.

Selected remote transfer session.