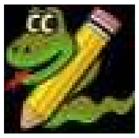
PyScripter - a Python IDE

PyScripter originally started as a lightweight IDE designed to to serve the purpose of providing a strong scripting solution for Delphi applications, complementing the excellent Python for Delphi (P4D) components. However, and with the encouragement of the P4D creator Morgan Martinez and a few early users, it has now evolved into a full-featured stand-alone Python IDE. It is built in Delphi using P4D and the SynEdit component but is extensible using Python scripts. Currently, it is only available for Microsoft Windows operating systems and features a modern user-interface. Being built in a compiled language is rather snappier than some of the other IDEs 🙂 and provides an extensive blend of features that make it a productive Python development environment.



Why yet another Python IDE?

There are many Python Integrated Development Environments around. And quite a few good ones, for example Spyder, SPE and Eric3, not to mention IDLE which is included in the standard Python distribution. So it is reasonable to ask why bother to develop yet another Python IDE. The short answer is for the fun of it! The long answer relates to the ambition to create a Python IDE that is competitive with commercial Windows-based IDEs available for other languages.

Main Features:

- Syntax Highlighting Editor
    - Unicode based
    - Full support for encoded Python source files
    - Code folding
    - Brace highlighting
    - Python source code utilities ((un)tabify, (un)comment, (un)indent, etc.)
    - Code completion and call tips
    - Code and debugger hints
    - Syntax checking as you type
    - Context sensitive help on Python keywords
    - Parameterized Code Templates
    - Accept files dropped from Explorer
    - File change notification
    - Converting line breaks (Windows, Unix, Mac)
    - Print preview and print syntax highlighted Python code
    - Syntax highlighting of HTML, XML and CSS files
    - Split view file editing
    - Firefox-like search and replace
    - Side-by-side file editing
    - Code Completion
    - Call Tips
    - Work with remote files
- Integrated Python Interpreter
- Integrated Python Debugging
    - Remote Python Debugger
    - Thread debugging
    - Call Stack

  - Variables Window
  - Watches Window
  - Conditional breakpoints
  - Debugger hints
  - Post-mortem analysis
  - Can run or debug files without first saving them
  - Run/debug scipts remotely on Windows and Linux servers
- Editor Views
  - Disassembly
  - HTML Documentation (pydoc)
- Code Explorer
- File Explorer
  - Easy configuration and browsing of the Python Path
  - Integrated Version Control using Tortoise GIT, Tortoise SVN or Tortoise CVS
- Project Explorer
  - Import existing paths
  - Multiple run configurations
- Integrated Unit testing
  - Automatic generation of tests
  - Unit testing GUI
- Access to Python manuals through the Help menu
- To Do List
- Find and Replace in Files
- Integrated regular expression testing
- Choice of Python version to run via command line parameters
- Run Python Script externally (highly configurable)
- External Tools (External run and capture output)
  - Integration with Python tools such as PyLint, TabNanny, Profile etc.
  - Powerful parameter functionality for customized external tool integration

- Find Definition/Find references
- Find definition by clicking and browsing history
- Modern GUI with docked forms and configurable look&feel (themes)
- Persistent configurable IDE options

History

Version 3.6 (January 12, 2019)

- *New features:*
  - Much faster Remote Engine using asynchronous Windows named pipes if pywin32 is available.
  - IDE option to force the use of sockets for connection to the Python server now defaults to False
  - Enhancements to the SSH Engine- now compatible with PuTTY
  - Execute system commands in the interpreter with ! - supports parameter substitution
  - Clickable status panels with Python version and engine type
  - Text drag & drop between PyScripter and other applications (#554)
  - Triple-click selects line and Quadraple-click selects all
  - Double-click drag selects whole words - triple-click drag selects whole lines
  - Consistent syntax color themes accross supported languages (#855)
  - New IDE option "Trim trailing spaces when saving files" (#667)
  - New IDE Option 'Step into open files only'. Defaults to False. (#510)
  - Localization of the installer
- *Issues addressed:*
  - *#624, #743, #857, #904, #922, #927, #928, #929, #936*

Version 3.5 (November 15, 2018)

- *New features:*
  - Work with remote files from Windows and Linux machines as if they were local
  - Run/debug scipts remotely on Windows and Linux servers
  - Python 3 type hints used in code completion
  - Connection to python server with Windows named pipes. Avoids firewall issues. Requires the installation of pywin32 (pip install pywin32).
  - IDE option to force the use of sockets for connection to the python

    - server. (default True)
    - New Editor commands Copy Line Up/Down (Shift+Alt+Up/Down) and Move Line Up/Down (Alt + Up/Down) as in Visual Studio
    - PyScripter icons given a facelift by Salim Saddaquzzaman
    - Upgraded rpyc to 4.x. As a result Python 2.5 is no longer supported.
- *Issues addressed:*
    - *#501, #682, #907*

Version 3.4.2 (September 9, 2018)

- *New features:*
    - New Edit Command Read Only (#883)
    - Files opened by PyScripter from the Python directory during debugging are read only by default to prevent accidental changes.
    - Close All to the Right Editor command added (#866)
    - New editor parameter [$-CurLineNumber] (#864)
    - New IDE Option "File Explorer background processing'. Set to false if you get File Explorer errors.
    - Console output including multiprocessing is now shown in interpreter #891
- *Issues addressed:*
    - *#645, #672, #722, #762, #793, #800, #869, #879, #889, #890, #893, #896, #898, #899, #906*

Version 3.4 (May 5, 2018)

- *New features:*
    - Switch Python Engines without exiting PyScripter
    - Faster loading times
    - Initial support for running Jupyter notebooks inside PyScripter
    - Syntax highlighting for JSON files
    - New IDE option "Style Main Window Border"
    - Find in Files and ToDo folders can include parameters (#828)

- *Issues addressed:*
    - *#627, #852, #858, #862, #868, #872*

Version 3.3 (March 14, 2018)

- *New features:*
    - Thread debugging (#455)
    - Form Layout and placement stored in PyScripter.local.ini
- *Issues addressed:*
    - *#659, #827, #848, #849*

Version 3.2 (January 14, 2018)

- *New features:*
    - Dpi awareness (Issue 769)
- *Issues addressed:*
    - *#705, #711, #717, #748*

Version 3.1 (December 31, 2017)

- *New features:*
    - Code folding
    - Indentation lines
    - New IDE option "Compact line numbers"
    - pip tool added
    - Internal Interpreter is hidden by default
    - KabyleTranslation added
- *Issues addressed:*
    - *#16, #571, #685, #690, #718, #721, #765, #814, #836*

Version 3.0 (October 17, 2017)

- *New features:*
  - Python 3.5, 3.6 and 3.7 support
  - New Style Engine (VCL Styles) with high quality choices
  - Visual Style Preview and selection (View, Select Style)
  - Visual Source highlighter theme selection (Editor Options, Select theme)
  - German Translation added

Version 2.6 (March 20, 2015)

- *New features:*
  - Python 3.4 support added

Version 2.5 (March 19, 2012)

- *New features:*
  - This is the first joint 32-bit and 64-bit version release
  - Python 3.3 support added
  - Recent Projects menu item added
  - Expandable lists and tuples in the Variables window (#583)
  - Expandable watches as in the Variables window (#523)
  - Basic support for Cython files added (#542)
  - New interpreter action Paste & Execute (#500) Replaces Paste with Prompt
  - New PyIDE option "Display package names in editor tabs" default True (#115)
  - New search option "Auto Case Sensitive" (case insensitive when search text is lower case)
  - The Abort command raises a KeyboardInterrupt at the Remote Engine (#618)
  - Incremental search in the Project Explorer matches any part of a

    - filename (#623)
    - New IDE option "File line limit for syntax check as you type" default 1000
- *Issues addressed:*
    - #516, #348, #549, #563, #564, #568, #576, #587, #591, #592, #594, #597, #598, #599, #612, #613, #615

Version 2.4.3 (September 20, 2011)

- New features:
    - 100% portable by placing PyScripter.ini in the PyScripter exe directory
    - Ctrl+Mousewheel for zooming the interpreter (#475)
    - Show docstrings during completion list (#274)
    - New IDE Option "File Change Notification" introduced with possible values Full, NoMappedDrives(default), Disabled (#470)
    - Background color for Matching and Unbalanced braces (#472)
    - New IDE option "Case Sensitive Code Completion" (default True)
    - New IDE option "Complete Python keywords" (default True)
    - New IDE option "Complete as you type" (default True, #473)
    - New IDE option "Complete with word-break chars" (default True)
    - New IDE option "Auto-complete with one entry" (default True, #452)
- *Issues addressed:*
    - Command line history not saved
    - Editing a watch to an empty string crashes PyScripter
    - Replace in Find-in-Files now supports subexpression substitution (#332)
    - Import statement completion does not include builtin module names
    - #461, #463, #468, #471, #474, #478, #488, #496, #504, #508, #509, #511, #512, #515, #525, #526, #527, #528, #532, #559, #560

Version 2.4.1 (December 12, 2010)

- *New features*:

    - Side-by-side file editing (#214)
    - Enhanced regular expression window (findall - #161)
    - Open file at a specific line:column (#447)
- *Issues addressed:*
    - Reduced flicker when resizing form and panels
    - #415, #437, #449

Version 2.3.4 (November 25, 2010)

- *New features:*
    - Compatibility with Python 3.1.3rc, 3.2a4
    - Add watches by dragging and dropping text
    - Ctrl + Mouse scroll scrolls whole pages in print preview
    - Search for custom skins first in the Skins subdirectory of the Exe file if it exists
- *Issues addressed:*
    - #430, #434, #435, #439, #440, #441, #443, #446

Version 2.3.3 (October 16, 2010)

- *New features:*
    - Native unicode strings throughtout (speed improvements on XP)
    - Revamped Code Explorer (#192, #163, #213, #225)
    - Improvements to Code completion
        - Auto-completion for the import statement in python 2.5 and later (#230)
        - Processing of function return statements
        - Background module parsing and caching of parsed modules
    - Start-up python scripts pyscripter_init.py and python_init.py. See help file for details.
    - Imporved "Match Brace" (#426) and New Editor Command "Select to brace"
    - Italian translation by Vincenzo Demasi added

- Russian translation by Aleksander Dragunkin added
    - New IDE option "Highlight selected word" (#404)
    - New IDE option "Use Python colors in IDE"
    - New Edit command "Copy File Name" available at the contex menu of the tab bar
    - New commands "Previous Frame", "Next Frame" to change frame using the keyboard (#399)
    - JavaScript and PHP Syntax Highlighters added
- *Issues addressed:*
    - #103, #239, #267, #270, #271, #294, #317, #324, #343, #378, #395, #403, #405, #407, #411, #412, #413, #419, #421, #422, #425, #432

Version 2.1.1 (August 20, 2010)

- *New features:*
    - Support for Python 3.2
    - New IDE Option added "Jump to error on Exception" (#130)
    - New IDE Option added "File template for new python scirpts" (#385)
    - New IDE Option added "Auto completion font" (#365)
    - French translation by Groupe AmiensPython added
- *Bug fixes:*
    - #297, #307, #346, #354, #358, #371, #375, #376, #382, #384, #387, #389

Version 2.0 (July 30, 2010)

- *New features:*
    - Support for Python 2.7
    - Moved to Rpyc v3.07, now bundled with PyScripter
    - IDE Option "Reinitialize before run" was added defaulting to True
    - The default Python engine is now the remote engine
    - Spanish translation by Javier Pim s (incomplete) was added
- *Bug fixes:*

        - #236, #304, #322, #333, #334

Version 1.9.9.7 (May 20, 2009)

- *New features:*
    - Updated theme engine with customizable themes
    - Python 3.1 support
- *Bug fixes:*
    - #269, #273, #278, #291, #292

Version 1.9.9.6 (Feb 16, 2009)

- *New features:*
    - Remote interpreter and Debugger
    - Python 2.6 and 3.0 support
    - Project Explorer supporting multiple run configurations with advanced options
    - New debugger command: Pause
    - Execute selection command added (Ctrl-F7)
    - Interpreter command history improvements:
        - Delete duplicates
        - Filter history by typing the first few command characters
        - Up|Down keys at the prompt recall commands from history
    - Code Explorer shows imported names for (from ... import) syntax (12)
    - Improved sort order in code completion
    - Save modified files dialog on exit
    - Finer control on whether the UTF-8 BOM is written
        - Three file encodings supported (Ansi, UTF-8, UTF-8 without BOM)
    - IDE option to detect UTF-8 encoding (useful for non-Python files)
    - IDE options for default linebreaks and encoding for new files
    - Warning when file encoding results in information loss
    - IDE option to position the editor tabs at the top

- IDE Windows navigation shortcuts
- Pretty print intperpreter output option (on by default)
- Pyscripter is now Vista ready
- Docking window improvements
- PYTHONDLLPATH command line option so that Pyscripter can work with unregistered Python
- Watches Window: DblClick on empty space adds a watch, pressing Delete deletes (45)
- Wrapping in Search & Replace (38)
- New IDE Option "Save Environment Before Run" (50)
- New IDE command Restore Editor pair to Maximize Editor (both work by double clicking the Tabbar)
- New IDE Option "Smart Next Previous Tab" (z-Order) on by default (20)
- Word Wrap option exposed in Editor Options
- New File Reload command
- Import/Export Settings (Shortcuts, Highlighter schemes)
- New IDE option "Auto-reload changed files" on by default (25)
- New menu command to show/hide the menu bar. The shortcut is Shift-F10 (63)
- New command line option --DPIAWARE (-D) to avoid scaling in VISTA high DPI displays (77)
- New command line option --NEWINSTANCE (-N) to start a new instance of PyScripter
- You can disable a breakpoint by Ctrl+Clicking in the gutter
- Syntax Errors are indicated by icon in the tabbar (93)
- Command to jump to the first syntax error (Shift+Ctrl+E)
- New Firefox-like search/replace interface
- Incremental Search (100)
- New command "Highlight search text" (Shft+Ctrl+H)
- New command line option --DEBUG (-B) to use debug version of Python dll (108)
- New command "Word wrap" visible in the Editor toolbar (112)
- New command "Go to Debugger Position" (118)
- The size of the auto completion list is now persisted
- Split Editor View (31)
- New parameter $CmdLineArgs that returns the active command line

  arguments and can be used with external tools
    - New IDE options "Editor code completion" and "Interpreter code completion" which can be used to disable code completion
    - New IDE option "Show Tab Close Button"
    - New debugger command "Post mortem" (26)
    - New IDE option "Post mortem on exception"
    - Auto-resizing the fields of list views by double clicking on column separators
    - Advanced search and replace external tool added (uses re.sub)
    - Enhanced Execute Selection command (73)
    - Two new IDE options added (Dock Animation Interval and Dock Animation Move Width - 134)
    - Toolbar customization
    - Two new IDE options added ("Interpreter History Size" and "Save Command History") (#131)
    - Cut and copy without selection now cut and copy the current line (as in Visual Studio, #64)
    - Removed the Interpeter options "Clean up Namespace" and "Clean up sys.modules"
    - Improved HTML, XML highlighting with code completion and Web preview
    - C/C++ highlighting added
    - Two new interpreter commands added: Copy without prompts, and Paste with prompts (#183)
    - Localization using gettext (Japanese, Chinese and Greek translations added)
    - YAML highlighter added
- *Bug fixes*
    - Shell Integration - Error when opening multiple files
    - Configure External Run - ParseTraceback not saved properly
    - Order of tabs not preserved in minimised docked forms
    - sys.argv contained unicode strings instead of ansi strings
    - Bug fixes and improvements in Editor Options Keystrokes tab (#6)
    - Better error handling of File Open and File Save
    - Page Setup Header and Footer not saved (#7)
    - Hidden Tabbed windows reappearing when restarting
    - Duplicate two-key editor command not detected

- "Clean up namespace" and "Clean up sys modules" settings become effective after restarting PyScripter
- Exception when setting the Active Line Color in Editor Options dialog
- Raw_input does not accept unicode strings
- Error in docstring extraction (#11)
- Fixed some problems with the toggle comment command
- Fixed rare bug in restoring layout
- Code tips wrong if comments are present among parameters (#15)
- Notification of file changes can miss files (#17)
- Certain syntax coloring options were not saved
- ToDo List did not support encoded files and unicode
- ToDo List did not support multiline comments (#14)
- Fixed bug in IDE Shortcuts dialog
- Swapped the positions of the indent/dedent buttons (#23)
- Syntax highlighter changes to the interpreter are not persisted
- Multiple target assignments are now parsed correctly
- Gutter gradient setting not saved
- Disabling a breakpoint had no effect
- Tab order not preserved when restarting PyScripter
- Disassembly and Documentation views not working with remote engines
- More robust "Reinitialize" of remote Python engines (Issues 143, 145)
- Shift-Tab does not work well with the Trim Trailing Spaces editor option
- #28, #32, #39, #40, #41, #46, #47, #48, #49, #52, #55, #56, #57, #65, #66, #67, #70, #71, #72, #74, #75, #76, #81, #82, #83, #86, #88, #90, #91, #92, #94, #96, #98, #99, #100, #102, #105, #106, #107, #109, #113, #117, #119, #120, #120, #122, #123, #125, #132, #134, #135, #136, #137, #138, #139, #140, #141, #146, #147, #150, #153, #155, #160, #164, #165, #166, #167, #168, #169, #171, #174, #178, #182, #186, #193, #195, #196, #197, #198, #201, #202, #204, #206, #208, #212, #219, #226, #228, #229, #234, #235, #237, #253, #261

Version 1.7.2 (Oct 26, 2006)

- *New features:*
    - Store toolbar positions
    - Improved bracket completion now also works with strings (#4)
- *Bug fixes:*
    - Bracket highlighting with non default background
    - Opening wrongly encoded UTF8 files results in empty module
    - File Format (Line End) choice not respected
    - Initial empty module was not syntax highlighted
    - Save As dialog had no default extension set
    - Unit Testing broken (regression)
    - Gap in the default tool bar (#3)

Version 1.7.1 (Oct 15, 2006)

- *New features:*
    - Repeat scrolling of editor tabs
    - Massively improved start up time
    - Faster Python source file scanning
- *Bug fixes:*
    - Infinite loop with cyclical Python imports

Version 1.7 (Oct 14, 2006)

- *New features:*
    - Unicode based editor and interactive interpreter
    - Full support for Python source file encodings
    - Support for Python version 2.5 and Current User installations
    - Check syntax as you type and syntax hints (IDE option)
    - Tab indents and Shift-Tab unindents (Editor Options - Tab Indents)
    - Editor Zoom in/out with keyboard Alt+- and Ctrl+mouse wheel
    - Improved Debugger hints and completion in the interpreter
    - work with expressions e.g. sys.path1.
    - for debugger expression hints place the cursor on ')' or ']'

    - Improved activation of code/debugger hints
    - IDE options to Clean up Interpreter namespace and sys.modules after run
    - File Open can open multiple files
    - Syntax highlighting scheme selection from the menu
    - File filters for HTML, XML and CSS files can be customized
    - Option to disable gutter Gradient (Editor Options - Gutter Gradient)
    - Option to disable theming of text selection (Editor Options - theme selection)
    - Option to hide the executable line marks
    - Active Line Color Editor option added. Set to None to use default background
    - Files submenu in Tabs popup for easy open file selection
    - Add Watch at Cursor added to the Run menu and the Waches Window popup menu
    - Pop up menu added to the External Process indicator to allow easy termination of such processes
    - If the Ini file exists in PyScripter directory it is used in preference to the User Directory in order to allow USB storage installations
    - Editor options for each open file are persisted
    - Auto close brackets in the editor
    - Improved speed of painting the Interpreter window
    - Interactive Interpreter Pop up menu with separately persisted Editor Options
    - Toggle comment (Ctrl+^) in addition to comment/uncomment
    - File Explorer improvements (Favourites, Create New Folder)
    - File Templates
    - Windows Explorer file association (installation and IDE option)
    - Command line history
    - Color coding of new and changed variables in the Variables Window
- *Bug fixes:*
    - Gutter glyphs painted when gutter is invisible
    - Sticky bracket highlighting in the interpreter window
    - Selecting lines by dragging mouse in the gutter sets breakpoint
    - Speed improvements and bugfixes related to layouts
    - Error in Variable Windows when showing dictionaries with non string keys

    - File notification error for Novel network disks
    - Wrong line number in External Run traceback message
    - No horizontal scroll in output window
    - Code completion Error with packages containing module with the same name
    - Problem with sys.stdin.readline() and partial line output (stdout) statements
    - Infinite loop when root of package is the top directory of a drive

Version 1.5.1 (Mar 14, 2006)

- *New features:*
    - Unit test integration (Automatic generation of tests, and testing GUI)
    - Added highlighting of HTML, XML and CSS files
    - Command line parameters for scripts run internally or debugged
    - IDE shortcut customization
    - Conditional breakpoints
    - Persistence of breakpoints, watches, bookmarks and file positions
    - Save and restore IDE windows layouts
    - Generate stack information when untrapped exceptions occur and give users the option to mail the generated report
    - Running scripts does not polute the namespace of PyScripter
    - Names in variables window are now sorted
    - Allow only a single Instance of Pyscripter and open command line files of additional invocations at new tabs
    - Interpreter window is now searchable
    - Added option to File Explorer to browse the directory of the Active script
    - New distinctive application icon thanks to Frank Mersmann and and Tobias Hartwich
    - File Explorer autorefreshes
    - Improved bracket highlighting
    - User customization (PyScripter.ini) is now stored in the user's Application Data direcrory to support network installations(breaking change). To restore old settings copy the ini file to the new location.

- *Bug fixes:*
  - Resolved problems with dropping files from File Explorer
  - Restore open files options not taken into account
  - Resolved problems with long Environment variables in Tools Configure
  - Resolved problems with help files
  - Reduced problems with running wxPython scripts
  - Changing the Python Open dialog filter did not affect syntax highlighting
  - CodeExplorer slow when InitiallyExpanded is set
  - Help related issues
  - Other fixes

Version 1.3 (Dec 18, 2005)

- *New features:*
  - Code completion in the editor (Press Ctrl+Space while or before typing a name)
  - Parameter completion in the editor (Press Shift+Ctrl+Space)
  - Find definition and find references independent of BicycleRepairMan much faster and arguably better
  - Find definition by clicking works for imported modules and names
  - A new feature-rich Python code parser was developed for implementing the above
  - Improved the Variable Window (shows interpreter globals when not debugging and Doc strings)
  - Improved code and parameter completion in the interactive interpreter
  - Integrated regular expression tester
  - Code and debugger hints
  - Set the current directory to the path of the running script
  - Added IDE option MaskFUPExceptions for resolving problems in importing Scipy
  - Tested with FastMM4 for memory leaks etc. and fixed a couple of related bugs
  - Bug fixes and other improvements

Version 1.2 (Aug 28, 2005)

- *New features:*
  - Extended code editor:
    - Context sensitive help on Python keywords
    - Parameterized Code Templates (Ctrl-J)
    - Accept files dropped from Explorer
    - File change notification
    - Detecting loading/saving UTF-8 encoded files
    - Converting line breaks (Windows, Unix, Mac)
  - Editor Views
    - Disassembly
    - HTML Documentation (pydoc)
    - To Do List
    - Find and Replace in Files
    - Parameterized Code Templates
    - Choice of Python version to run via command line parameters
    - Run Python Script externally (highly configurable)
    - External Tools (External run and capture output)
    - Integration with Python tools such as PyLint, TabNanny, Profile etc.
    - Powerful parameter functionality for external tool integration
    - Find Procedure
    - Find Definition/Find references using BicycleRepairMan
    - Find definition by clicking and browsing history
    - Modern GUI with docked forms and configurable look&feel (themes)

Version 1.0 (Apr 13, 2005)

- Initial release

Known Issues

Importing and using numerical modules
If you have troubles in importing and using numerical modules such as NumPy and Scipy make sure that *Mask FPU exceptions* IDE option is checked. Alternatively you may execute the following code at the interactive interpreter or from your scripts:

```
import DebugIDE
DebugIDE.maskFPUexceptions()
```

(*The DebugIDE is a Python extension module internal to PyScripter)*

Running wxPython scripts
Use the remote Python engine for running and debugging such scripts. If you use the internal python engine PyScripter may become unstable.

You can always run such scripts externally using the Run, External Run menu command. It that case you should configure the External Run so that the "Hide Console" option is unchecked (see Running Scripts for details).

Debugging scripts which use Psyco
PyScripter like other Python IDE's has problems debugging scripts which import Pscyco. Pscyco works by replacing standard Python functions from the sys module (e.g. settrace) which are essential for debugging. Even if you run a Pscyco script without debugging it still messes up the internal interpreter for good and you will no longer be able to debug other scripts (you have to restart PyScripter). You can run such scripts externally or else you need to comment out Psyco related stuff.

Future

Here is a list of the features planned for the not-too-distant future:

- Python plugin architecture
- UML graphs generation
- wxPython form designer

You are welcome to provide feedback regarding the planned features.

Credits

PySCripter was developed using Delphi.
Special thanks to the many great developers who, with their amazing work, have made PyScripter possible. PyScripter makes use of the following components and projects:

- Python for Delphi (https://github.com/pyscripter/python4delphi)
- JVCL (jvcl.sf.net)
- SynEdit (synedit.sf.net)
- SynWeb highlighters (github.com/KrystianBigaj/synweb)
- VirtualTreeView (github.com/Virtual-TreeView/Virtual-TreeView)
- VirtualShellTools (github.com/pyscripter/mustangpeakvirtualshelltools)
- GExperts (www.gexperts.org)
- Syn Editor (sourceforge.net/projects/syn)
- Toolbar2000 (www.jrsoftware.org/tb2k.php)
- SpTBXLib (www.silverpointdevelopment.com/sptbxlib/)
- CommadLineReader (www.benibela.de)
- Silk icons (www.famfamfam.com)

Translations

- Translation manager: Lübbe Onken
- Chinese translation by "Love China"
- French translation by Groupe AmiensPython
- Italian translation by Vincenzo Demasi
- Japanese translation by Tokibito
- Russian translation by Aleksander Dragunkin
- Slovak translation by Marian Denes
- Spanish translation by Javier Pimás
- Kabyle translation by Muḥend Belqasem

Theme design

- Tanmaya Meher (www.github.com/tanmayameher)
- jprzywoski (www.github.com/jprzywoski)

Thanks are due to Frank Mersmann and and Tobias Hartwich for designing the program icon and logo.

License

the software. If this condition is not acceptable for confidential or legal reasons, everyone is free to derive components or to generate a diff file to our or other original sources.
The Pyscripter project contains code from other open source projects (see credits). These parts of the code are covered by the licenses of the respective projects.

Support and Updates

Previous Top Next

Please submit bug reports and enhancement requests using the Issue Tracker at the Github PyScripter project page. A discussion and support forum is available at http://groups.google.com/group/pyscripter.

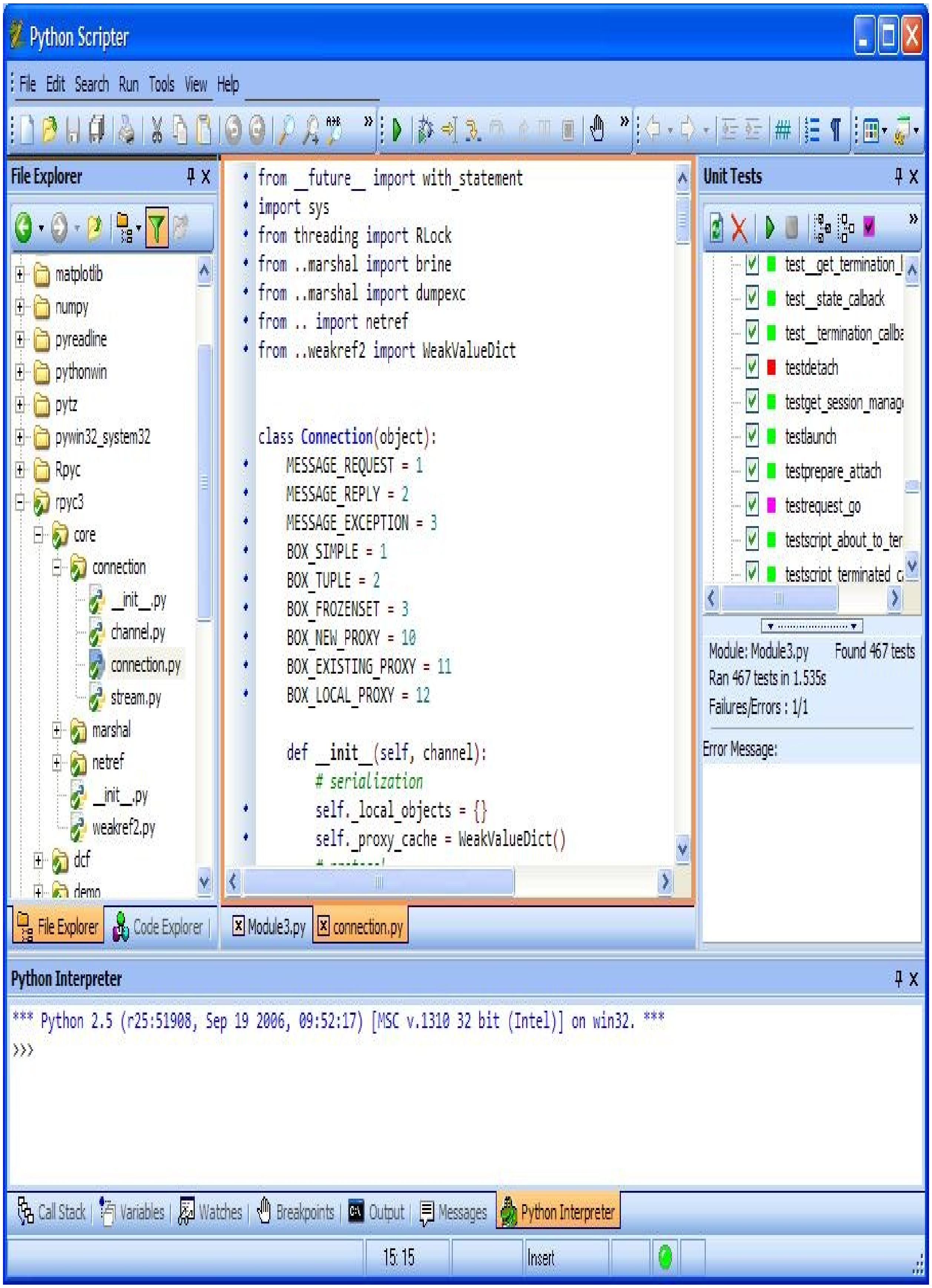
Updates are availabe through Souceforge (PyScripter downloads).
You may also get support and help by emailing pyscripter@gmail.com.

The Main IDE Window

The IDE main window shown below, consists of the main editing area organized in a tabbed form and a number of different windows which can be docked on the sides of main IDE window or can be free-floating. The toolbars at the top provide access to commonly used functions. They can also be repositioned to the sides or the bottom of the IDE window or they can be free-floating.

Tips:

- *You can rearrange the editor files by dragging and dropping the editor tabs.*
- *You can maximize the editor by double-clicking in the editor tab and then you can restore it in the same way.*
- *You can dock/undock a given window by double clicking on its window title area.*



```
from __future__ import with_statement
import sys
from threading import RLock
from ..marshal import brine
from ..marshal import dumpexc
from .. import netref
from ..weakref2 import WeakValueDict


class Connection(object):
    MESSAGE_REQUEST = 1
    MESSAGE_REPLY = 2
    MESSAGE_EXCEPTION = 3
    BOX_SIMPLE = 1
    BOX_TUPLE = 2
    BOX_FROZENSET = 3
    BOX_NEW_PROXY = 10
    BOX_EXISTING_PROXY = 11
    BOX_LOCAL_PROXY = 12

    def __init__(self, channel):
        # serialization
        self._local_objects = {}
        self._proxy_cache = WeakValueDict()
```

Module: Module3.py   Found 467 tests
Ran 467 tests in 1.535s
Failures/Errors : 1/1

Error Message:

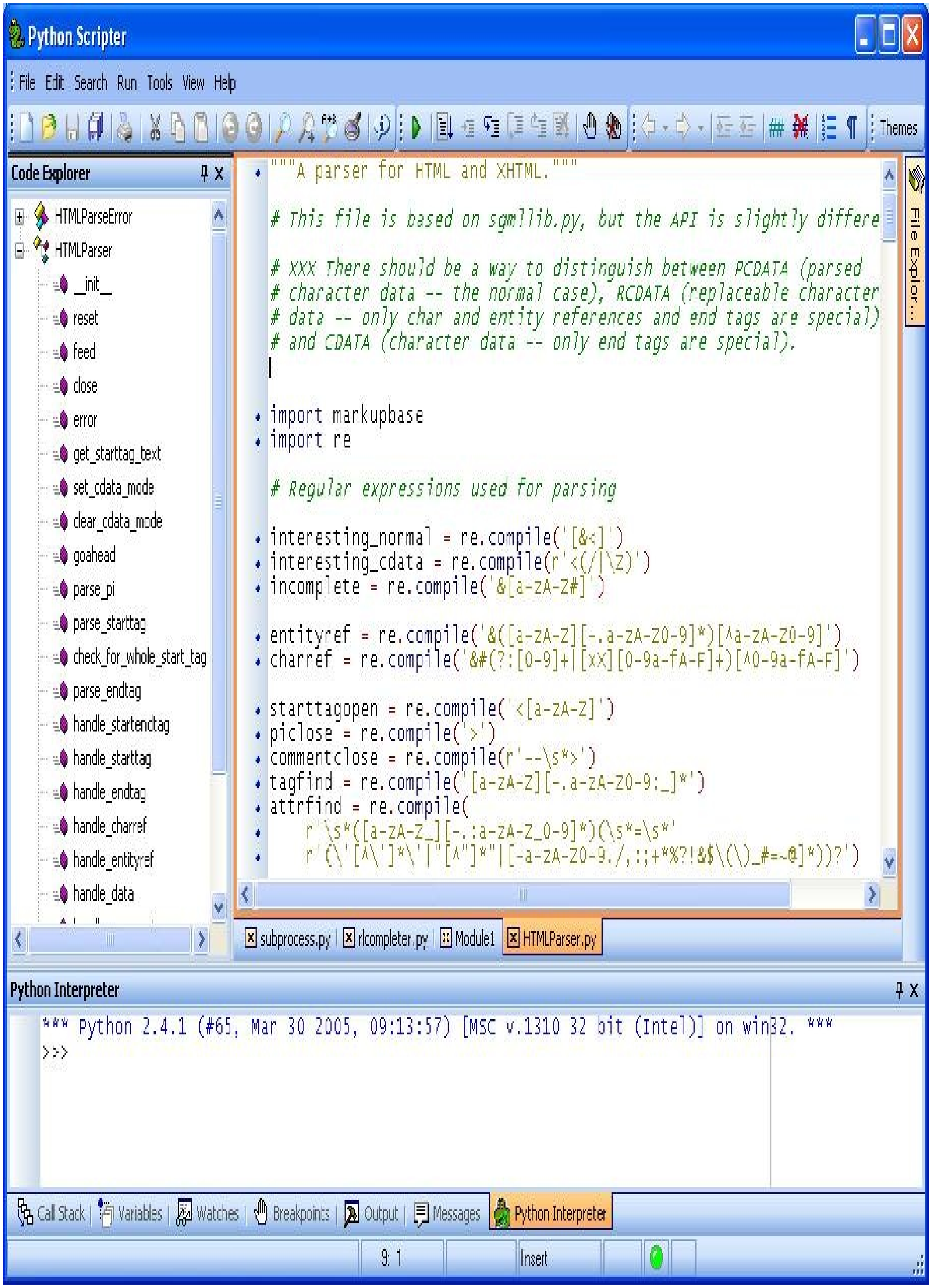
```
*** Python 2.5 (r25:51908, Sep 19 2006, 09:52:17) [MSC v.1310 32 bit (Intel)] on win32. ***
>>>
```

Window Docking

Previous Top Next

The docking features of PyScripter resemble those found Visual Studio 2003 and later.   In the picture below you see three different forms of docking. At the bottom multiple forms are docked in tabbed form. At the right-hand side a single form (Code Explorer) is docked, while on the right-hand side the File Explorer has been "unpinned" and hidden in order to occupy minimal space.

- You can undock forms by double clicking either on the window header bar or on the tab of a form.
- You can dock forms on the sides of the main window or inside one-another by dragging and dropping.
- You can unpin forms and minimize the space they occupy by clicking on the click button of the window header bar.
- You can hide forms by clicking on the close button of the window header bar.
- You can rearrange tab-docked forms by dragging and dropping the tabs



```
"""A parser for HTML and XHTML."""

# This file is based on sgmllib.py, but the API is slightly differe

# XXX There should be a way to distinguish between PCDATA (parsed
# character data -- the normal case), RCDATA (replaceable character
# data -- only char and entity references and end tags are special)
# and CDATA (character data -- only end tags are special).


import markupbase
import re

# Regular expressions used for parsing

interesting_normal = re.compile('[&<]')
interesting_cdata = re.compile(r'<(/|\Z)')
incomplete = re.compile('&[a-zA-Z#]')

entityref = re.compile('&([a-zA-Z][-.a-zA-Z0-9]*)[^a-zA-Z0-9]')
charref = re.compile('&#(?:[0-9]+|[xX][0-9a-fA-F]+)[^0-9a-fA-F]')

starttagopen = re.compile('<[a-zA-Z]')
piclose = re.compile('>')
commentclose = re.compile(r'--\s*>')
tagfind = re.compile('[a-zA-Z][-.a-zA-Z0-9:_]*')
attrfind = re.compile(
    r'\s*([a-zA-Z_][-.:a-zA-Z_0-9]*)(\s*=\s*'
    r'(\'[^\']*\'|"[^"]*"|[-a-zA-Z0-9./,:;+*%?!&$\(\)_#=~@]*))?')
```

```
*** Python 2.4.1 (#65, Mar 30 2005, 09:13:57) [MSC v.1310 32 bit (Intel)] on win32. ***
>>>
```

IDE Styles

The look-and-feel of the IDE can be customized in different ways. The color scheme in particular can be changed to one of many provided styles.
You can change the active style by using the "View, Select Style" command. Your choice is then saved and used in future activations of PyScripter.

Keyboard shortcuts

The table below provides an overview of menu commands and associated keyboard shortcuts. Further shortcuts are available in the editor (see Editor shortcuts). You can customize most IDE shortcuts using the Customize IDE shortcuts dialog.

Command	**Shortcut**	**Description**
File Management		
Close	Ctrl+F4	Close active Document
Close All Files	Shift+Ctrl+F4	Close All opened Documents
Exit	Alt+F4	Exit this program
New	Ctrl+N	Create new file
Open..	Ctrl+O	Open existing file
Save	Ctrl+S	Save active file
Save all		Save all opened files
Save As...		Save active file under a different name
Page Setup...		Setup page
Print Preview		Preview active file
Print...	Ctrl+P	Print active Document
Edit Commands		
Comment out	Ctrl+Alt+.	Comment out a block of code
Copy	Ctrl+C	Copy selection to Clipboard
Cut	Ctrl+X	Cuts selection
Dedent block	Shift+Ctrl+U	Decrease the indentation of a block of code
Delete		Delete selection
Dos/Windows		Convert linebreaks to Windows format
Indent block	Shift+Ctrl+I	Increase the indentation of a block of code
Insert parameter	Shift+Ctrl+P	Insert a parameter

Insert modifier	Shift+Ctrl+M	Insert a parameter modifier
Insert template	Ctrl+J	Insert a code template
Code completion	Ctrl+Space	Start code completion
Call tips	Shft+Ctrl+Space	Show a call tip
Mac		Convert linebreaks to Mac format
Paste	Ctrl+V	Paste Clipboard contents
Replace parameters	Shft+Ctrl+R	Replace parameters with their values
Redo	Shift+Ctrl+Z	Reverse the action of "Undo"
Select All	Ctrl+A	Select all text in the editor
Tabify		Convert spaces to tabs
Toggle comment	Ctrl + ^	(Un)comment a block of code (toggle)
Uncomment	Ctrl+Alt+,	Reverse the action of "Comment out"
Undo	Ctrl+Z	Undo last action
Unix		Convert linebreaks to Unix format
UnTabify		Convert tabs to spaces
UTF-8		Toggle the UTF-8 file format
Search Commands		
Find...	Ctrl+F	Display the search dialog
Find Definition		Find the definition of the identifier under the caret
Find Function...	Ctrl+G	Display the Find Function Dialog
Find In Files...	Shift+Ctrl+F	Display the Find-In-Files dialog
Find Next	F3	Repeat last search, searching forward
Find Next Identifier	Shift+Ctrl+Down	Find the next reference of the identifier containing the caret
Find Previous	Shift+F3	Repeat last search,

		searching backwards
Find Previous Identifier	Shift+Ctrl+Up	Find the previous reference of the identifier containing the caret
Find References		Find references of the identifier under the caret
Go To Line..	Alt-G	Go to a specific line of the code
Go To Syntax Error	Shift+Ctrl+E	Jump to the first syntax error
Matching Brace		
Replace...	Ctrl+H	Display the "Replace Text" dialog
Highlight Search Text	Shift+Ctrl+H	Highlight all occurrences of the search text
Run Commands		
Abort Debugging	Ctrl+Alt+F9	Abort debugging session
Clear All Breakpoints		Clear all breakpoints in open files
Configure External Run		Configure the "External Run" command
Debug	F9	Debug the active file
Debug Last Script	Shift+F9	Debug last script
External Run	Alt+F9	Run active file in an external Python interpreter
Run Last Script Externally	Shift+Alt+F9	Run last script in an external Python interpreter
Import Module		Import the active file in the embedded Python interpreter
Run	Ctrl+F9	Run active file without debugging
Run Last Script	Shift+Ctrl+F9	Run last scirpt
Run to Cursor	F4	Run using the debugger

		up to the cursor position
Step Into	F7	Starts or resumes debugging by stepping into the next line of code
Step Over	F8	Resumes debugging by stepping over the next line of code
Step Out	Shift+F8	Resumes debugging by stepping out the current execution frame
Syntax Check		Perform a syntax check of the active file
Toggle Breakpoint	F5	Toggles the breakpoint at the cursor position
Reinitialize Python	Ctrl+F2	Reinitialize the remote Python engine
Tools Commands		
Code Templates...		Configure Code Templates
Configure Tools...		Configure External Tools
Custom Parameters...		Configure Parameters
Documentation		Show the HTML documentation for the active file
Disassembly		Show the disassembly of the active Python file
Editor Options...		Configure Editor Options
IDE Options...		Configure IDE Options
Python Path...		Configure the Python path
View Commands		
Next Editor	Ctrl+TAB	Show the next editor
Previous Editor	Shift+Ctrl+TAB	Show the previous editor
Maximize editor	Alt+Z	Maximize editor window
Restore editor	Shift+Alt+Z	Restore maximized editor

		window
Main Menu	Shift+F10	Show/Hide main menu
Navigation Commands		
Editor	F12	Activate the editor
Interpreter	Alt+Ctrl+I	Activate the Interpreter
Code Explorer	Alt+Ctrl+C	Activate the Code Explorer window
File Explorer	Alt+Ctrl+X	Activate the File Explorer
Unit Tests	Alt+Ctrl+U	Activate the Unit Tests window
Command Output	Alt+Ctrl+O	Activate the Command Output window
Regular Expressions	Alt+Ctrl+R	Show/hide the Regular Expressions window
Todo List	Alt+Ctrl+T	Activate the Todo list window
Breakpoints	Alt+Ctrl+B	Activate the Breakpoints window
Call Stack	Alt+Ctrl+S	Activate the Call Stack window
Messages	Alt+Ctrl+M	Activate the Call Stack window
Variables	Alt+Ctrl+V	Activate the Variables window
Watches	Alt+Ctrl+W	Activate the Watches window
Help Commands		
About...		Shows the "About" dialog
Contents		Shows the help file table of contents
Editor Shortcuts		Shows editor shortcuts
External Tools		Shows help information about external tools
Parameters		Shows help information about parameters
Python Manuals		Shows Python help file

IDE Window Layouts

Previous Top Next

You can save a layout of the IDE windows under a name of your choice so that you can later restore this layout. What is saved is the visibility, docking position and size of each IDE window and the size and position of the main window. This is done via the Layouts submenu of the View menu and the corresponding View toobar button. For example you make create different layouts for editing scripts, debugging and Unit Testing.

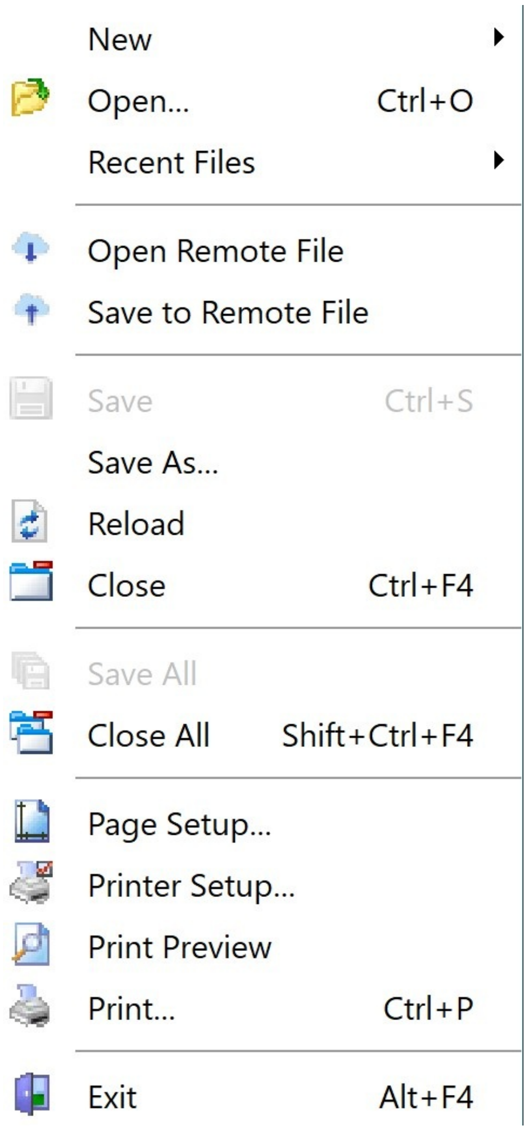
The Debug layout

If a layout named 'Debug' is available when you start debugging, then this layout is loaded. When debugging terminates the layout active before starting debugging is restored. You can save the current layout under the name 'Debug ' by using the "Set Debug Layout" command of the View|Layouts menu.

Note:

Switching layouts at the start of debugging introduces a delay which you may want to prevent. To avoid the switching of IDE Window layouts when debugging, just delete the layout named 'Debug'.

The File Menu



Commands:

New
Create a new Python script/module.   If the IDE option "File template for new python scripts" points to an existing File Template that template is used .

Open
Open an existing Python script/module. Multiple files can be opened from the File Open dialog.

Recent Files
Submenu with recently used files

Open Remote File
Open a remote file using the remote file dialog.

Save to Remote File
Save a local file to a remote computer. Specify the file name and the SSH server in the remote file dialog.

Save
Save the file in the active editor

Save As..
Save the file in the active editor under a different name

Reload
Reload the file in the active editor

Close
Close active editor

Save All
Save all editor files

Close All
Close all editors

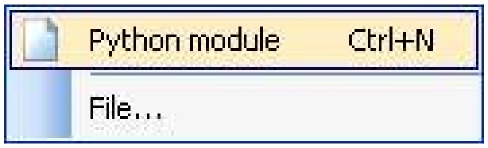
Page Setup
Setup page for printing

Printer Setup
Setup the printer

Print Preview
Print preview the file in the active editor

Print
Print the file in the active editor using the current settings

The New submenu

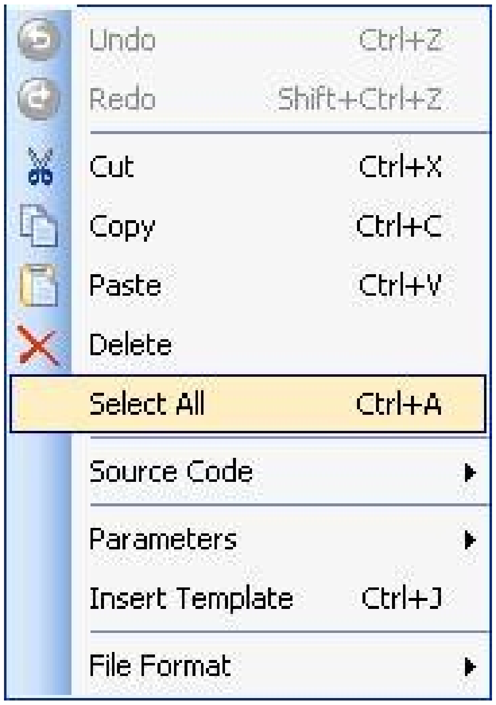


Commands:

Python module
Create a new Python script/module

File...
Shows the New File dialog

The Edit Menu



Commands:

Undo
Undo the last change in the Editor

Redo
Reverse the action of the last Undo command

Cut
Cut the selected text. If no text is selected cut the current line to the clipboard.

Copy
Copy the selected text to the Clipboard.   If no text is selected copy the current line to the clipboard.

Paste
Paste the selected text from the Clipboard

Delete

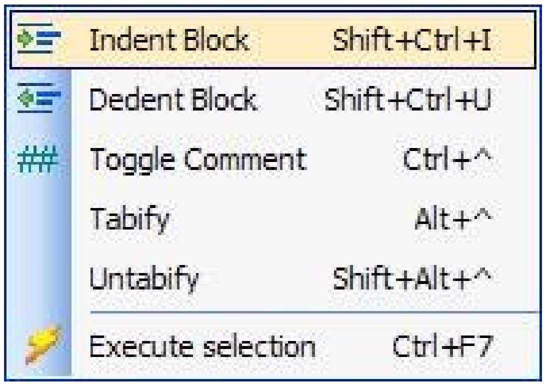
Delete the selected text

Select All
Select all text in the active editor

Read Only
Enable or disable editing in the active editor. Files opened by PyScripter from the Python directory during debugging are read only by default to prevent accidental changes.

Insert Template
Insert a code template in the active editor

The Source Code submenu


Indent Block Shift+Ctrl+I
Dedent Block Shift+Ctrl+U
Toggle Comment Ctrl+^
Tabify Alt+^
Untabify Shift+Alt+^
Execute selection Ctrl+F7

Commands:

Indent Block
Indent the selected block of code

Dedent Block
Dedent the selected block of code

Comment out
Comment out the selected block of code by inserting "##" at the beginning of

each line

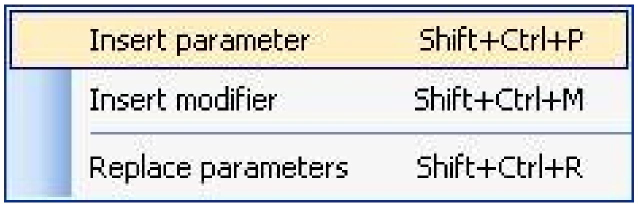
Uncomment
Delete "##" at the beginning of each line of the selected block of code

Tabify
Replace spaces with tabs in the selected block of code

Tabify
Replace tabs with spaces in the selected block of code

Execute Selection
Execute the current editor selection in the interpreter (multi-line selection). If only part of a line is selected, the selection is evaluated and the result is printed in the interpreter window. Finally, if there is no selection, the word at cursor is evaluated.

The Parameters submenu


Insert parameter Shift+Ctrl+P
Insert modifier Shift+Ctrl+M
Replace parameters Shift+Ctrl+R

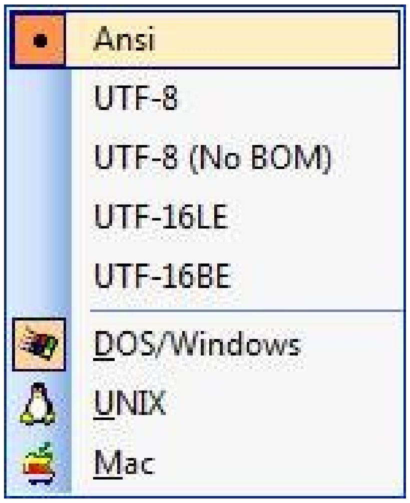
Commands:

Insert parameter
Select a parameter from a pop-up list and insert it into the active editor

Insert modifier
Select a parameter modifier from a pop-up list and insert it into a parameter in the active editor

Replace parameters
Replace all parameters with their values. If a block of text is selected the command is effected in that block, otherwise it is applied to all text in the active editor.

The File Format submenu



Commands:

Ansi
If checked the active file will be saved in Ansi encoding. Python files may provide an encoding
comment (see Python Source File Encodings)

UTF-8
If checked the active file will be saved in the UTF-8 encoding including the BOM mark.

UTF-8 (No BOM)
If checked the active file will be saved in the UTF-8 encoding without the BOM mark.

UTF-16LE
If checked the active file will be saved in the UTF-16 LE (little-endian) format.

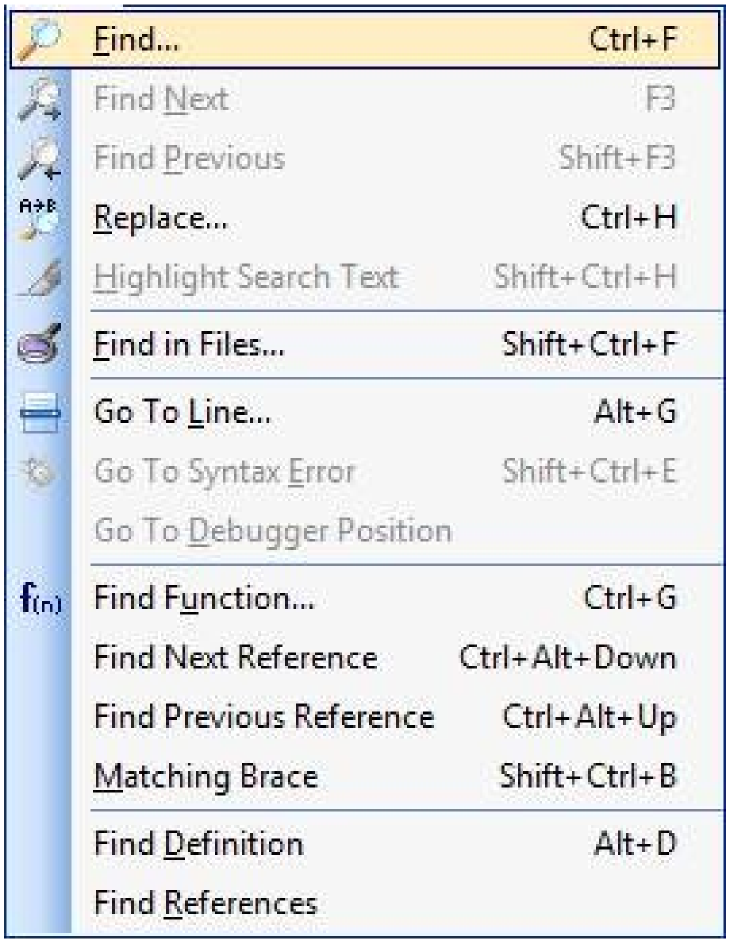
UTF-16BE
If checked the active file will be saved in the UTF-16 BE (big-endian) format.

DOS/Windows
If checked the active file will be saved using DOS/Windows line breaks (CRLF).

UNIX
If checked the active file will be saved using Unix line breaks (LF).

Mac
If checked the active file will be saved using Mac line breaks (CR).

The Search Menu


Find... Ctrl+F
Find Next F3
Find Previous Shift+F3
Replace... Ctrl+H
Highlight Search Text Shift+Ctrl+H
Find in Files... Shift+Ctrl+F
Go To Line... Alt+G
Go To Syntax Error Shift+Ctrl+E
Go To Debugger Position
Find Function... Ctrl+G
Find Next Reference Ctrl+Alt+Down
Find Previous Reference Ctrl+Alt+Up
Matching Brace Shift+Ctrl+B
Find Definition Alt+D
Find References

Commands:

Find..
Display the "Find" toolbar

Find Next
Search forward for the next match of a previously defined search

Find previous
Search backwards for the next match of a previously defined search

Replace..
Display the "Find Toolbar" showing the replace text field

Highlight Search Text
Highlights all occurrences of the search text

Find in Files..
Display the "Find in Files" dialog

Go To Line..
Displays an input box for entry of a line number and the repositions the cursor to that line number

Go To Syntax Error
Jump to the first syntax error (if any) in the active script

Go To Debugger Position
Jump to the current execution line of the debugger if the debugger is active

Find Function..
Displays the "Find Function" dialog.

Find Next Reference
Moves the cursor to the next occurance of the identifier containing the caret

Find Previous Reference
Moves the cursor to the previous occurance of the identifier containing the caret

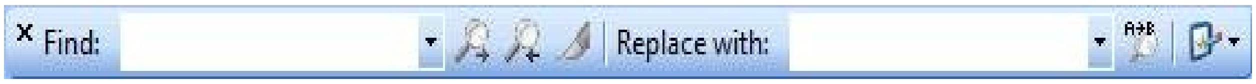
Matching Brace
If the cursor is at a brace ('(', ')', '[',]', '{' , '}') moves the cursor to the matching brace.

Find Definition
Finds the definition of the identifier containing the caret

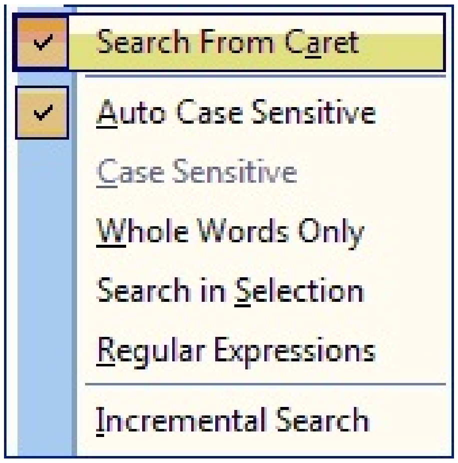
Find References
Finds references of the identifier containing the caret

The Find Toolbar

PyScripter provides a Firefox like Find Toobar for search or replace functionality:



The last button in Toolbar allows you to select different search options from the menu below:



Here is a brief explanation of the options:

Search From Caret
If checked the search begins from the cursor position, otherwise from the top of the file.

Auto Case Sensitive
Case sensitive search when .the search text contains upper case characters.

Case Sensitive
Specifies whether the search is case sensitive.

Whole Words Only
If checked the search is restricted to whole words only .

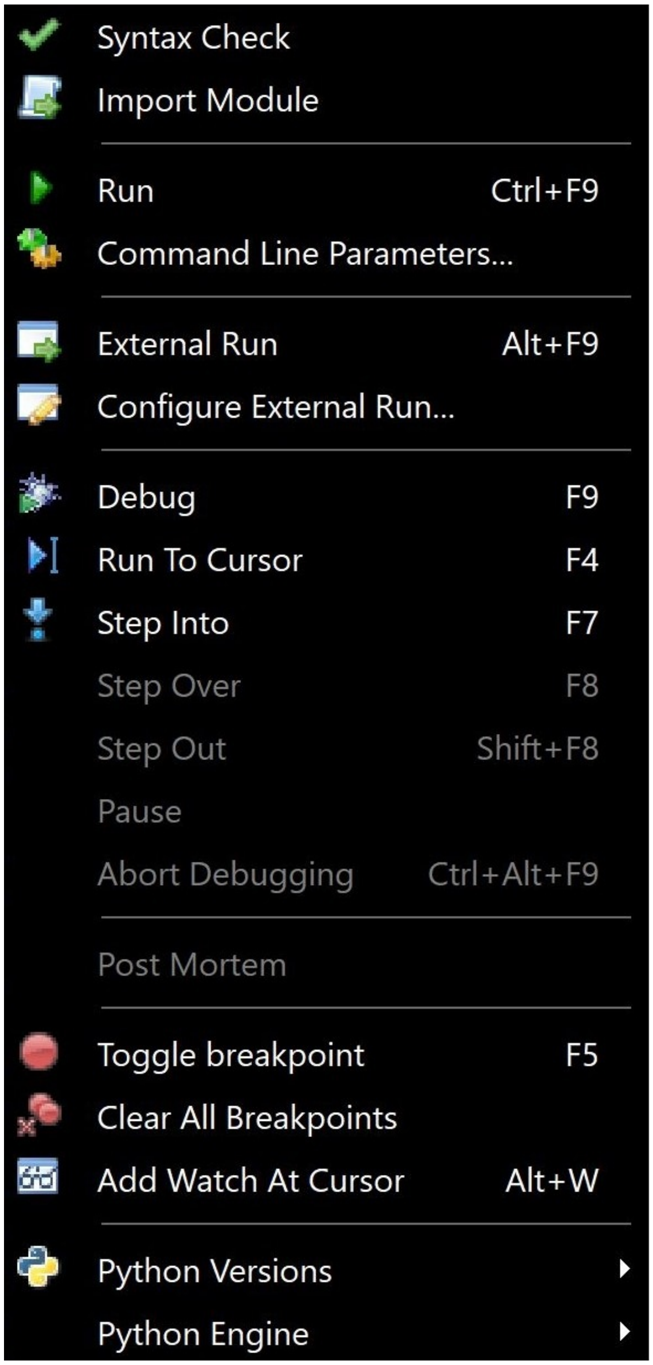
Search in Selection

If checked the search is restricted to the current selection, otherwise the whole file is searched.

Regular Expressions
If checked the search text is interpreted as a regular expression and the replacement text can contain sub-expressions (e.g. $1).

Incremental Search
If checked, Find Next is executed every time you modify the search text.

The Run Menu



Commands:

Syntax Check
Checks the syntax of the active Python script

Import Module
Imports the active Python script into the Interactive Python Interpreter

Run
Runs the active Python script without debugging using the embedded Python interpreter

Command Line Parameters...
Provide command line parameters for a script running in the embedded Python interpreter or being debugged

External Run
Runs the active Python script in an external Python interpreter

Configure External Run...
Configures the External Run command. Options are provided for selecting the Python interpreter, specifying command line parameters, capturing the standard output and reporting Traceback information. The options are the same as for the specification of External Tools.

Debug
Runs the active Python script with debugging using the embedded Python interpreter

Run To Cursor
Inserts a temporary breakpoint at the cursor position and runs the active Python script with debugging using the embedded Python interpreter

Step into
Starts or resumes debugging by stepping into the next line of code

Step over
Resumes debugging by stepping over the next line of code

Step out
Resumes debugging by stepping out the current execution frame

Pause
Stops the running program at the first available opportunity. Please note that pausing and aborting is only possibly if there are breakpoints in the running script.

Abort Debugging
Aborts debugging.

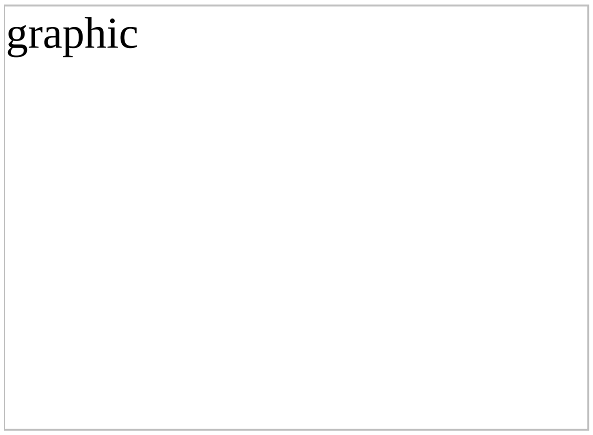
Post mortem
Enter post mortem analysis mode after an unhandled exception has occurred. In this mode you can use the Call Stack, Variables and Watches windows as well as evaluate expressions in the interpreter, to examine the causes of the exception. To exit this mode use the Abort Debugging command.

Toggle Breakpoint
Toggles the breakpoint at the cursor position

Clear all Breakpoints
Clear all breakpoints in all open files

Add Watch at Cursor
Add the expression at the current editor position as a watch expression

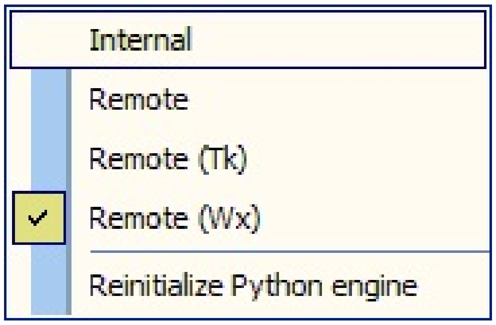
Python Engine submenu



From this submenu you can switch python versions by selecting from the shown list. It is also available from the toolbar of the main application window and the context menu of the Interactive Interpreter.

Setup Python...
Manage the python versions that PyScripter knows about. See the the Setup Python versions topic for details.

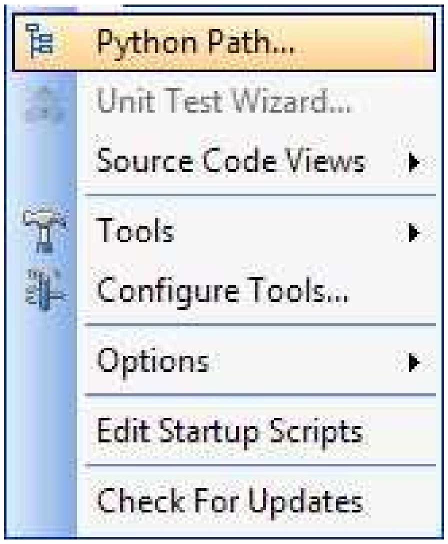
Python Engine submenu



From this submenu you can select the active Python engine. See the the Remote Python Engines topic for details.

Reinitialize Python engine
This option is only available with the remote Python engines. It restarts the active remote engine and it works even when a script is running.

The Tools Menu



Commands:

Python Path
Configures the Python Path

Unit Test Wizard...
Shows the Unit Test Wizard dialog.

Configure Tools
Pops-up a dialog which allows the creation deletion or modification of External Tools

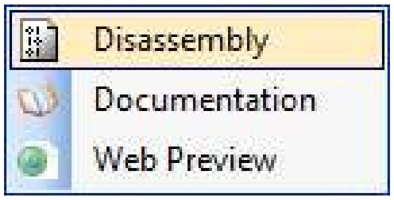
Edit Startup Scripts
Loads to the editor the Startup Python Scripts.

Check for Updates...
Checks whether an updated version of PyScripter is available.

The Source Code Views

PyScripter provides different views of Python modules that can be seen alongside the source code. Currently two such views are provided. The

Documentation view and the Disassembly view. When views other than the source code are available you can switch between the source code and these views using the tabs at the top of the editing area. You can close additional views by right clicking on their tab and selecting "Close". When the only view is the source code the tabs at the top of the editing area are hidden.



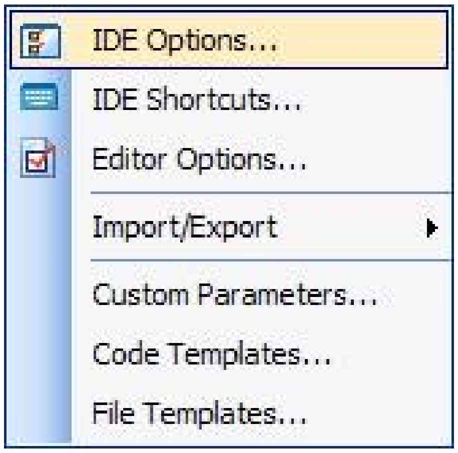
Commands:

Documentation
Shows the Documentation view

Disassembly
Shows the Disassembly view

Web Preview
Shows a Web preview of html files in a built-in browser.

The Options submenu



Commands:

IDE Options...
Shows the IDE Options dialog

IDE Shortcuts...
Shows the IDE Shortcuts dialog

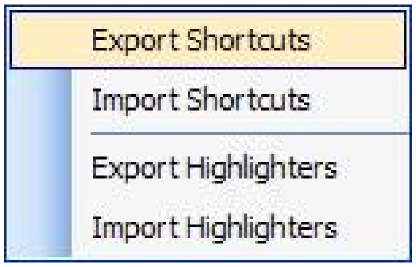
Editor Options...
Shows the Editor Options dialog

Custom Parameters...
Pops-up a dialog allowing you to define, delete or modify custom parameters

Code Templates...
Pops-up a dialog allowing you to define, delete or modify code templates

File Templates
Shows the File Templates dialog which allows you to customize the available File Templates.

The Import/Export submenu



Export Shortcuts
Exports the IDE and editor shortcuts to an "ini" file.

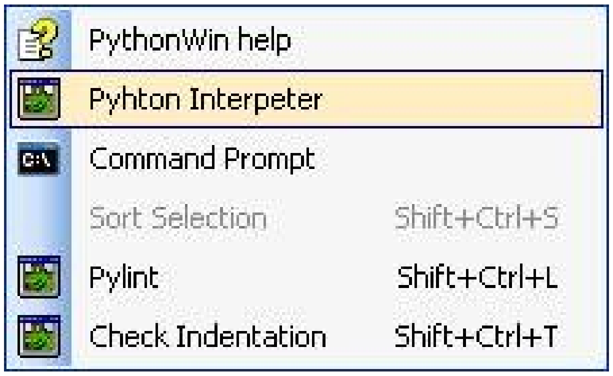
Import Shortcuts
Imports IDE and editor shortcuts from an "ini" file.

Export Highlighters
Exports syntax highlighter information to an "ini" file.

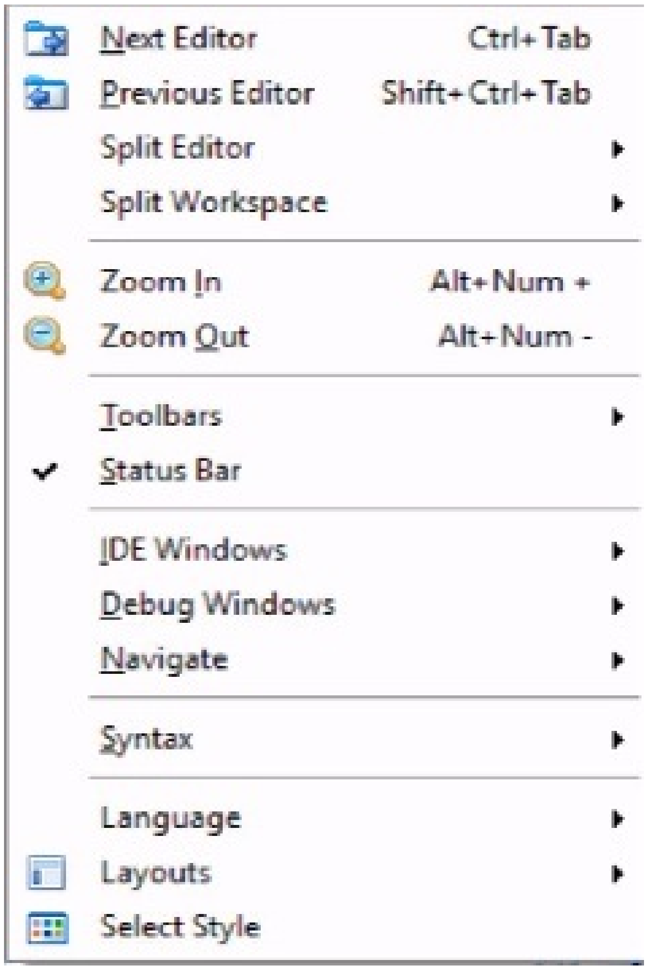
Import Highlighters
Imports syntax highlighter information from an "ini" file.

The Tools submenu

PyScripter offers the ability to define External Tools that can be run independently or interact with the IDE editor. The Tools submenu shows you the currently defined external tools. The picture below shows you some of the pre-defined tools.



The View Menu



Commands:

Next/Previous Editor
Shows the next/previous editor

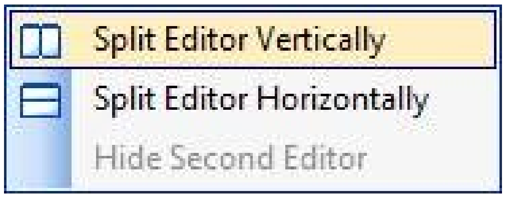
Status Bar
Shows/hides the Status bar

Zoom In
Increase the editor font size by 1.

Zoom Out
Decrease the editor font size by 1.

Select Style...
Allows you to change the visual appearance (style) of the application.

The Split Editor submenu



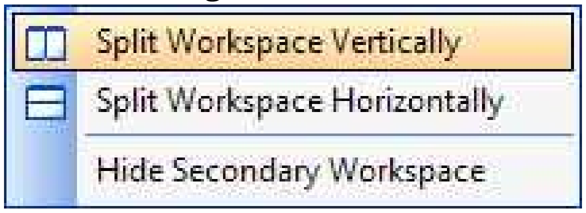
Split Editor Vertically
Creates two editor views arranged side by sided. This allows the editing of two different sections of the **same file**.

Split Editor Horizontally
Similar to the previous option, but the editor views are arranged one above the other.

Hide Second Editor
Reverses the impact of the previous two commands hiding the second editor.

The Split Workspace submenu

The following commands allow side-by-side file editing.

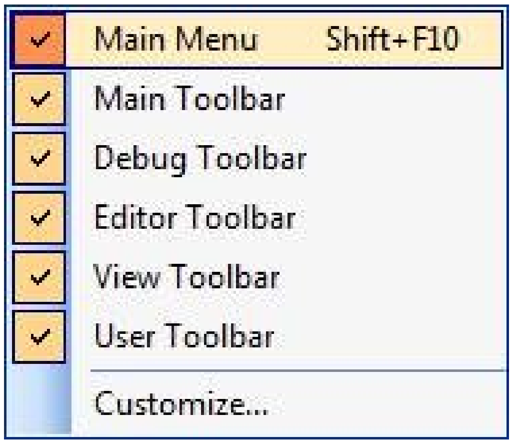


Split Workspace Vertically
View secondary workspace vertically aligned to the primary one.

Split Workspace Horizontally
View secondary Workspace horizontally aligned to the primary one.

Hide Secondary Tabs
Hide the secondary workspace and move all contained tabs to the primary one.

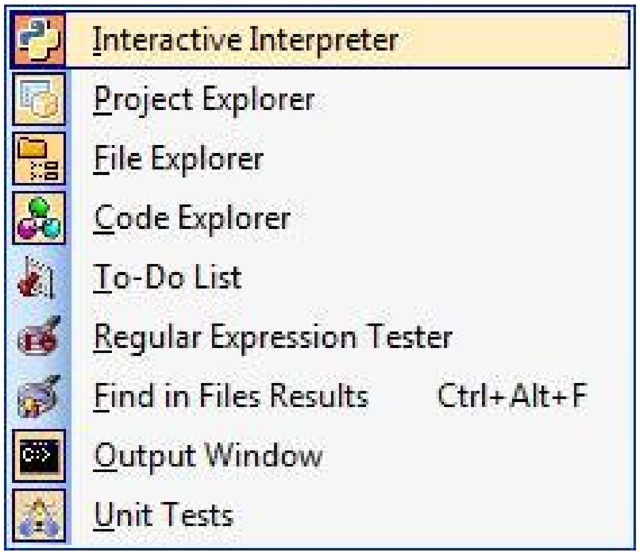
The Toolbars submenu



This submenu allow you to show/hide the different PyScripter toolbars. The Customize command displays the Toolbar customization dialog box.

The IDE Windows submenu

This submenu allow you to show/hide the different PyScripter IDE windows.



Interactive Interpreter
Shows/hides the Interactive Interpreter window

Project Explorer

Shows/hides the Project Explorer window

File Explorer
Shows/hides the File Explorer window

Code Explorer
Shows/hides the Code Explorer window

To-Do List
Shows/hides the To-Do List window

Regular Expression Tester
Shows/hides the Regular Expression Tester window

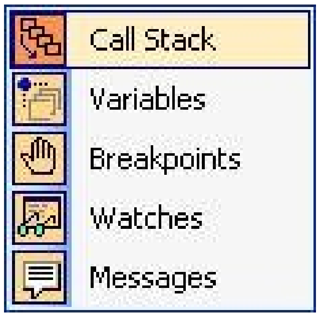
Find-in-Files Results
Shows/hides the Find-in-Files Results window

Output Window
Shows/hides the Output window

Unit Tests
Shows/hides the Unit Tests window

The Debug Windows submenu

This submenu allows you to show/hide the different PyScripter debugger windows.



Call Stack

Shows/hides the Call Stack window

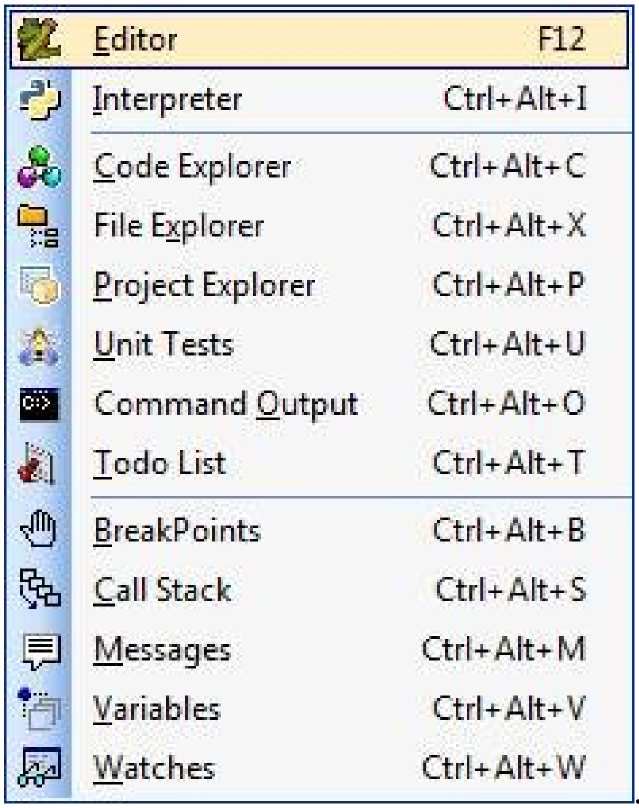
Variables
Shows/hides the Variables window

Breakpoints
Shows/hides the Breakpoints window

Watches
Shows/hides the Watches window

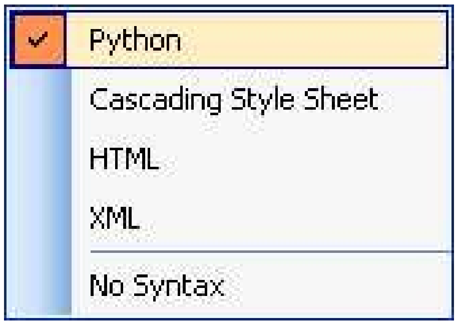
Messages
Shows/hides the Messageswindow

The Navigate submenu
This submenu allows you to move between the different PyScripter IDE and debugger windows and the editor.

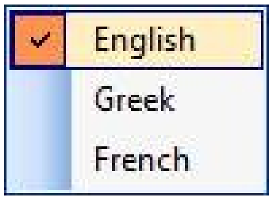


.

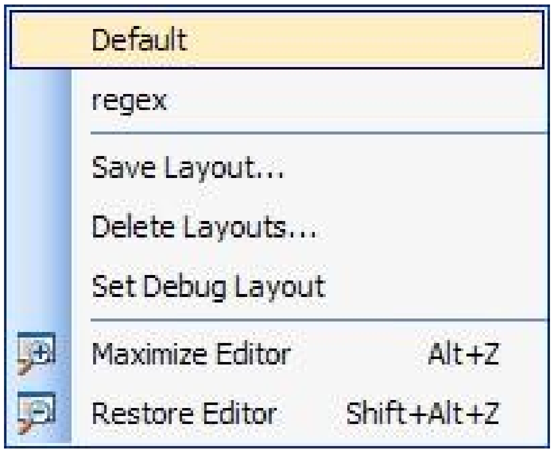
The Syntax submenu
The Syntax submenu allows you to select the syntax highlighting scheme for the active editor.



The Languages submenu
Use the Languages submenu to change the language of the User Interface of PyScripter. See the Localization topic for information about creating new translations.



The Layouts submenu
The entries above the menu separator correspond to specific layouts that have been saved and which the user can restore.



Commands:

Save Layout...
Saves the current layout under a name the user provides

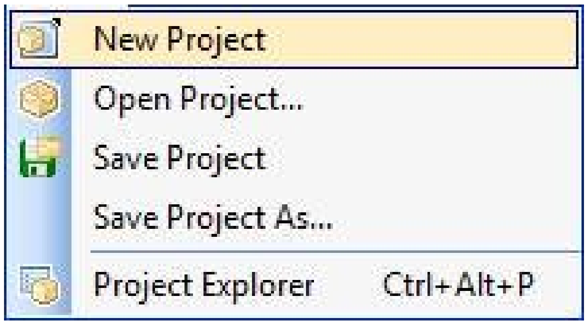
Delete Layouts...
The user is prompted for the list of layouts to delete.

Set Debug layout
Saves the current layout under the name 'Debug'. If a layout named 'Debug' is available when you start debugging, then this layout is loaded. When debugging terminates the layout active before starting debugging is restored.

Maximize editor
Maximize the editor window by auto-hiding all other IDE windows.

Restore editor
Restore the maximized editor window to each state before maximizing.

The Project Menu



Commands:

New Project
Clear the active project and start a new one.

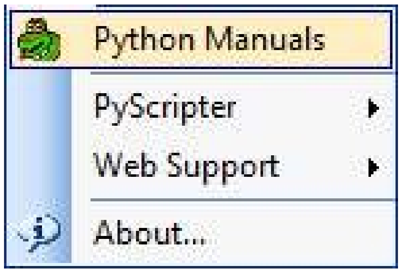
Open Project
Open a saved project and replace the active one.

Save Project
Save the active project.

Save Project As...
Save the active project under a different name.

Project Explorer
Show the Project Explorer IDE window.

The Help Menu

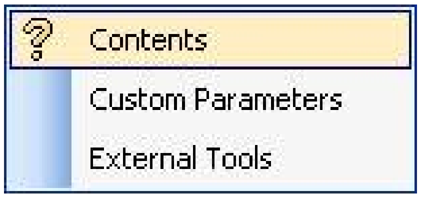


Commands:

Python Manuals
Shows the Python help files

About..
Shows the PyScripter About dialog

The PyScripter submenu



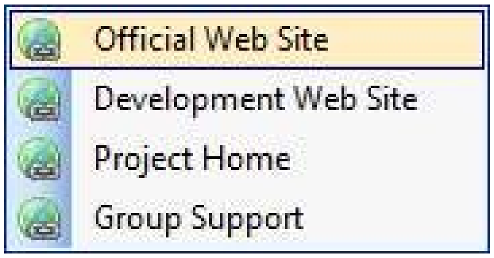
Commands:

Contents
Shows the Contents of the PyScripter Help file

Custom Parameters
Displays the help file topic on custom parameters

External tools
Displays the help file topic on creating external tools

The Web Support menu



Commands:

Official Web Site
Visit the PyScripter official web site at mmm-experts.com

Development Web Site
Visit the PyScripter development web site at pyscripter.googlepages.com

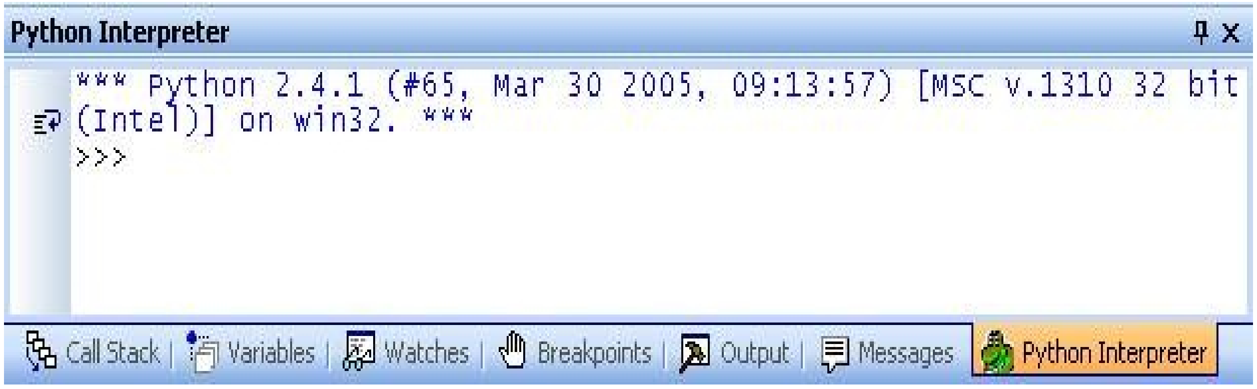
Project Home
Visit the PyScripter project home page at pyscripter.googlecode.com

Group support
Visit the PyScripter internet group at groups.google.com/group/PyScripter

The Python Interactive Interpreter

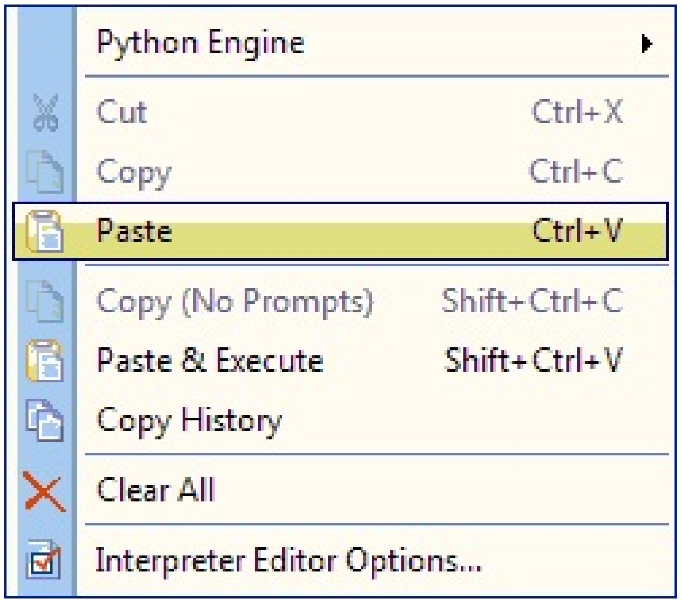
Previous Top Next

PyScripter provides an integrated interactive Python interpreter featuring command history, code completion and call tips. This window also serves as the standard output of scripts running within the IDE. During debugging and when the execution has stopped at a breakpoint, the prompt changes to "[Dbg]>>> ".



```
*** Python 2.4.1 (#65, Mar 30 2005, 09:13:57) [MSC v.1310 32 bit
(Intel)] on win32. ***
>>>
```

Context Menu


Python Engine
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Copy (No Prompts) Shift+Ctrl+C
Paste & Execute Shift+Ctrl+V
Copy History
Clear All
Interpreter Editor Options...

Copy (No Prompts)
Copies the selected text to the clipboard without the interpreter prompts.

Paste & Execute
Pastes text from the clipboard adding each contained statement to the prompt and interpreter history and executing it.

Copy History
Copies the entered command history to the clipboard.

Clear All
Clear all interpreter output.

Interpreter Editor Options...
Shows the Editor Options Dialog (see the Editor Options topic for details) for the Interpreter Window

The Python engine submenu is the same to the one available in the Run menu.

Command History

- Alt-Up : previous command
- Alt-Down : next command
- Esc : clear command

If you scroll up and click on a previously issued command, possibly modified, then this command is copied to the current prompt ready to be reissued. Copy and paste operations work as in the text editor, but pieces of code need to be entered line-by-line.

New in Version 1.7.2.4
Command Filtering: If you type some characters in the Python prompt and then invoke the history commands the history is filtered and only entries matching what you typed are shown.

New in Version 1.7.2.4
Up/Down Keys:
Up/Down keys can be used for the history previous/next commands, when the cursor is at the last line of the interpreter and this line contains the Python prompt. In that case thought the Up/Down keys are unavailable for scrolling, so you have to use the mouse to move to say the previous line, beyond which the

Up/Down keys work as normal.

Code Completion and Call Tips

Code completion and call tips are available when you type code in the interactive Python interpreter window. Click here for details.

Traceback Information

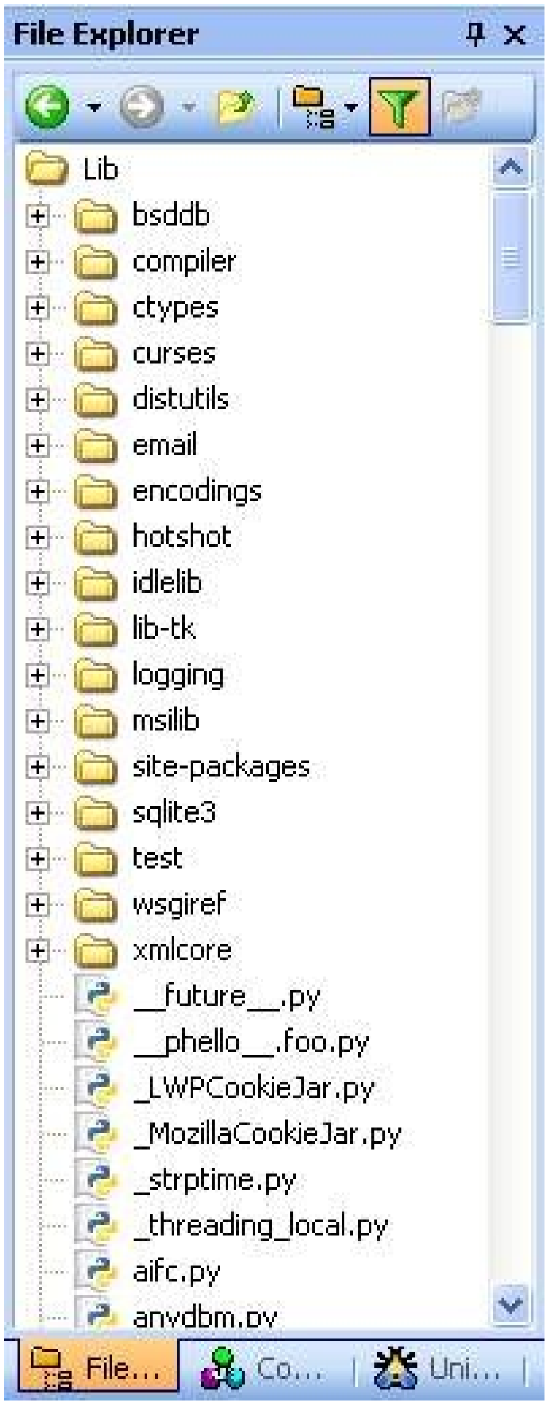
Traceback information is displayed in red. By double clicking on a line with traceback call stack information the corresponding file position is displayed in the editor if available.

The File Explorer Window

Previous Top Next

This is a powerful file explorer similar to that supplied by the Windows operating system. You can navigate through the local file system and open Python scripts by clicking on them. You can filter the displayed files and navigate directly to commonly used directories and directories on the Python path.
If you have installed Tortoise Git or Tortoise SVN you have access to version control functionality directly from PyScripter.

You can add the folders you commonly use to the **Favourites** list and you can easily set the root directory of the File Explorer to one of these folders.



The Toolbar

Commands:

Browse Back/Forward
Navigated through the history of browsed directories. Note that the browsing

history is cleared when you change the root directory of the File Explorer.

Go Up
Change the root of the File Explorer to the parent of the currently selected directory

Browse Path
Change the root of the File Explorer to commonly used directories, directories in the Favourites list and directories on the Python path

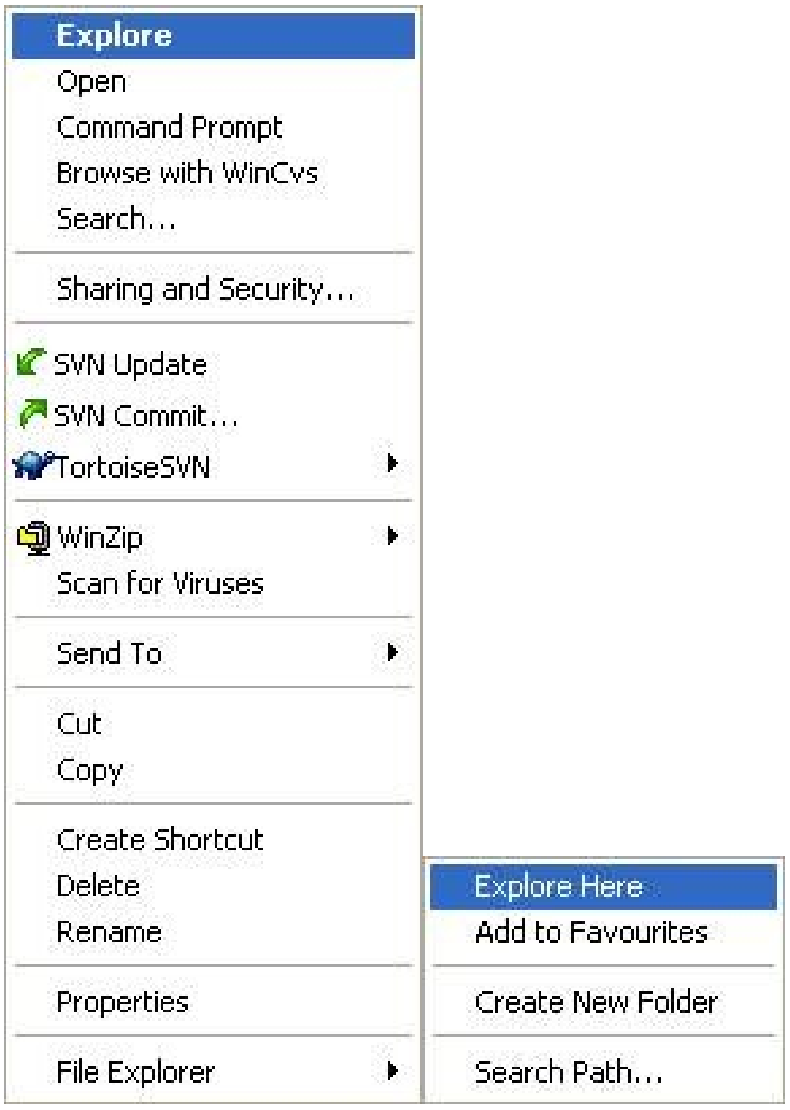
Filter files
If selected the File Explorer hides all files except Python scripts and modules. By default these are the files with ".py" extension. You can modify this default filter thought the IDE options customization.

Create New Folder
Create a new sub-folder of the currently selected folder

The Context Menu (on files and directories)

This is the standard explorer menu with the addition of the last item "File Explorer". This submenu contains the following options:

- "*Explore here*", which allows to change the root directory of the File Explorer to the selected folder.
- "*Add to Favourites"*, which adds the currently selected folder to the *Favourites* list
- *"Create New Folder"* which *c*reates a new sub-folder of the currently selected folder
- "*Search Path*" which invokes the Find-in-Files tool on the selected folder.



The Context menu (on empty space)

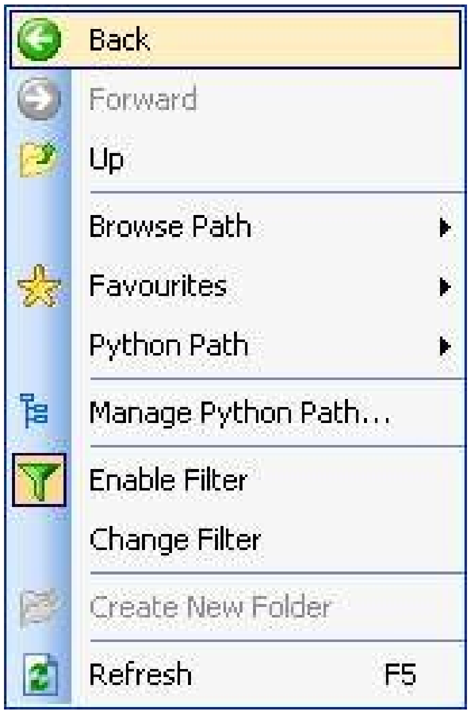
This context menu offers options similar to those found in the toolbar plus the following commands:

Manage Python Path...
Shows a dialog box from which you can modify (add and remove folders) from the python path (sys.path).

Change Filter...
Allows you to change the filter which applies to the files shown. Use a

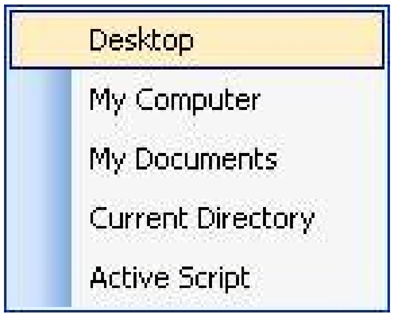
semicolon separated list, i.e. "*.py;*.pyw".

Refresh
Refresh the contents of the File Explorer. Normally not needed since it updates automatically.



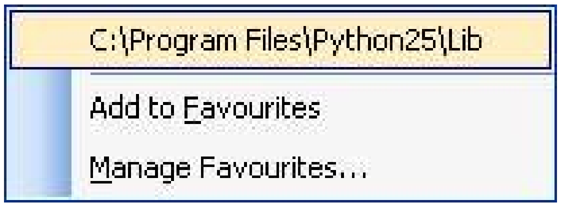
The Browse path submenu

By selecting menu options from this submenu, you can set the root directory of the File Explorer to the corresponding file directory. "Active Script" sets the root directory to the directory of the script currently edited.



The Favourites submenu

Shows the list of favourites from which you can select the one you want to set as the root folder of the File Explorer.



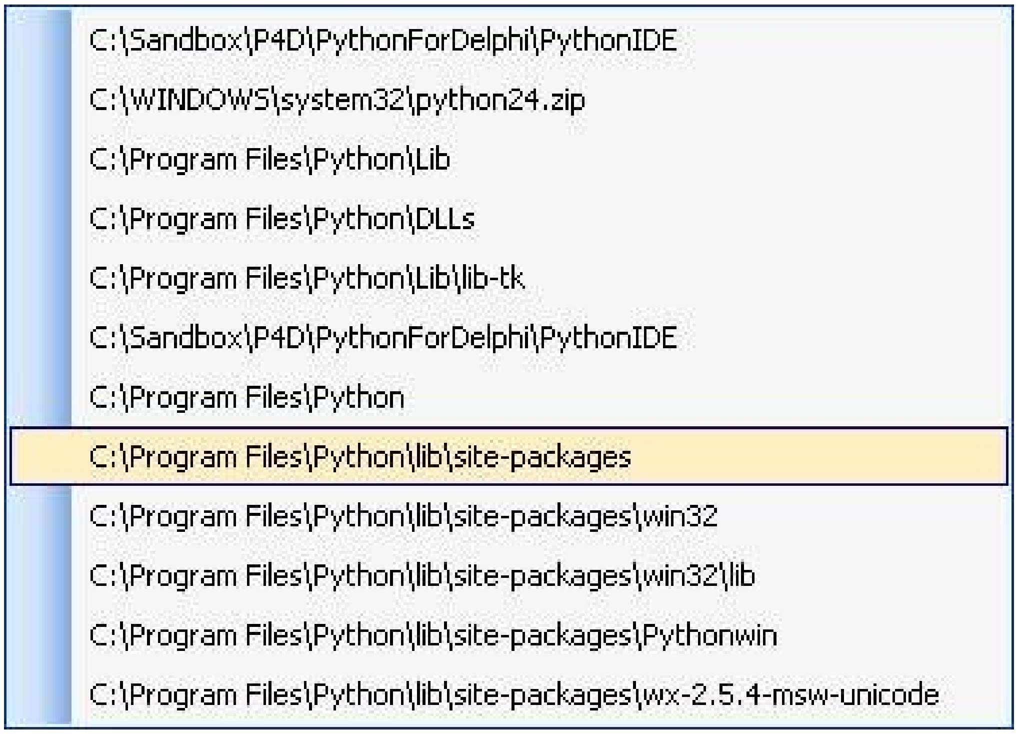
It also provides the following commands:

Add to Favourites
Adds the currently selected folder to the *Favourites* list.

Manage Favourites...
Shows a dialog box (see below) from which can manage (add and remove folders) the Favourites list.

The Python Path submenu

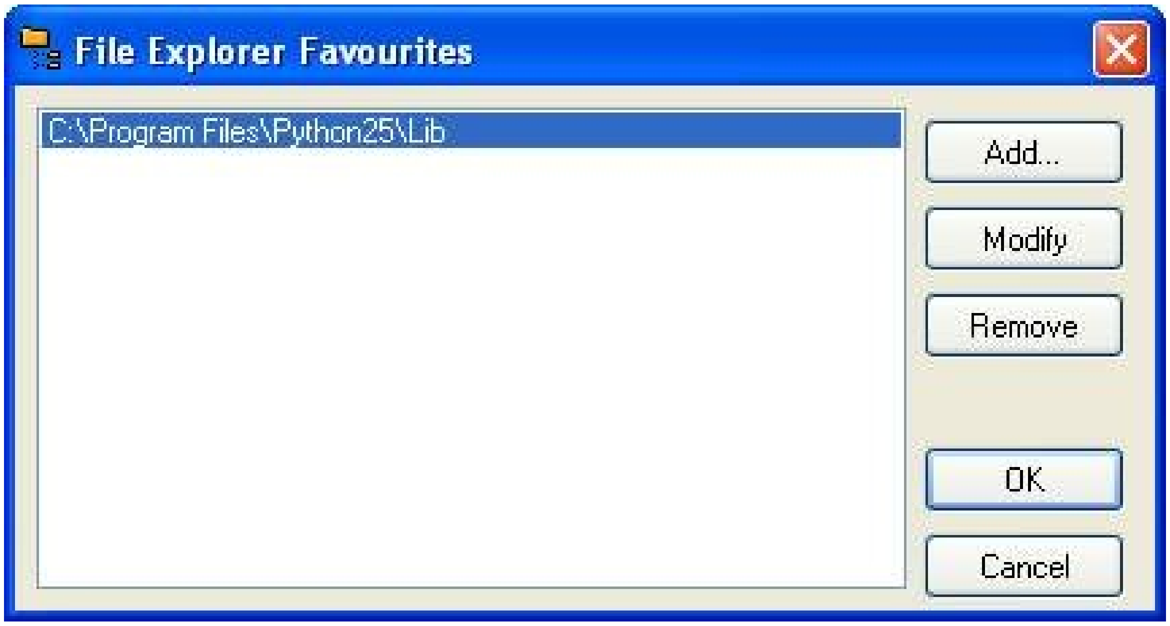
This submenu shows you the directories in the Python path of the embedded Python interpreter. Selecting a directory changes the the root directory of the File Explorer to that directory



```
C:\Sandbox\P4D\PythonForDelphi\PythonIDE
C:\WINDOWS\system32\python24.zip
C:\Program Files\Python\Lib
C:\Program Files\Python\DLLs
C:\Program Files\Python\Lib\lib-tk
C:\Sandbox\P4D\PythonForDelphi\PythonIDE
C:\Program Files\Python
C:\Program Files\Python\lib\site-packages
C:\Program Files\Python\lib\site-packages\win32
C:\Program Files\Python\lib\site-packages\win32\lib
C:\Program Files\Python\lib\site-packages\Pythonwin
C:\Program Files\Python\lib\site-packages\wx-2.5.4-msw-unicode
```

The Manage Favourites dialog box:

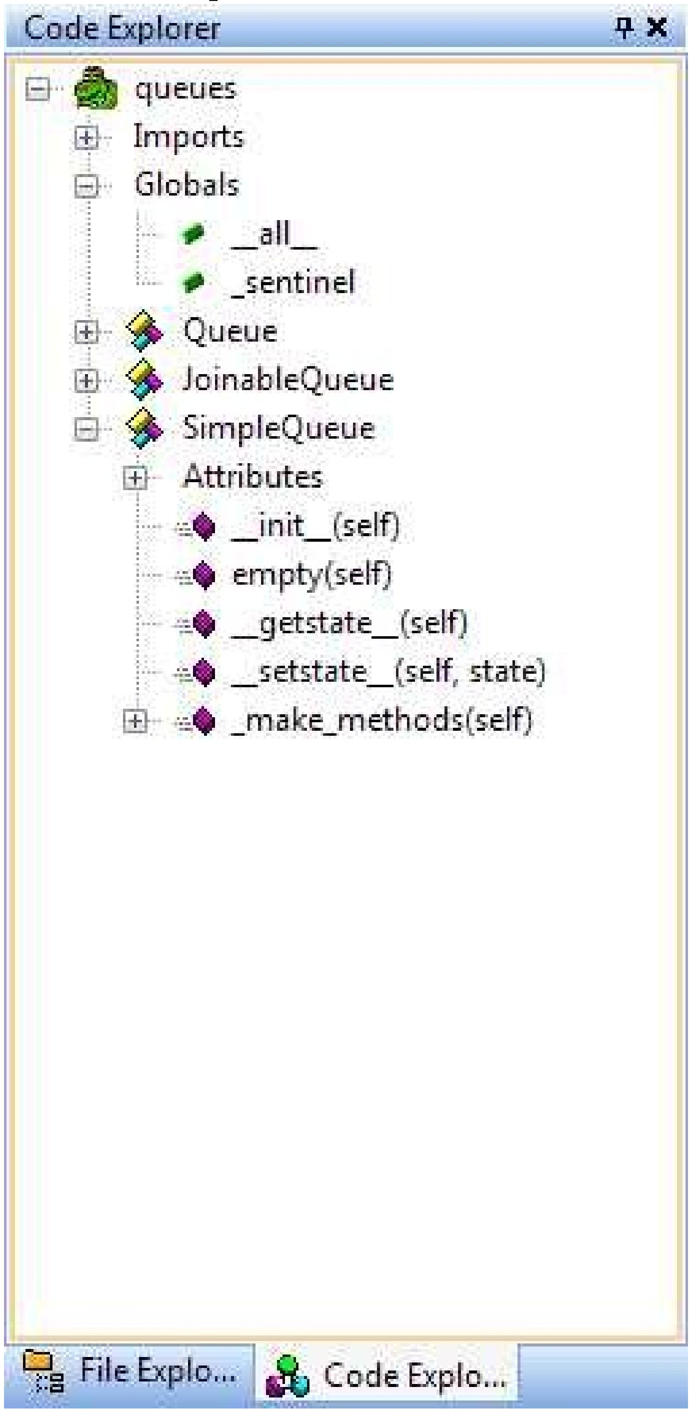
This dialog box allows you to add or remove folders from the Favourites list.



C:\Program Files\Python25\Lib

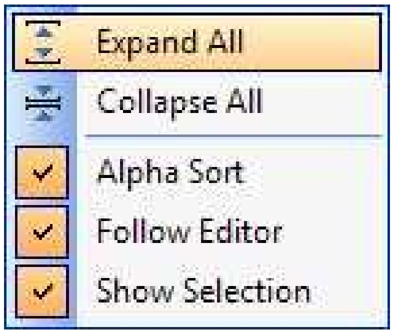
The Code Explorer Window

It shows a structured (tree) view of the source code with functions classes and their methods. It can help you navigate through the code.   Double-clicking on a any function or class name moves the editor caret to the section of the code where the respective function or class are defined.



The context menus

a) Window Background



Commands:

Expand All
Expand all nodes of the tree

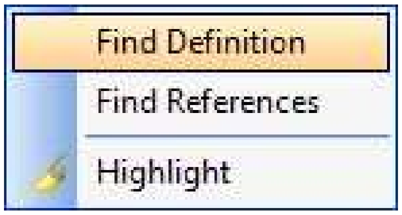
Collapse All
Collapse all nodes of the tree

Alpha Sort
If checked, tree nodes are sorted alphabetically, otherwise the node order follows the position of the identifiers in the code.

Follow Editor
If checked, as you move the cursor in the editor the class, method or function that contains the cursor gets selected in the Code Explorer.

Show Selection
If checked, when you select a node by mouse or keyboard, the position of the identifier in the code is shown without moving the focus to the editor.

b) Node context menu



Commands:

Find Definition
Moves the editor caret to the section of the code where the respective function or class are defined. Focus is shifted to the editor. It does the same as double-clicking on the identifier.

Find References
Invokes the Find References command for the selected identifier.

Highlight
If checked the occurrences of the selected identifier in the editor are highlighted.

The Project Explorer Window

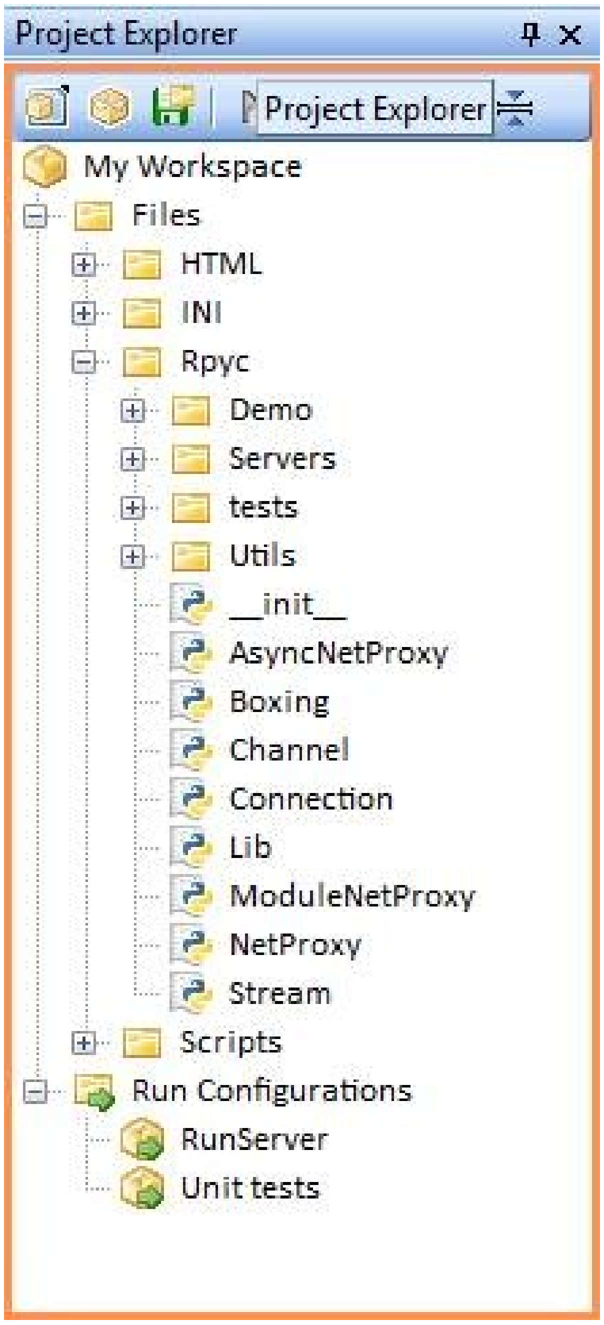
Previous Top Next

PyScripter projects server two purposes:

1. To create and maintain collections of files with which you tend to work together, structured hierarchically in folders and sub-folders. These folders, do not necessarily correspond to file system folders but instead they may based on some other logical categorization of files, for example based on their type (e.g Html, ini and script files grouped under different folders) . This similar to what other IDEs call Workspaces.
2. To create and maintain an associated set of Run configurations.

Other PyScripter Tools such as the Find in Files and the Todo list tools are designed can take advantage of projects.

PyScripter projects are saved as "ini" files have the default extension "psproj". At any point in time one such project is active and if that project is not saved it has the name "Untitiled". The Project Explorer IDE Window helps you explore and manage PyScripter projects.



The Root project node always has exactly two child nodes:

- Files
- Run Configurations

Under the "Files" node you can add files or folders which can contain further files and folders. Under the "Run Configurations" node you can add multiple Run Configurations, which can be used for running and debugging Python

Scripts.

Drag & Drop support

You can drag and drop files/folders from the built-in File Explorer or the Windows Explorer onto folder nodes to import these files and folders. You can also restructure the project by using drag and drop to move project nodes.

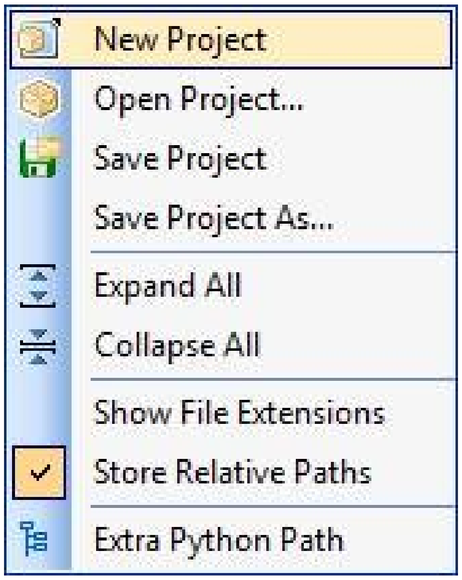
Opening files for editing

You can double click on files to open them in the editor. Alternatively you can select multiple files and select the "Edit" command from the context menu (see below).

Context sensitive menus

There are different context menus available for each type of node:

a) Window background and Root Project node context menu



Commands:

New Project
Clear the active project and start a new one.

Open Project
Open a saved project and replace the active one.

Save Project
Save the active project.

Save Project As...
Save the active project under a different name.

Expand All
Expand all project nodes

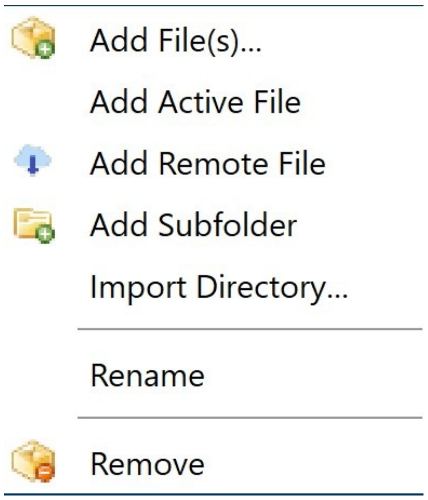
Collapse All
Collapse all project nodes

Show File Extensions
If this option is set, file extensions are shown, otherwise hidden

Store Relative Paths
If this option is set, project files that are in the same directory as the project file or a subfolder of the directory are saved as file paths relative to the Project file path.

Extra Python Path
This option allows a project specific customization of the Python Path. Extra directories specified are added to the Python path at the time the project is loaded and every time an engine is reinitialized.

b) Folder context menu



Commands:

Add File(s)...
Add on or more files to the selected folder using and Open File dialog.

Add Active File
Add the active editor file to the the selected folder.

Add Remote File
Add a remote file to the the selected folder using the remote file dialog.

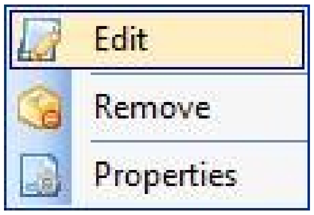
Add Subfolder
Add a subfolder to the selected folder.

Import Directory...
This command recursively imports a folder with its files and subfolders to the project replicating the directory structure on the disk.

Rename
Rename the selected folder.

Remove
Remove the selected folder and its child nodes.

c) File context menu



Commands:

Edit
Open the selected file in the editor. The same can be done by double-clicking the file name.

Remove
Remove the selected file from the project.

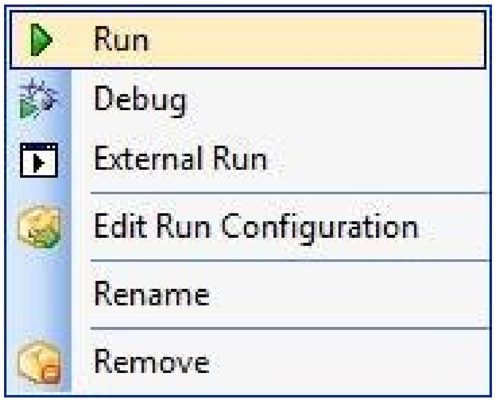
Properties
Show the standard Windows File Properties dialog for the selected file.

d) Run Configurations node context menu

Commands:

Add Run Configuration
Create and add to the project a new Run Configuration.

e) Run Configuration context menu



Commands:

Run
Run the selected Run Configuration.

Debug
Debug the selected Run Configuration.

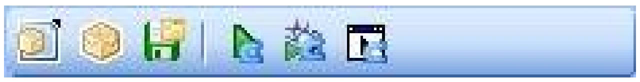
External Run
Run the selected Run Configuration using an external Python interpreter.

Edit Run Configuration
Edit the selected Run Configuration.

Rename
Rename the selected Run Configuration.

Remove
Remove the selected Run Configuration.

The Project Explorer Toolbar



Commands:

New Project

Clear the active project and start a new one.

Open Project
Open a saved project and replace the active one.

Save Project
Save the active project.

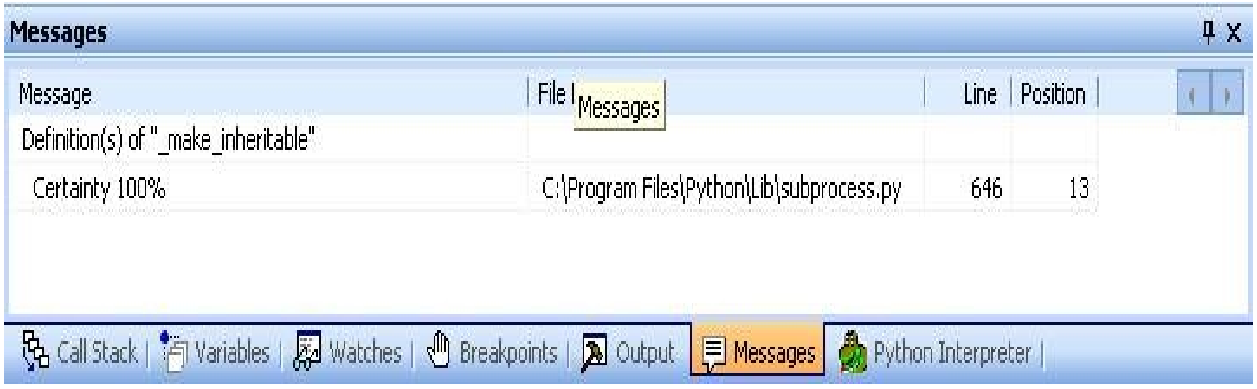
Run Last Configuration
Run the Configuration that was run last.

Debug Last Configuration
Debug the Configuration that was run last.

External Run Last Configuration
Run the Configuration that was run last using an external Python interpreter.

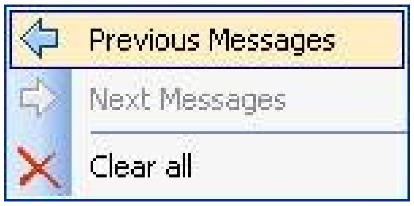
The Messages Window

The Messages Window serves the purpose of logging and displaying to the user syntax and runtime errors, Python tracebacks information and other warnings and messages. If there is File location information in a message you display that file and move the cursor to the respective position by double-clicking on that message. A history of up to 10 messages logs is maintained and you can browse through those message logs by using the arrow buttons at the top right-hand side of the window or the context menu.



Message	File	Line	Position
Definition(s) of "_make_inheritable"			
Certainty 100%	C:\Program Files\Python\Lib\subprocess.py	646	13

The Context Menu



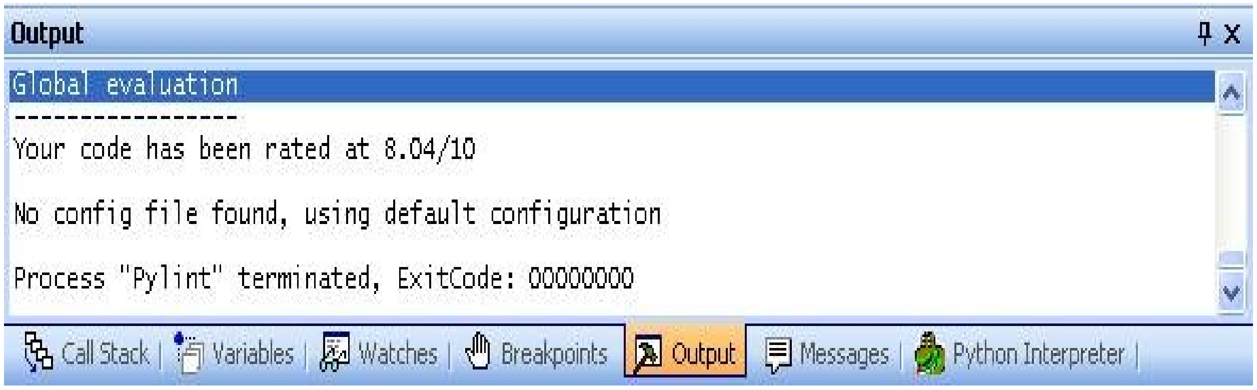
Commands:

Previous/Next Messages
Navigate through the history of message logs.

Clear all
Clear the whole history of message logs.

The Output Window

The Output window shows the captured output from external tools. Here it shows the output of the Lint style checker.

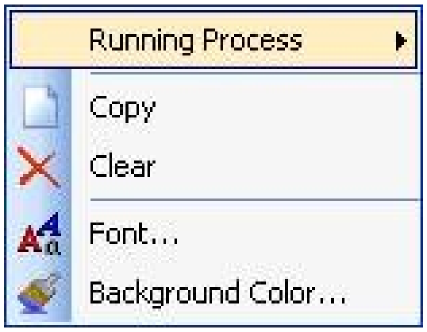


```
Global evaluation
-----------------
Your code has been rated at 8.04/10

No config file found, using default configuration

Process "Pylint" terminated, ExitCode: 00000000
```

The Context Menu



Commands:

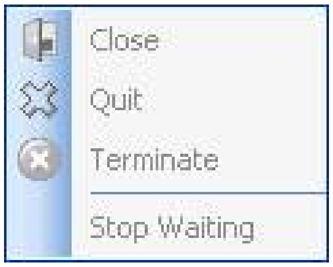
Copy
Copies the contents of the Output window to the Clipboard

Clear
Clears the contents of the Output window

Font..
Change the font of the Output window

Background Color..
Change the background color of the Output window

The running processes submenu



Commands:

These four commands are only available when an external tool is running with the "Wait for termination" option set.

Close
Posts a WM_CLOSE message to the running external process

Quit
Posts a WM_QUIT message to the running external process

Terminate
Terminates the running external process (use as last resort).

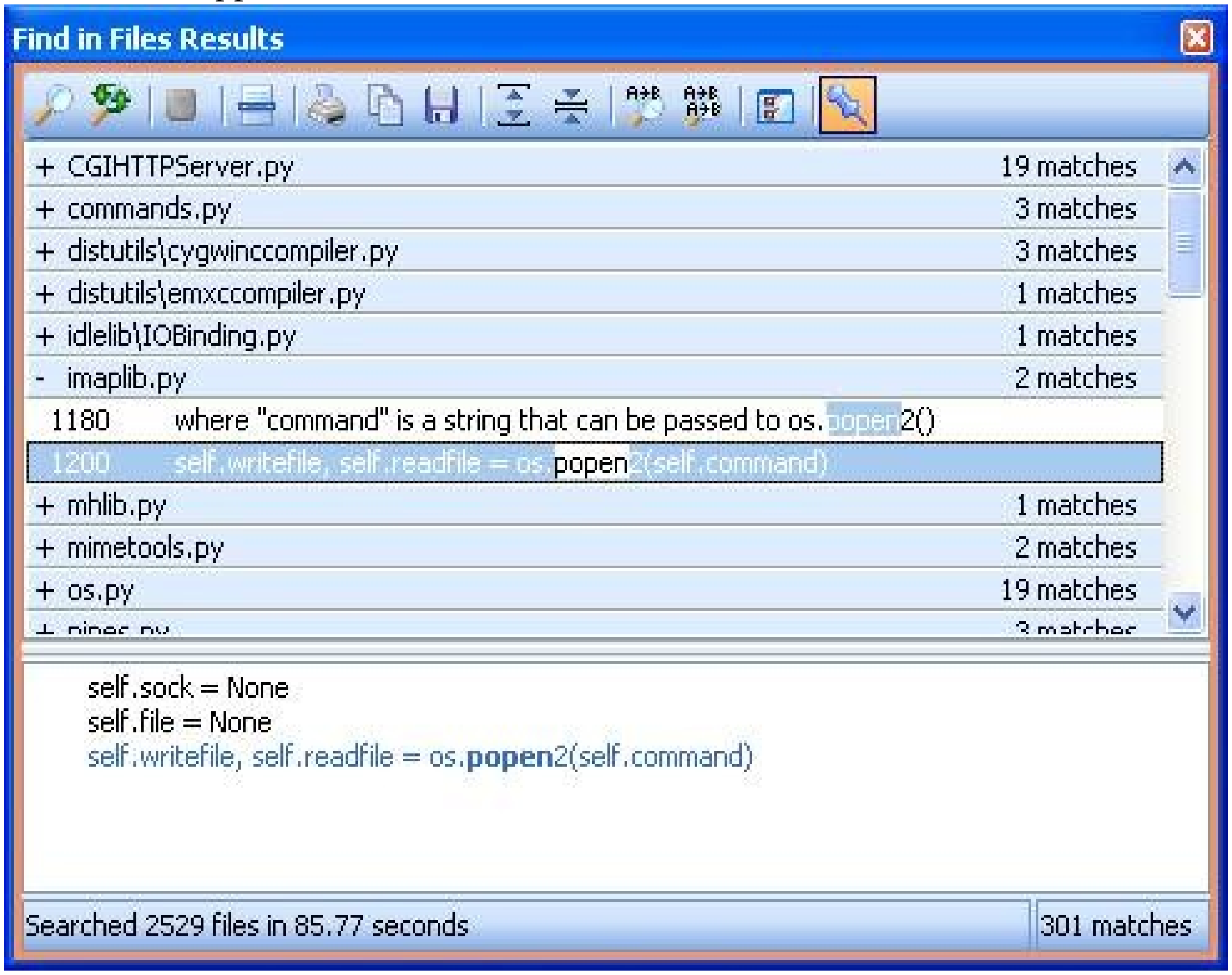
Stop Waiting
Stop waiting for the external process

Tip: The running processes submenu commands are also available by right clicking the External Processes indicator in the Status Bar.

The Find-in-Files Window

Previous Top Next

The Find-in-Files Results window is where the results of a Find-in-Files Search are shown. It also provides an interface for multi-file search and replace on matches. The window uses a folding display of matches to allow you to easily locate a particular match. This window supports IDE docking. An example of this window appears below.

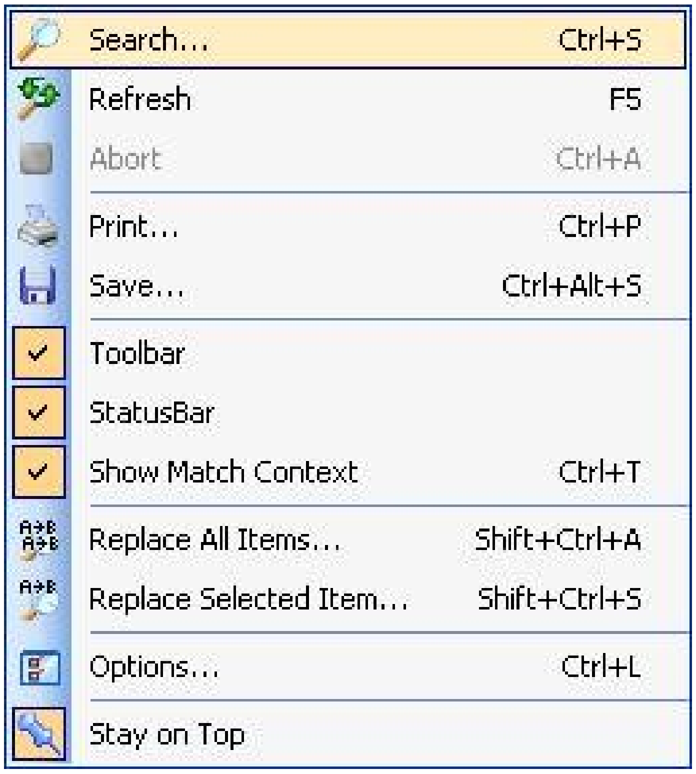


```
+ CGIHTTPServer.py                          19 matches
+ commands.py                                3 matches
+ distutils\cygwinccompiler.py               3 matches
+ distutils\emxccompiler.py                  1 matches
+ idlelib\IOBinding.py                       1 matches
- imaplib.py                                 2 matches
  1180    where "command" is a string that can be passed to os.popen2()
  1200    self.writefile, self.readfile = os.popen2(self.command)
+ mhlib.py                                   1 matches
+ mimetools.py                               2 matches
+ os.py                                     19 matches
+ pipes.py                                   3 matches
```

```
        self.sock = None
        self.file = None
        self.writefile, self.readfile = os.popen2(self.command)
```

```
Searched 2529 files in 85.77 seconds          301 matches
```

The Context Menu



The results window displays all files which contained one or more matches for the search term. Under each file, a list of matches for that particular file can be shown. To expand or contract a file's matches, click on the filename, press enter, or use the '+' and '-' keys. When a specific match line is selected, the window can show a number of lines of match context using the Show Match Context menu item. Note that the match context might not be accurate if you have edited the searched files since the search.

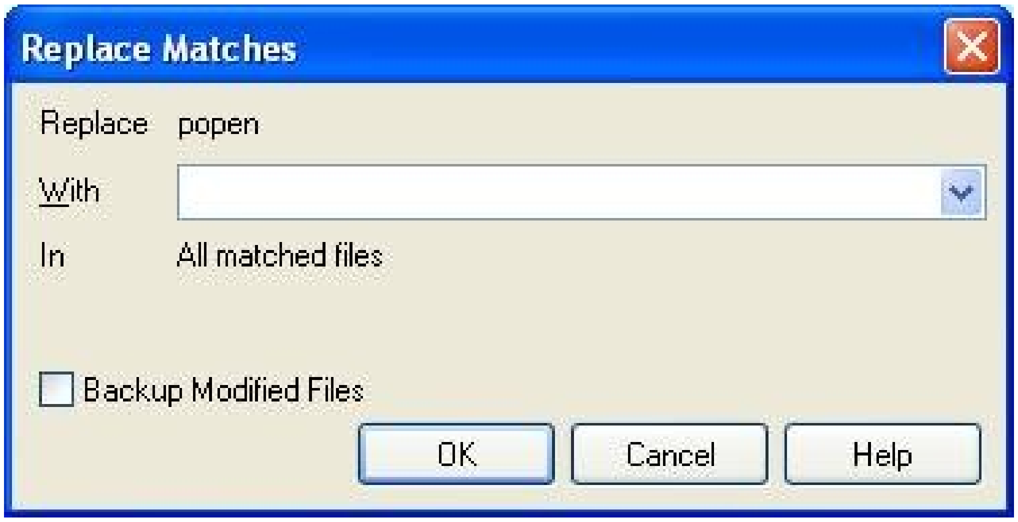
The number to the left of each match is the line number where the match was found. The results list highlights the matching characters in each entry to indicate where the match occurred.

To jump to a match in the IDE editor, double click the desired line, press enter, or use the Goto toolbar button. To start a new search, click the Search button and the Find-in-Files Search dialog will appear. As the search progresses, the new search button will be disabled and the abort button will be enabled to cancel the current search. Once a search is completed, the results window displays on the status bar the number of files searched, the search time, and the total number of matches.

You can expand all items in the list by clicking the expand button in the toolbar. Similarly, clicking the contract button in the tool bar will contract all result items.   The entire match list can be coped to the clipboard or saved to a file using the items on the File menu.

Search and Replace on Matches

You can do a search and replace operation on all of the matches in the list or only the selected file/match. When you choose one of those options, the dialog below appears prompting for the string to use in place of the matched text. If you were using a Regular Expression search you can use sub-expressions in the replace expression using the $x syntax, where x is the sub-expression number.


Replace Matches

Replace popen

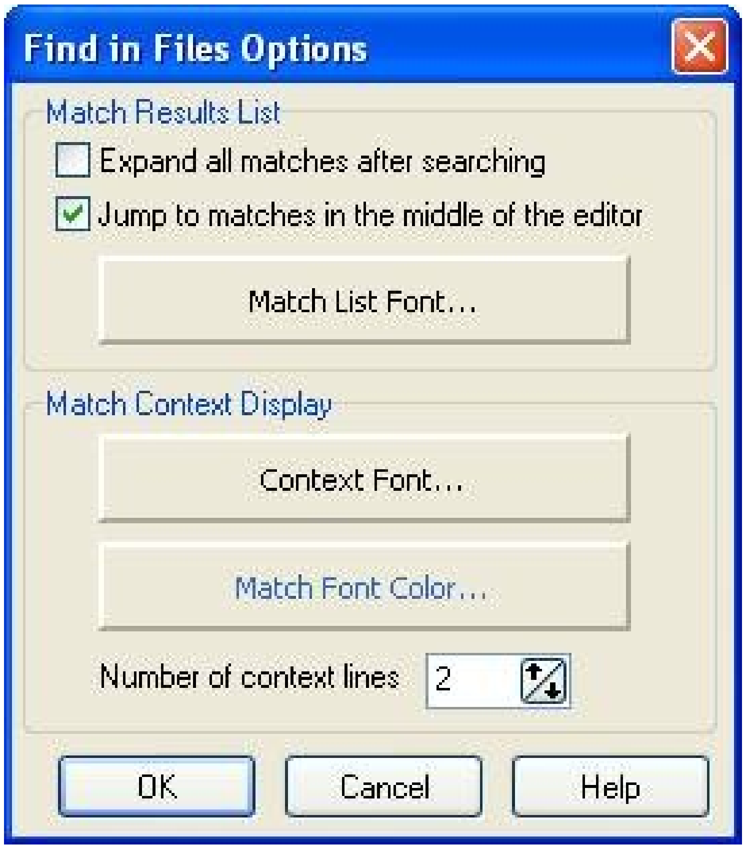
With

In All matched files

☐ Backup Modified Files

Find-in-Files Results options

Using the Options dialog below, define how selected matches are shown in the editor, whether to expand all matches by default, the number of context lines to show, and the list and context fonts/colors.


Find in Files Options

Match Results List

- [ ] Expand all matches after searching
- [x] Jump to matches in the middle of the editor

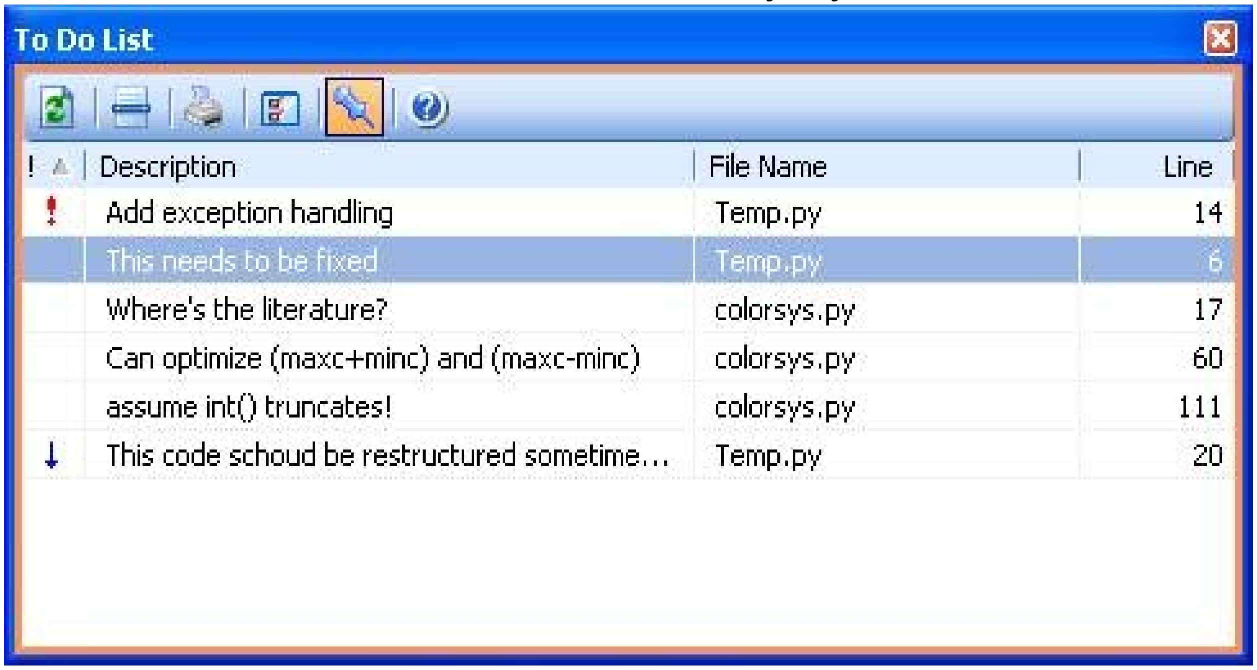
Match Context Display

Number of context lines: 2

Credits: This utility is based on code from the GExperts project (www.gexperts.org).

The To-do List Window

The To Do List helps you organize a list of items in your source that need special attention. You can click a column header to sort by any column in the list.



!	Description	File Name	Line
!	Add exception handling	Temp.py	14
	This needs to be fixed	Temp.py	6
	Where's the literature?	colorsys.py	17
	Can optimize (maxc+minc) and (maxc-minc)	colorsys.py	60
	assume int() truncates!	colorsys.py	111
↓	This code schoud be restructured sometime...	Temp.py	20

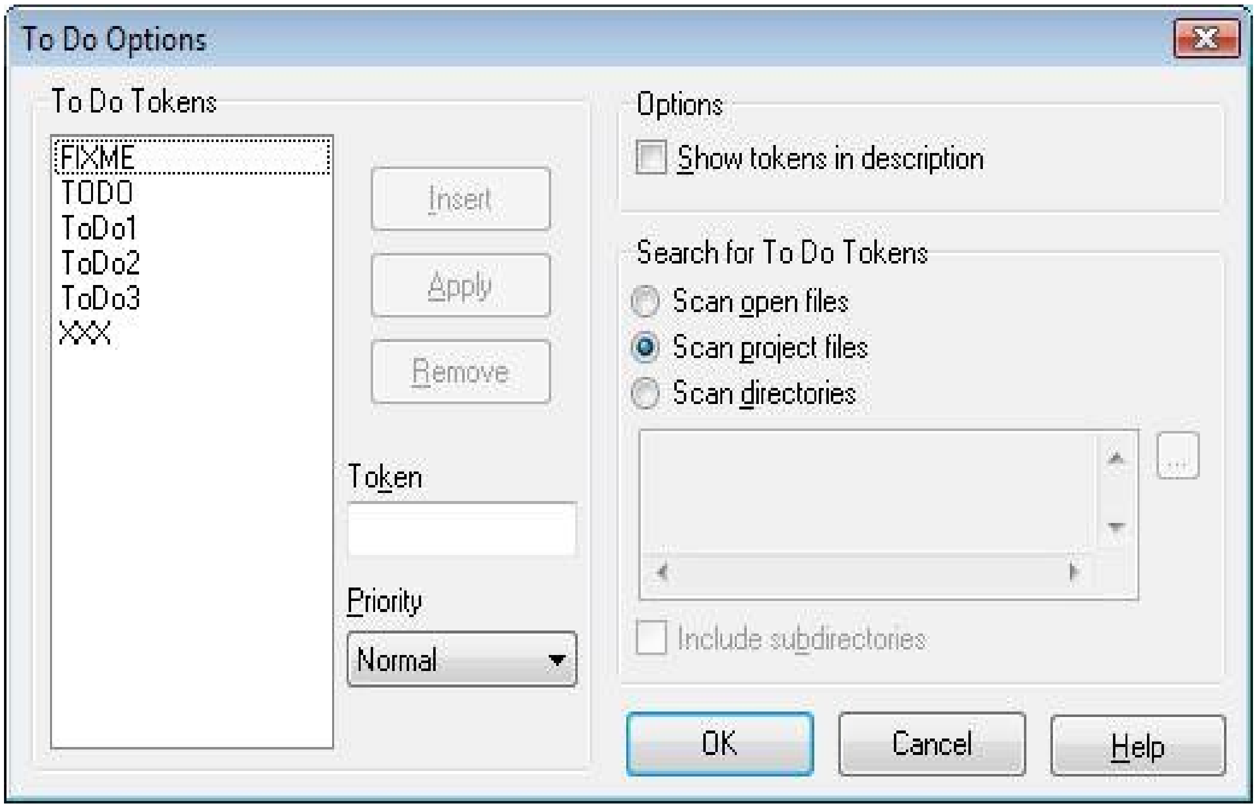
To add new to do items, type comments in your code such as:

#ToDo1 Rewrite this code to work under NT
#ToDo2 Add support for Oracle here later

To jump to a To-Do item in the IDE editor, double click the desired line, press enter, or use the Goto toolbar button. Using the buttons in the toolbar or the similar options in the context menu, you can refresh or print the To-Do list. Clicking in header columns sorts the items according to that column. Clicking the same column again changes the sort order.

To-Do options

The to do keywords (such as ToDo1, ToDo2, etc.) can each have an assigned priority of High, Medium, or Low. You can also add new keywords with associated priorities using the To-Do configuration dialog shown below. This utility can scan open files, all project files, or complete directories for to do items. It may be helpful to create an IDE Code Template to quickly create new to do items while you are coding.


To Do Options

To Do Tokens

FIXME
TODO
ToDo1
ToDo2
ToDo3
XXX

Token:

Priority: Normal

Options

[ ] Show tokens in description

Search for To Do Tokens

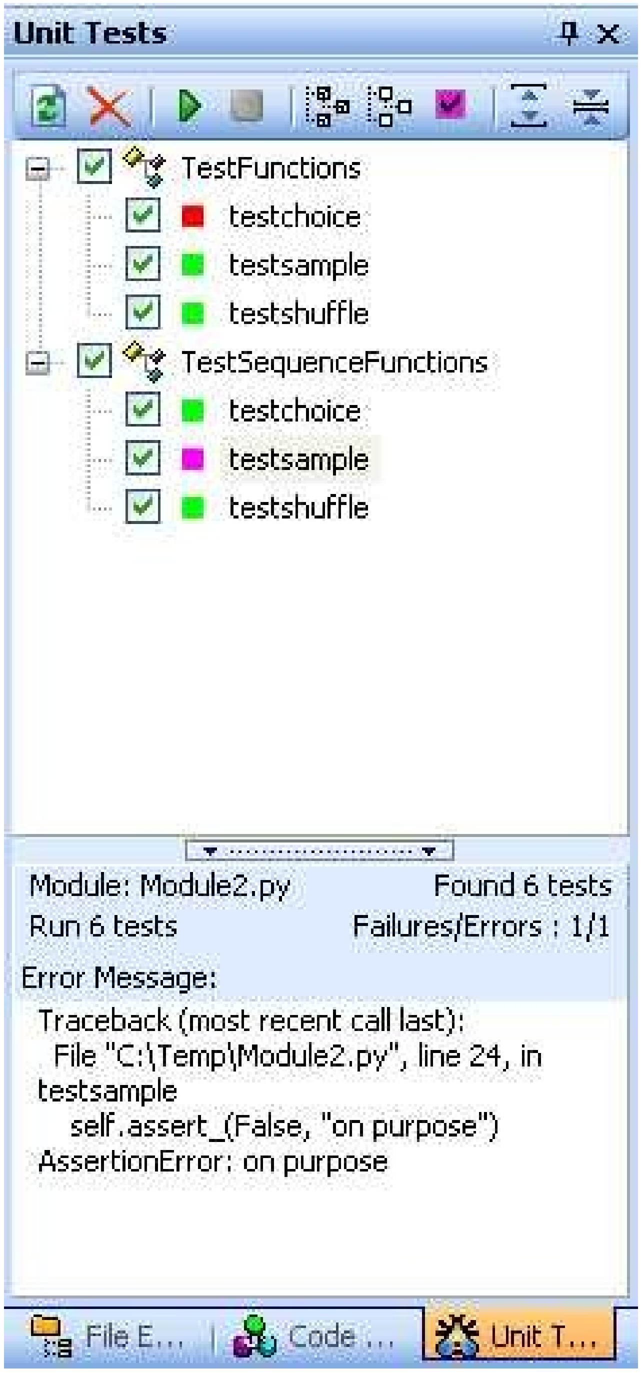
( ) Scan open files
(●) Scan project files
( ) Scan directories

[ ] Include subdirectories

Credits: This utility is based on code from the GExperts project (www.gexperts.org).

The Unit Tests Window

This window provides an advanced GUI for running tests based on unittest, the standard Python module.



Module: Module2.py Found 6 tests
Run 6 tests Failures/Errors : 1/1
Error Message:

```
Traceback (most recent call last):
  File "C:\Temp\Module2.py", line 24, in
testsample
    self.assert_(False, "on purpose")
AssertionError: on purpose
```

Toolbar Commands:

Refresh
Loads unit tests from the currently active module. Note that this involves importing the module into the integrated interpreter. After loading a module you can then select (check) the tests you want to run at the provided tree view. Double clicking on a test or a test class name takes you to the source code where the test method or class are defined.

Clear
Clears all the tests and related information from this window.

Run
Run the selected tests. After running the tests their status is indicated by the colour next to the test in the tree view. Green indicates success, Purple indicates assertion failure and Red indicates a Python exception (i.e. any other error). The pane below the tree view shows the overall statistics and you can view information about the errors that occurred by clicking on the tests that failed.

Stop tests
If clicked while running the tests the testing process stops.

Select all
Selects all available tests.

Deselect all
Deselects all available tests.

Select failed tests
After running a set of tests this command selects the tests that were not successful.

Expand all
Expands all tree nodes

Collapse all
Collapses all tree nodes

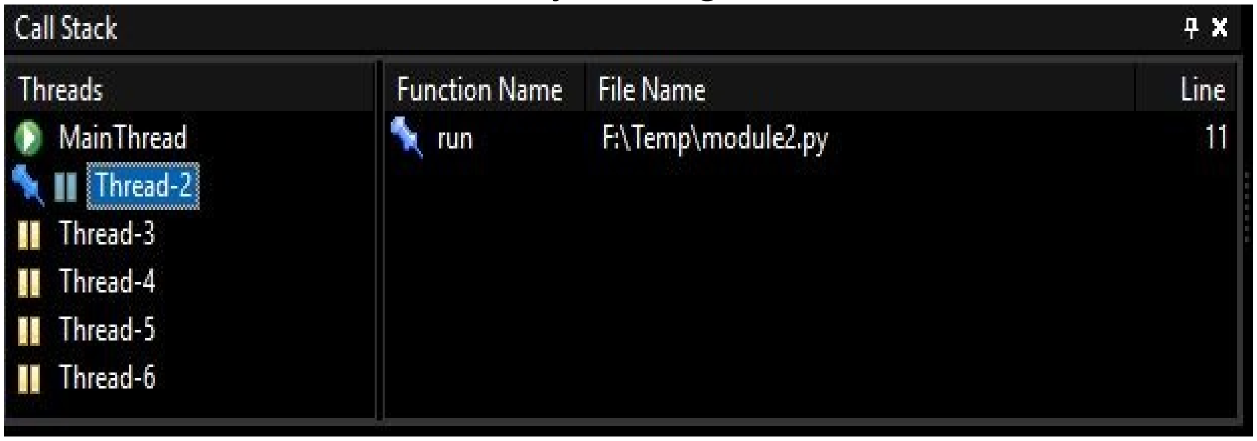
Note:
To use this GUI to run tests from multiple Python files (for example tests1.py, tests2.py and tests3.py) create a new script with the following content:

```
from tests1 import *
from tests2 import *
from tests3 import *
```

Then use the Unit Tests GUI with that file. (Press the Refresh button while this file is active).

The Call Stack Window

The call stack window includes a list of active threads and displays the Python interpreter call stack while debugging. It shows the function name and the corresponding source code position for each stack frame. You can jump to a given code position by double-clicking on a stack frame line.
The pinned thread is the active "broken" thread and the pinned frame is the active frame of the active thread. You should note that the Call Stack window works in tandem with the Variables window, which displays the local and global variables for the selected (active)stack frame in this window. Initially the top stack frame is selected in the Call Stack window. The Watches window evaluates watch expressions inside the active frame. Also commands you issue in the Interpreter window and debugger hints (hovering the mouse on variable names in the editor) are also evaluated inside the active frame. You can change the active thread and the active frame by selecting with the mouse a different one.



Threads
MainThread
Thread-2
Thread-3
Thread-4
Thread-5
Thread-6

Function Name	File Name	Line
run	F:\Temp\module2.py	11

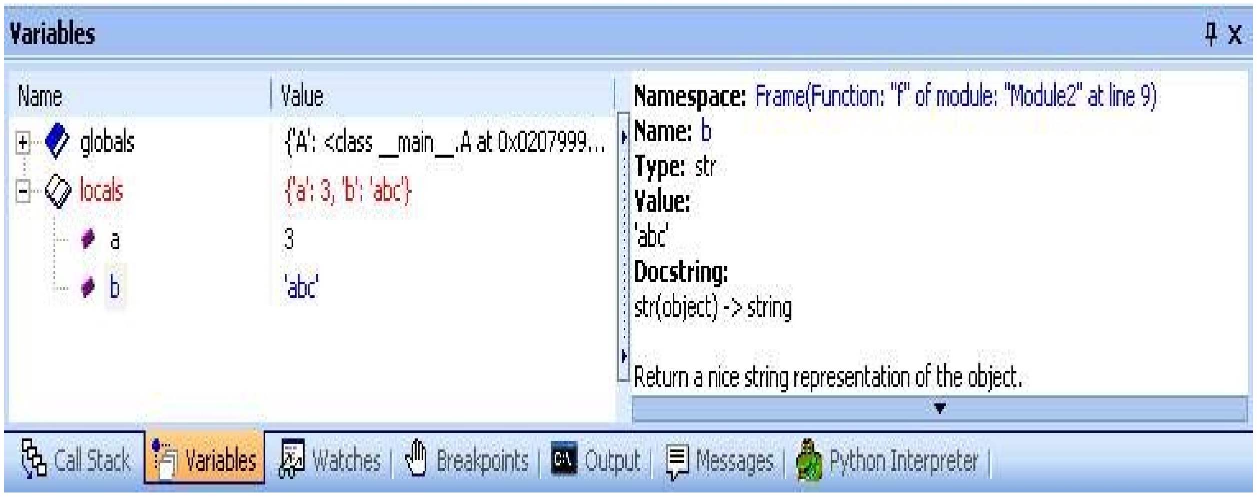
Debugger commands
The Resume command (F9) resumes execution of all broken threads. All other debug commands (e.g. Step in, Step over, Step out) resume execution of the active thread only.
Commands:
Two commands are provided to change the active stack frame using the keyboard and without having to switch view to the Call Stack Window.
Previous Frame
Select previous (older) frame (default shortcut F11)
Next Frame
Select next (newer) frame (default shortcut Shift+F11)

The Variables Window

Previous Top Next

During debugging and while the interpreter is stopped at a breakpoint , the Variables window displays the local and global variables for the selected stack frame in the Call Stack window which is usually the top frame. The left pane shows a hierarchical view with the value of each variable. Any Python object with a dictionary interface (classes, objects, dictionaries etc.) can be expanded so that key-value pairs are inspected. Variables that have been changed or are new while stepping through code are color coded. Changed variables are displayed with red color and new variables with blue color. The left hand pane of the Variables window displays the type, value and documentation of the selected variable.
When the debugger is not active the Variables window displays the global variables of the interpreter.

You cannot change the values of variables in this window. In fact you cannot change local function variables while debugging in Python (the locals dictionary is read-only). Global variables can be changed though in the Interactive Interpreter window.



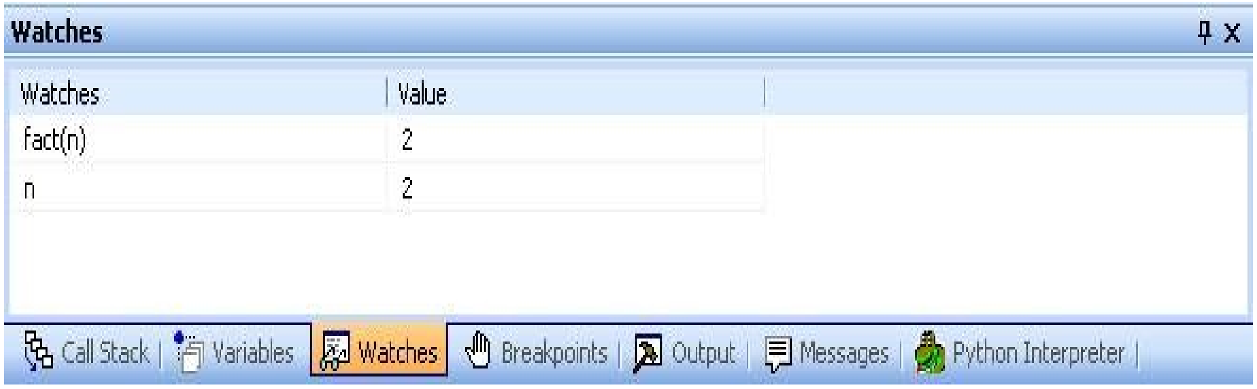
Name	Value
globals	{'A': <class __main__.A at 0x0207999...
locals	{'a': 3, 'b': 'abc'}
a	3
b	'abc'

Namespace: Frame(Function: "f" of module: "Module2" at line 9)
Name: b
Type: str
Value:
'abc'
Docstring:
str(object) -> string

Return a nice string representation of the object.

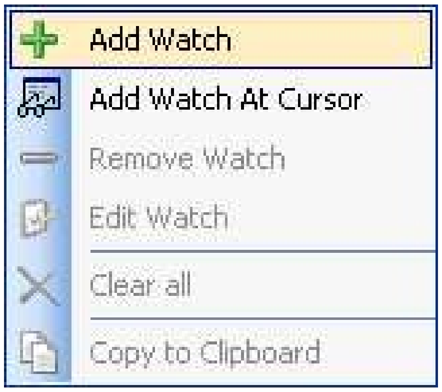
The Watches Window

This provides typical "Watch Expression" functionality found in most debuggers. You can set watches for arbitrary Python expressions. These expressions get re-evaluated as you step through the code or when you stop at breakpoints.



Watches	Value
fact(n)	2
n	2

The Context Menu



Commands:

Add Watch
Add a new Watch Expression

Add Watch at Cursor
Add the expression at the current editor position as a watch expression

Remove Watch
Remove the currently selected Watch Expression

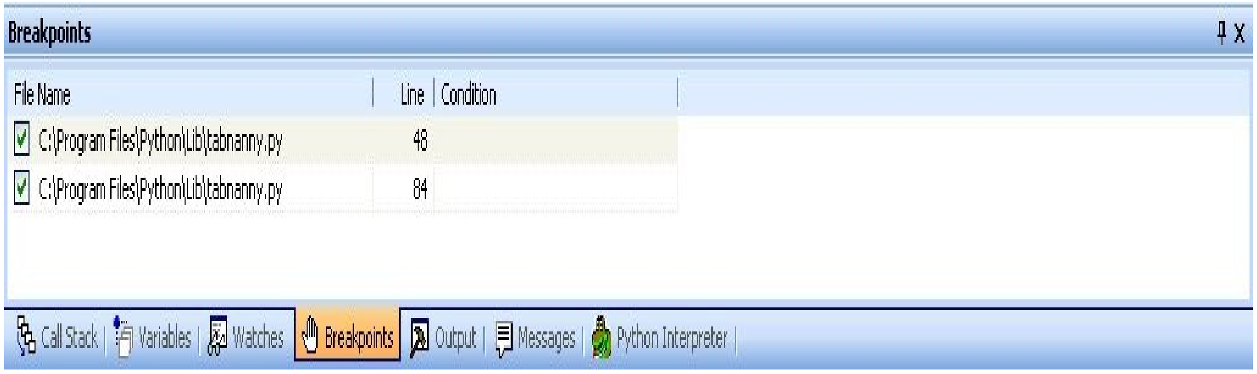
Edit Watch
Modify the currently selected Watch Expression

Clear All
Clear all Watch Expressions

Tips: You can double-click on a watch to edit it or you can double click on empty space to add a new watch. You can also drag&drop text from the editor to the Watches window to add it as a watch.

The Breakpoints Window

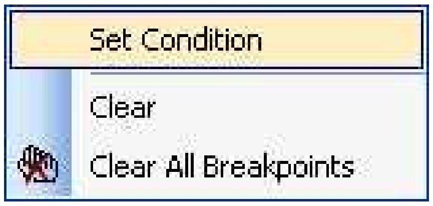
This window shows the breakpoints in all open Python scripts and modules. Double-clicking on a specific breakpoint takes you to the given code position.



File Name	Line	Condition
✓ C:\Program Files\Python\Lib\tabnanny.py	48	
✓ C:\Program Files\Python\Lib\tabnanny.py	84	

You can enable/disable a breakpoint by checking/unchecking the check-box at that start of the corresponding row. You can also apply a condition by specifying a Python expression using the context menu.

The Context Menu



Commands:

Set Condition
Specify a python expression to serve as a condition for the breakpoint. The execution will stop at this breakpoint only if the evaluated expression returns True.

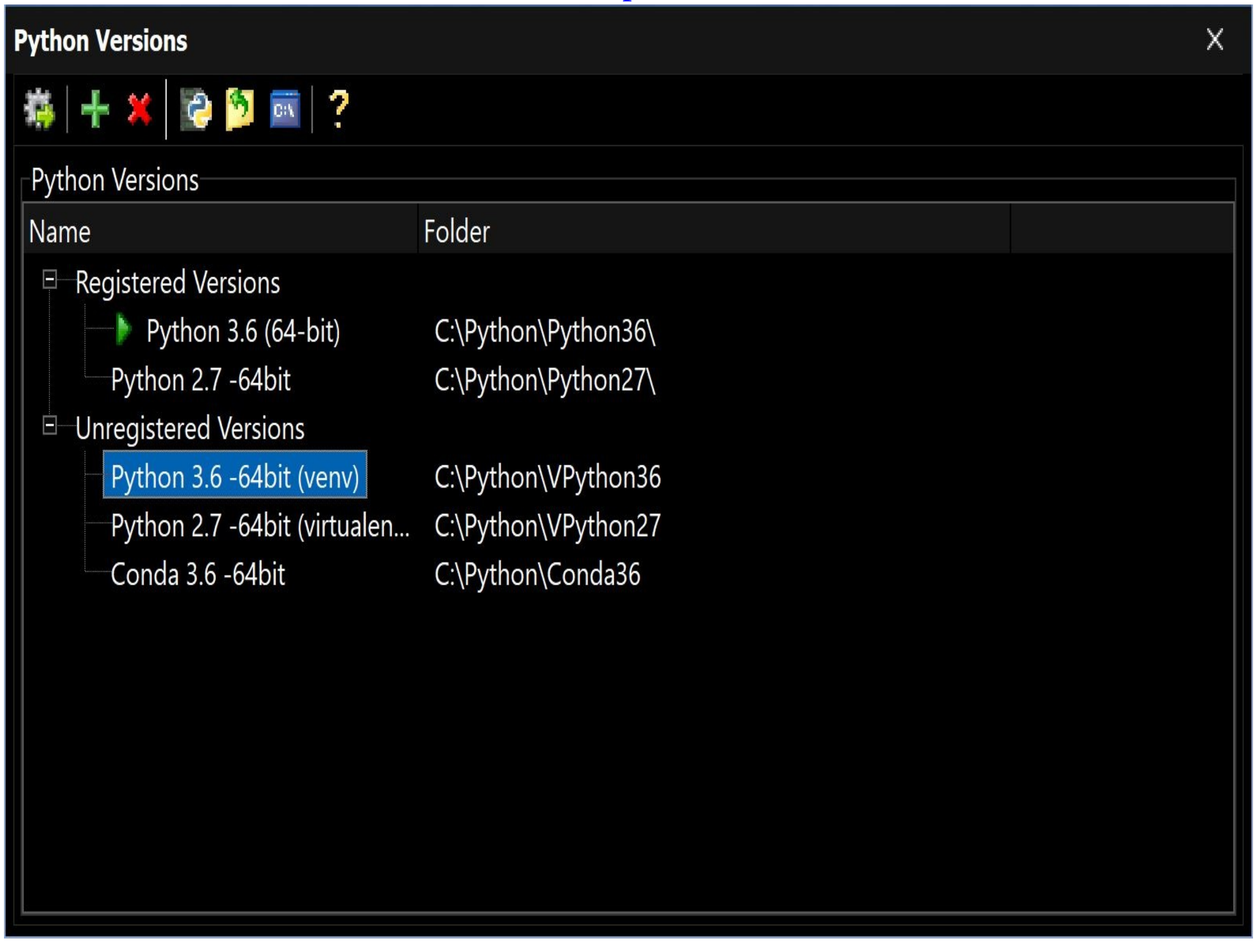
Clear
Clear the currently selected breakpoint

Clear All Breakpoints

Clear all breakpoints

Python Versions

The Python versions dialog allows you to switch python versions and environments as well as setup new ones. It is accessible from the Run menu or the context menu of the Interactive Interpreter. Here is how is looks:



Name	Folder
Registered Versions	
Python 3.6 (64-bit)	C:\Python\Python36\
Python 2.7 -64bit	C:\Python\Python27\
Unregistered Versions	
Python 3.6 -64bit (venv)	C:\Python\VPython36
Python 2.7 -64bit (virtualen...	C:\Python\VPython27
Conda 3.6 -64bit	C:\Python\Conda36

Commands (accessible from the toolbar):

Activate python version
Switches to and activates the selected python version.

Add python version
Adds a new python version from a directory selection.

Remove python version
Removes the selected python version. Registered versions cannot be removed.

Rename python version
Renames the selected python version. Registered versions cannot be renamed.
Setting the name to an empty string restores the default name.

Test python version
Opens a python interpreter with the selected python version.

Show python version
Shows the folder of the selected python version in the Windows File Explorer.

Command prompt
Opens a command prompt at the selected python version folder.

Help
Shows this help page.

Remote Python Engines

In addition to using the internal integrared Python engine, PyScripter offers you the option to use one of three remote Python engines. These remote engines run in a separate process, so, when using them, script errors should not affect the stability of PyScripter. You can select the python engine that will be active from the Python Engine submenu of the Run menu. Here is a brief explanation of the Python engine options:

Python Engines:

- **Internal (depricated)**
It is faster than the other options however if there are problems with the scripts you are running or debugging they could affect the stability of PyScripter and could cause crashes. Another limitation of this engine is that it cannot run or debug GUI scripts nor it can be reinitialized. Since version 3.1 the internal Python engine is hidden by default. This is controled by an IDE option.

- **Remote**
This the default engine of PyScripter and is the recommended engine for most Python development tasks. It runs in a child process and communicates with PyScripter using rpyc. Rpyc is bundled with the PyScripter destribution and no separate installation is required. It can be used to run and debug any kind of script. However *if you run or debug GUI scripts it is a good idea to reinitialize the engine before each run*. *This is done automatically by default.*

- **Remote Tk**
This Python engine is specifically designed to run and debug Tkinter applications including pylab using the Tkagg backend. It also supports running pylab in interactive mode. The engine activates a Tkinter mainloop and replaces the mainloop with a dummy function so that the Tkinter scripts you are running or debugging do not block the engine. You may even develop and test Tkinter widgets using the interactive console.

- **Remote Wx**
This Python engine is specifically designed to run and debug wxPython

applications including pylab using the WX and WXAgg backends. It also supports running pylab in interactive mode. The engine activates a wx MainLoop and replaces the MainLoop with a dummy function so that the wxPython scripts you are running or debugging do not block the engine. You may even develop and test wxPython Frames and Apps using the interactive console. Please note that this engine prevents the redirection of wxPython output since that would prevent the communication with Pyscripter.

- **SSH Engine**

This engine type runs a python interpreter in a remote Windows or Linux machine or inside a virtual environment (servers). You first need to define one or more SSH servers as explained in the topic Working with Remote Files. This topic also describes the requirements for using SSH with PyScripter. Once you choose this type of engine you need to select a defined SSH server. PyScripter starts a python engine on the remote server using SSH and communicates with it using rpyc. You can then run and debug remote or local scripts on the SSH server as if the scripts were running locally. You can also use python running inside the SSH server with the Python Interactive Interpreter. While debugging tracing into remote modules works transparently for the user. If you are running python 2.x locally the remote version also needs to be 2.x and similarly if you run python version 3.x locally the remote version needs to be 3.x. Beyond this constraint, the local and remote versions do not need to be the same.

Note: When using the Tk and Wx remote engines you can of course run or debug any other non-GUI Python script. However bear in mind that these engines may be slightly slower than the standard remote engine since they also contain a GUI main loop. Also note that these two engines override the sys.exit function with a dummy procedure.

Debugging Wx and Tkinter scirpts using the remote Wx and Tk engines

As mentioned above the Wx and Tk engines activate a main loop and replace the MainLoop with a dummy function. Therefore, when debugging Gui scripts using these engines, as soon as you reach the MainLoop statement debugging ends and you can then test the running application but without further debugging support. This means two things:

- Breakpoints and debugging would work up to the point the script enters the MainLoop routine

- You will not be able to debug event triggered code using these two engines.

To debug event code of Wx and Tkinter scripts use the standard remote engine. You may wonder why should you ever use the Wx and Tk specific remote engines. Here is a few reasons:

- These engine allow you to interactively develop and test frames and widgets. (possible because they run their own main loop.
- They support running pylab in interactive mode like IPython does, which was a request from many Pyscripter users.
- There is no need to reinitialize the engines after running Gui scripts.
- Pyscripter does not stay in running mode while the Gui Windows are showing but instead it returns in ready mode allowing further work and runs.

Troubleshooting

- If the remote Python engine becomes unresponsive you can try to reinitialize the engine from the Python Engine submenu of the Run menu (also available in the context menu of the interactive interpreter).
- If Pyscripter fails to start or appears locked when starting this may be due to remote python engines from earlier runs still being active. This could happen after a Pyscripter crash. In such cases you should kill the python engines using the Windows Task Managers. Look in the Processes tab for processes with image name "python.exe".

Running Scripts

Previous Top Next

There are many ways of running Python scripts from PyScripter:

- **Debug using the internal integrated Python debugger (depricated)**

Set any breakpoints you need and then from the Run menu select the Debug command. All the debugging facilities (step-into, step-out etc.) are available in this case. When the execution stops at a breakpoint or while stepping through the code you can use the Call Stack window, the Variables window and the Watches window to better understand the behavior of your code. All output is redirected to the Interpreter window. You can also use the Interpreter Window while debugging for running Python code in the context of the Call Stack frame at which the execution stopped for example if you want to change the value of a variable.

You can also start debugging by using the "Step-Into" and the "Run to Cursor" commands of of the Run menu. In that case executions stops at the first executable statement or the current line of the active module.

- **Run without debugging using the internal Python interpreter (depricated)**

Select the Run command from the Run menu. All output is again redirected to the Interpreter Window.

- **Run or debug using one of the remote Python engines**

Select an appropriate remote Python engine from the Python Engine submenu of the Run menu.
Then run or debug as when using the internal Python engine. See the Remote Python Engine topic for details.

- **Run externally from PyScripter**

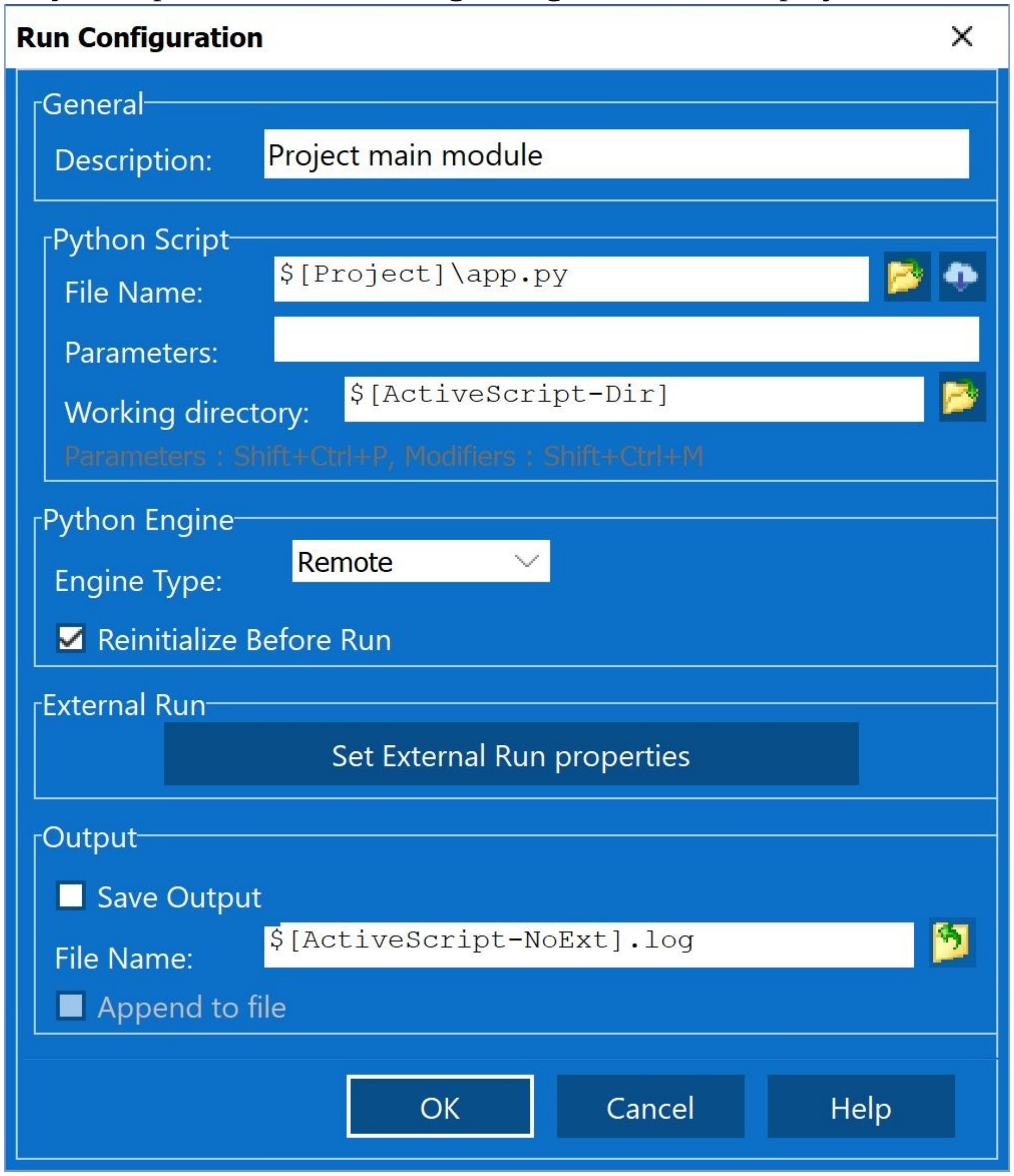
Select the External Run command from the Run menu. Extensive customization (choice of interpreter, command-line, environment variables etc.) is available through the Configure External Run command. The various settings are the same as in the External Tools configuration. By default output is captured and shown at the Output window.

For the first two cases you can set command line parameter using the Command Line command.

In addition, using the Project Explorer, you can create multiple Run Configurations with some more advanced options.

Run Configurations

Using the Project Explorer, you can create multiple Run Configurations which offer some more advanced options than those available when you run/debug the active script. You can create run configurations by selecting "Add Run Configuration" from the context menu of the "Run Configurations" node of the Project Explorer. The following dialog box is then displayed.



Run Configuration

General

Description: Project main module

Python Script

File Name: `$[Project]\app.py`

Parameters:

Working directory: `$[ActiveScript-Dir]`

Parameters : Shift+Ctrl+P, Modifiers : Shift+Ctrl+M

Python Engine

Engine Type: Remote

☑ Reinitialize Before Run

External Run

Set External Run properties

Output

☐ Save Output

File Name: `$[ActiveScript-NoExt].log`

☐ Append to file

Here follows an explanation of the main fields:

Description (optional):
A short description of the purpose of this run configuration. It is currently only used as a hint, when hover the mouse on top of the run configuration in the project explorer.

File Name:
The name of the script you would like to run. You can select a local or remote file using the buttons next to the edit box.

Parameters:
Command line parameters that are placed in the argv list of the Python sys module before running or debugging scripts. Note that the script name is automatically inserted as the first argument and should not be specified here.

Working Directory:
If specified the current directory will be changed to this one before running the script and restored back to the original one at the end of the run.

Python Engine:
The engine with which you would like to run or debug the script.

Reinitialze Before Run;
If checked, the Python engine will be reinitialized before running the script. This is necessary with some GUI scripts. This option is not available with the internal Python engine.

Save Output:
If checked the output of the script will be saved to the specified file.

Output File Name:
The path of the output file.

Append to File:
If checked output will be appended to that of earlier runs. Otherwise the most recent output will overwrite earlier output.

Set External Run properties:
Press this button to specify the options for running the script with an external Python interpreter. The various settings are the same as in the External Tools configuration. By default output is captured and shown at the Output window.

In entering the File Name, Parameters, Working Directory and Output file name you can use parameters and modifiers.

Executing Run Configurations
After defining a Run Configuration you can execute it in three different ways, by selecting the appropriate command from its context menu in the Project Explorer:

- Run
- Debug
- External Run

After the first execution you can use the commands
- Run last script (Shift+Ctrl+F9)
- Debug last script (Shift+F9)
- Run last script externally(Shift+Alt+F9)

to run/debug the last run configuration again.

This commands are available from the toolbar of the Project Explorer.

Post-Mortem Analysis

If your program raises an exception and stops, you can use the Post-Mortem command from the Run Menu to analyse the reason of failure. In this mode you can use the debugger windows (Call Stack, Variables, Watches) to examine the state of execution when the exception occurred and you can also issue interpreter commands in the context of the selected frame of the Traceback. To exit the Post-Mortem analysis you can use the "Abort Debugging" command.
The Post-Mort command is available only when the last script that you ran or debugged exited with an exception an until the next time you run or debug a script.

You may also set the IDE option "Post mortem on exception" to automatically enter the Post-Mortem when a script exits with an exception.

Using matplotlib with PyScripter

With the new Remote Python Engines you can now run matplotlib in interactive mode. Here is how:
1) Choose the right Python engine (Run, Python Engine menu option) for the backend of your choice e.g. Remote engine Tk for the "TkAgg" backend or Remote Engine wx for the "WX" and "WxAggg" backends.

2) Assuming that you have selected the Remote Engine wx, issue the following commands in the interpreter:

```
>>> import matplotlib
>>> matplotlib.interactive(True)
>>> matplotlib.use("WXAgg")
>>> from matplotlib.pylab import *
>>> plot([1,2,3])
>>> xlabel('time (s)')
```

You can set the backend and interactive mode in the matplotlibrc file. If this is done, the following is sufficient for the above example:

```
>>> from pylab import *
>>> plot([1,2,3])
>>> xlabel('time (s)')
```

3) Issue more pylab commands, or close the pylab window and call plot again for a new plot, etc.

Sample script by heylam

```
#Tested on: PyScripter 1.9.9.2, Matplotlib 0.91.2
#Assumed setup:
#In site-packages\matplotlib\mpl-data\matplotlibrc set backend to WXAgg.

def demoplot():
    '''This represents your work'''
    close('all') #closes figures
    t = c_[0:1:20j]
    s = cos(2*pi*t)
    for ampl in c_[0.1:1:10j]:
        plot(t, ampl*s, 'o-', lw=2)
    xlabel('time (s)')
    ylabel('voltage (mV)')
    title('demo', color='b')
    grid(True)
    draw() #update figure if already shown

#Select a demonstration:
if 0:  #Normal session
    #Starts non-interactive.
    #Figures have toolbar for zooming and panning.
    #Disadvantage: You can't re-run your script with PyScripter Remote
    # engine without first reinitializing the Remote interpreter.
    #Best use Remote(Wx) engine. This also allows interactive mode using
    # ion() and ioff(). For disadvantages: see the PyScripter help file.

    #from numpy import *
    #from scipy import *  #includes numpy
    from pylab import *   #includes scipy

    demoplot()
    show() #Open the figure, let figure GUI take over.
        #This should be last line of script.
        #You can also type this at command line after the script exits.

    if 0:
        ion() #turns interactive mode on   (needs Remote(Wx) engine!)
        ylabel('interactive modification')
        plot( rand(200), rand(200), 'go' )
        ioff() #turns interactive mode off

elif 0:  #Same but use WX instead
```

```
    try:
        type(matplotlib)
    except NameError:
        import matplotlib
        matplotlib.use('WX')
        from matplotlib.pylab import *

    demoplot()
    show() #Open the figure, let figure GUI take over.
        #This should be last line of script.
        #You can also type this at command line after the script exits.

elif 0:  #Same but start as interactive session, needs Remote(Wx)engine.
    try:
        type(matplotlib)
    except NameError:
        import matplotlib
        matplotlib.interactive(True)
        from matplotlib.pylab import *

    demoplot()
    show() #Open the figure, let figure GUI take over.
        #This should be last line of script.
        #You can also type this at command line after the script exits.

elif 0:#pdf output, allows use of Remote engine without re-initialization.
    #Disadvantage: no figure toolbar.
    #WARNING: close the file in acrobat reader before the next run.
    #(Maybe other pdf viewers don't block file overwrite?)
    try:
        type(matplotlib)
    except NameError:
        import matplotlib
        matplotlib.use('PDF')
        from pylab import *

    demoplot()
    filename='demo_plot'
    savefig(filename)

    #view the file:
    import win32api
    win32api.ShellExecute(0, "open", filename+'.pdf', None, "", 1)

elif 1:#png output, allows use of Remote engine without re-initialization.
    #Disadvantage: no figure toolbar.
    #Tip: make Irfanview your standard viewer.
    from pylab import *

    demoplot()
    filename='demo_plot'
    savefig(filename)
    #view the file:
    import win32api
    win32api.ShellExecute(0, "open", filename+'.png', None, "", 1)
```

Debugging Django Applications

Previous Top Next

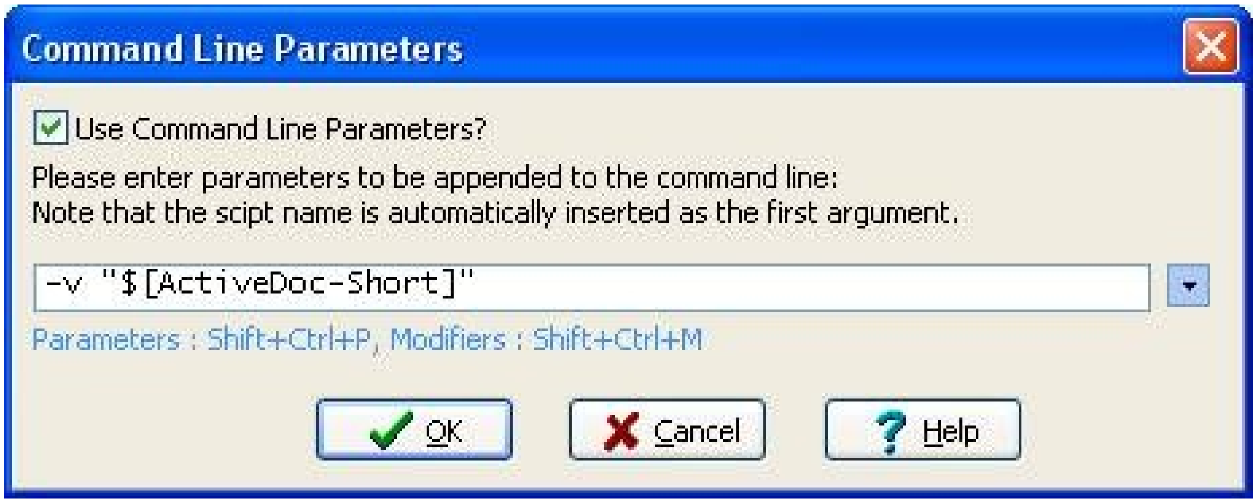
Here is how you can debug Django applications with Pyscripter in six simple steps.

1. In the File explorer locate the root directory of your Django application. You may want to right click on the directory name and select File Explorer, Explore here.
2. Open the project files in Pyscripter (e.g., models,py, views.py etc.) and set whatever breakpoints you want.
3. Select Run, Command Line Parameters... and set the command line to "runserver --noreload". Also check the "Use Command line" checkbox.
4. Make sure the remote engine is selected (Run, Python Engine, Remote).
5. Open the manage.py file and press the debug button (or press F9).
6. Start a web browser and test your application. Pyscripter should now stop at whatever breakpoints you have set and you can use the various debugging facilities (call stack, variables, interpreter prompt etc.)

To stop debugging, right-click on the interpreter window and select "Reinitialize Interpreter", then go to the browser and reload the document.

Command Line

You can specify command line parameters for scripts running internally or being debugged via the *Command Line Parameters...* command of the Run menu. This command invokes the following dialog in which you specify the command line parameters as well as enable or disable their use.



☑ Use Command Line Parameters?

Please enter parameters to be appended to the command line:
Note that the scipt name is automatically inserted as the first argument.

```
-v "$[ActiveDoc-Short]"
```

Parameters : Shift+Ctrl+P, Modifiers : Shift+Ctrl+M

The small button with the down arrow next to the edit field provides access to the most recently used command line parameters.

When command line parameters are enabled the provided parameters are placed in the argv list of the Python sys module before running or debugging scripts. Note that the script name is automatically inserted as the first argument and should not be specified here. In entering the command line you can use parameters and modifiers.

Note that:

- Shft+Ctrl+P provides Parameter completion
- Shft+Ctrl+M provides Modifier completion

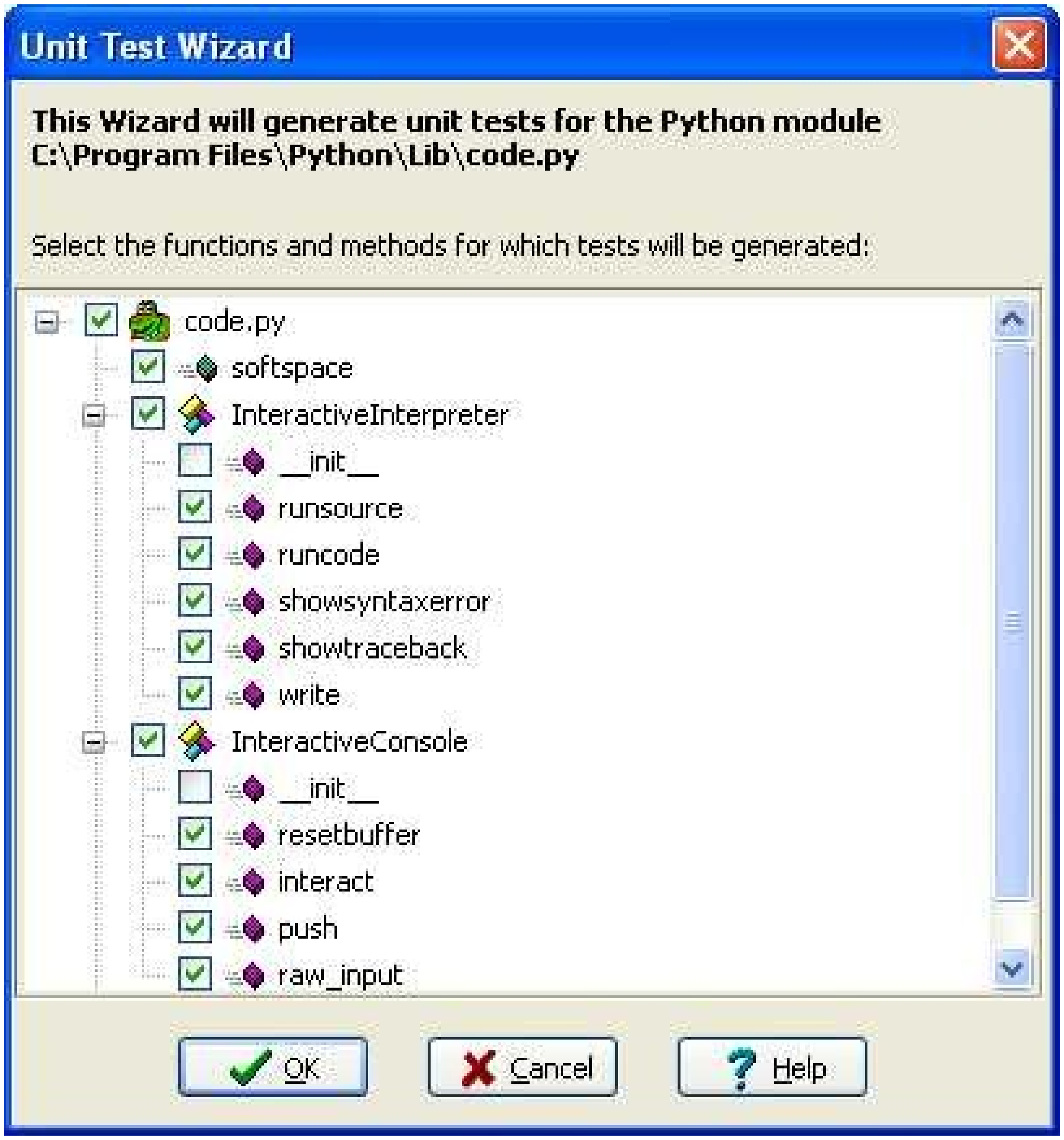
Unit Testing

Previous Top Next

PyScripter provides important support for unit testing. The support comes in two levels:

a) Automatic generation of test scripts

Use the Unit Test Wizard command from the Tools menu to generate the basic structure of a test script. This command invokes the following dialog box:



This Wizard will generate unit tests for the Python module C:\Program Files\Python\Lib\code.py

Select the functions and methods for which tests will be generated:

- [✓] code.py
  - [✓] softspace
  - [✓] InteractiveInterpreter
    - [ ] __init__
    - [✓] runsource
    - [✓] runcode
    - [✓] showsyntaxerror
    - [✓] showtraceback
    - [✓] write
  - [✓] InteractiveConsole
    - [ ] __init__
    - [✓] resetbuffer
    - [✓] interact
    - [✓] push
    - [✓] raw_input

The source code of the active module is scanned and its functions and methods are displayed. You can select (check) the functions and methods for which you want to generate tests and then press OK. A test script based on the unittest standard Python module is automatically generated for you. You can then write the code for each test that is generated.

b) GUI for unit testing

From the View Menu select Unit Tests to show the Unit Tests window (see this topic for a detailed description). This window provides an advanced user interface for running unit tests based on Python's unittest standard module.

Note:
To use this GUI to run tests from multiple Python files (for example tests1.py, tests2.py and tests3.py) create a new script with the following content:

```
from tests1 import *
from tests2 import *
from tests3 import *
```

Then use the Unit Tests GUI with that file.

Editor Features

The editing component of PyScripter is based on Synedit. Synedit is a highly customizable that offers a large number of features including:

- Syntax highlighting of Python files
- Code folding
- Drag and drop editing
- Line numbers
- Numbered bookmarks
- Practically unlimited file size
- Automatic detection of Unix and Mac files
- MBCS (multi-byte character set) support

PyScripter has adapted Synedit for the purpose of editing Python files and has added the following features:

- Context aware indentation of source code lines
- Code Completion
- Brace Highlighting
- Python source code utilities ((un)tabify, (un)comment, (un)indent)
- Context sensitive help on Python keywords
- Parameterized Code Templates
- Accept files dropped from Explorer
- File change notification
- Detecting loading/saving UTF-8 encoded files
- Converting line breaks (Windows, Unix, Mac)

Click below to find out about

- Customizing Editor Options
- Editor Shortcuts

Code Completion and Call Tips

Code completion and call tips are available both in the editor windows and in the interactive Python interpreter window.

Code Completion

When you type a qualified identifier (containing ".", e.g. sys.modules) as soon as you press the "." and after a short delay a list with all available members pops up from which you can select using the mouse or filter by typing the first few letters. The current selection in this list is copied to the interactive interpreter window as soon as:

- you press ENTER
- you press TAB
- you press ".", "(", ")", "[", "]" or space.

You can hide the code completion list by pressing the ESC key.

You may also activate code completion at any point either before you start writing an identifier name or after, as well as before typing the '.' or after, by pressing the **keyboard shortcut Ctrl+SPACE**. At all times you will get a filtered list of the names which are within the scope of the position at which you are within a module.

Support is also provided for the completion of the import statement. e.g. (^ stands for pressing Ctrl+Space)

```
import ^
import in^
import inspect as isp, cty^
from ^
from inspect import g^
from inspect import a as b, g^
from ..modname import a as b, g^
from .. import modname as m, another^
```

Call tips

When you open a left bracket "(" after typing a function name or a class name, and after a short delay, PyScripter pops up a call tip (hint window) with information about the expected parameters of the function you are entering as well as the doc string of the function if it is available. This call tip window stays on until you complete entering the function parameters and type the right bracket. You can hide a call tip by clicking on it).

You may also activate call tips at any point after you started writing the parameters of a function, by pressing the **keyboard shortcut Shift+Ctrl+SPACE**.

Note: The code and parameter completion should be one of the best you can find in any Python IDE. However,if you find that code and parameter completion is not very accurate for certain modules and packages such as wxPython and scipy you can achieve near perfect completion if you add these packages to the "Special Packages" IDE option (comma separated list). By default it is set to "os, wx, scipy". Special packages are imported on demand to the interpreter instead of scanning their source code.

Editor shortcuts

The following table presents the editor commands and the default associated shortcuts. These shortcuts can be changed using Editor Options. For further IDE commands and keyboard shortcuts see Keyboard Shortcuts.

Command	Shortcut	Description
Block Indent	Shft+Ctrl+I	Indent selection
Block Unindent	Shft+Ctrl+U	Unindent selection
Clear All		Delete everything
Column Select	Shft+Ctrl+C	Set Column selection mode
Context Help	F1	Context Sensitive Help
Copy	Ctrl+C Ctrl+Ins	Copy selection to Clipboard
Copy Line Down	Shft+Alt+Down	Copy line(s) down
Copy Line Up	Shft+Alt+Up	Copy line(s) up
Cut	Ctrl+X Shft+Del	Cuts selection
Delete BOL		Delete from cursor to beginning of line
Delete		Delete selection
Delete Char	Del	Delete next char
Delete EOL	Shft+Ctrl+Y	Delete from cursor to end of line
Delete Last Char	Bksp	Delete last char (i.e. backspace key)
Delete Last Word	Ctrl+Bksp	Delete from cursor to start of word
Delete Line	Ctrl+Y	Delete current line
Delete	Ctrl+T	Delete from cursor to end of

Word		word
Down	Down	Move cursor down one line
Editor Bottom	Ctrl+End	Move cursor to end of file
Editor Top	Ctrl+Home	Move cursor to start of file
Go To Marker i	Ctrl+i	Go to numbered bookmark i (i=0..9)
Insert Line	Ctrl+N	Break line at current position, leave caret
Insert Mode		Set editor mode to insert mode
Left	Left	Move cursor left one char
Line Break	Ctrl+M	Break line at current position, move caret to new line
Line End	End	Move cursor to end of line
Line Select	Shft+Ctrl+L	Set selection mode to Line
Line Start	Home	Move cursor to beginning of line
Lower Case	Ctrl+K, Ctrl+L	Change word at cursor or selection to lower case
Match Bracket	Ctrl+]	Go to matching bracket
Select to Bracket	Shift+Ctrl+]	Go to matching bracket extending the selection
Move Line Down	Alt+Down	Move line(s) down
Move Line Up	Alt+Up	Move line(s) up
Normal Select	Shift+Ctrl+N	Set selection mode to Normal
Overwrite mode		Set editor mode to overwrite mode
Page Bottom	Ctrl+PgDn	Move cursor to bottom of page
Page Down	PgDn	Move cursor down one page
Page Left		Move cursor left one page
Page Right		Move cursor right one page

Page Top	Ctrl+PgUp	Move cursor to top of page
Page Up	PgUp	Move cursor up one page
Paste	Ctrl+V Shft+Insert	Paste Clipboard to current position
Redo	Shift+Ctrl+Z Shft+Alt+BkSp	Perform redo if available
Right	Right	Move cursor right one char
Scroll Down	Ctrl+Down	Scroll down one line leaving cursor position unchanged
Scroll Left		Scroll left one char leaving cursor position unchanged
Scroll Right		Scroll right one char leaving cursor position unchanged
Scroll Up	Ctrl+Up	Scroll up one line leaving cursor position unchanged
Sel Down		Extend selection one line down
Sel Editor Bottom	Shft+Ctrl+End	Extend selection to the end of file
Sel Editor Top	Shft+Ctrl+Home	Extend selection to the start of file
Sel Left	Shft+Left	Extend selection one char left
Sel Line End	Shft+End	Extend selection to the end of line
Sel Line Start	Shft+Home	Extend selection to the start of line
Sel Page Bottom	Shft+Ctrl+PgDn	Extend selection to the bottom of the page
Sel Page Down	Shft+PgDn	Extend selection one page down
Sel Page Left		Extend selection one page left
Sel Page Right		Extend selection one page right
Sel Page Top	Shft+Ctrl+PgUp	Extend selection to the top of the page
Sel Page Up	Shft+PgUp	Extend selection one page up
Sel Right	Shft+Right	Extend selection one char right
Sel Up	Shft+Up	Extend selection one line up

Sel Word Left	Shft+Ctrl+Left	Extend selection one word to the left
Sel Word Right	Shft+Ctrl+Right	Extend selection one word to the right
Select All	Ctrl+A	Select all text in the editor
Set Marker i	Shft+Ctrl+i	Set numbered bookmark i at the current position (i=0..9)
Shift Tab	Shft+Tab	Action dependent on indentation options
Tab	Tab	Insert tab or spaces depending on editor options
Title Case	Ctrl+K, Ctrl+T	Change word at cursor or selection to title case
Toggle Case		Toggle the case of the word at cursor or selection
Toggle Mode	Ins	Toggle insert/overwrite mode
Undo	Ctrl+Z Alt+Bksp	Undo last action
Up	Up	Move cursor up one line
Upper Case	Ctrl+K, Ctrl+U	Change word at cursor or selection to upper case
Word Left	Ctrl+Left	Move cursor left one word
Word Right	Ctrl+Right	Move cursor right one word
Zoom in	Alt+'+'	Increase the font size of the editor
Zoom out	Alt+'-'	Decrease the font size of the editor
Code folding shortcuts		
Fold All	Shft+Ctrl+-	Fold all ranges
Unfold All	Shft+Ctrl+/	Unfold all ranges
Fold Nearest	Ctrl+/	Fold nearest range
Unfold Nearest	Shft+Ctrl+/	Unfold nearest range
Fold Regions		Fold all regions

Unfold Regions		Unfold all regions
Fold Level 1	Ctrl+K Ctrl+1	Fold top level ranges (level 1)
Unfold Level 1	Shft+Ctrl+K Shft+Ctrl+1	Unfold top level ranges (level 1)
Fold Level 2	Ctrl+K Ctrl+2	Fold level 2 ranges
Unfold Level 2	Shft+Ctrl+K Shft+Ctrl+2	Unfold level 2 ranges
Fold Level 3	Ctrl+K Ctrl+3	Fold level 3 ranges
Unfold Level 3	Shft+Ctrl+K Shft+Ctrl+3	Unfold level 3 ranges
Fold Functions		Fold all functions (IDE command)
Unfold Functions		Unfold all functions (IDE command)
Fold Classes		Fold all classes (IDE command)
Unfold Classes		Unfold all classes (IDE command)
Other shortcuts available in the editor		
Code completion	Ctrl+Space	Start code completion
Call tips	Shft+Ctrl+Space	Show a call tip

Code Templates

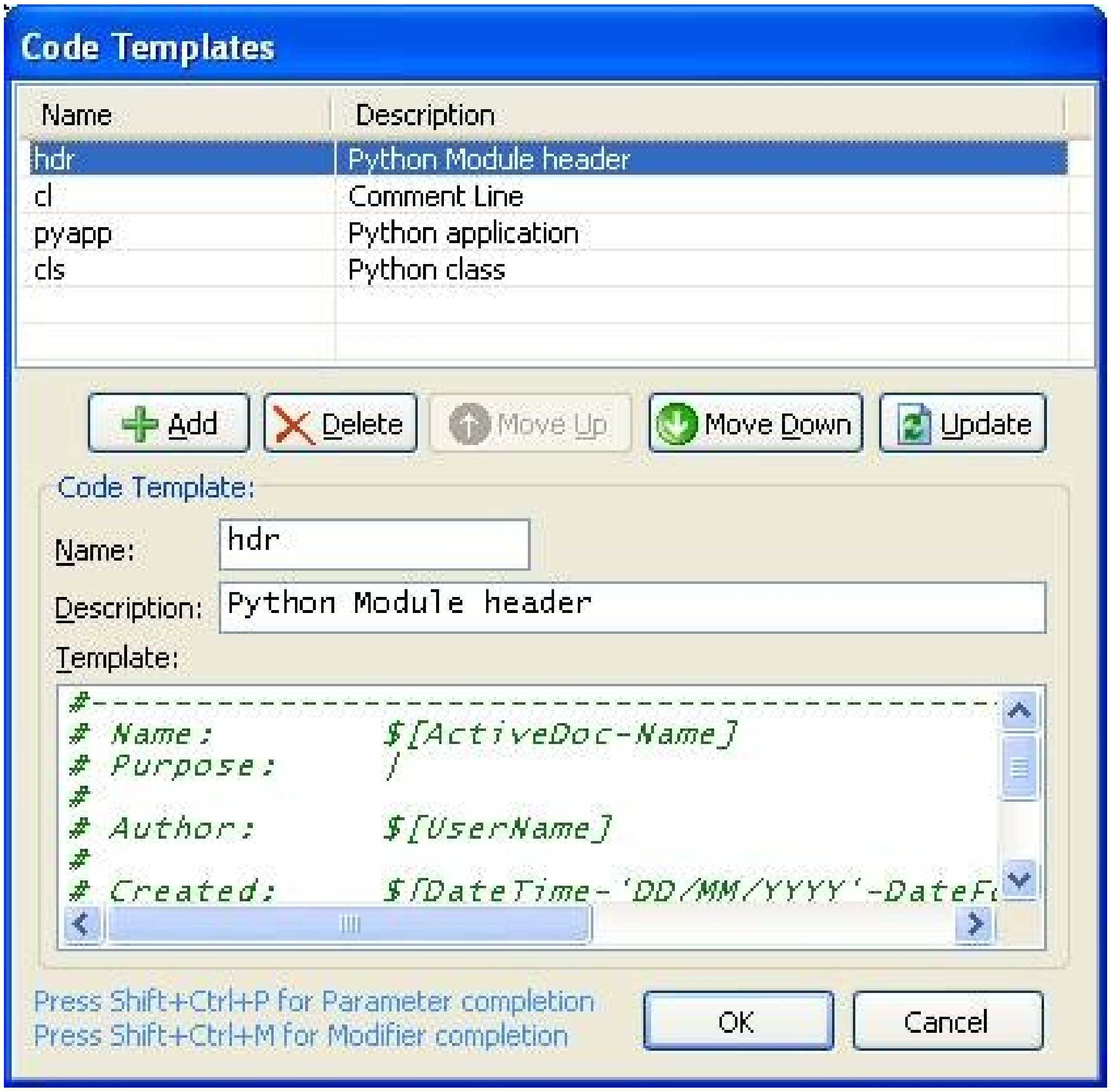
Previous Top Next

Code templates are snippets of code that can be inserted in the editor. Each template has a short name, a description and the associated text. The text of a code template can contain parameters which are automatically expanded when the text is inserted. For example the "Python Module Header" shown below contains the $[ActiveDoc-Name] and other parameters. If the character "|" is present in the template, after the insertion of the template text, the cursor is placed at the position of that character and the character is deleted.
How to insert a code template in the editor:

Type the template name and press the code template shortcut Ctrl-J. If you do not remember the name of template just press Ctrl-J and select from the pop-up list.

How to create/modify code templates:

Select "Customize Templates..." from the Tools|Options menu which displays the Code Templates dialog shown below.



Name	Description
hdr	Python Module header
cl	Comment Line
pyapp	Python application
cls	Python class

Code Template:

Name: hdr

Description: Python Module header

Template:

```
#-----------------------------------------
# Name:          $[ActiveDoc-Name]
# Purpose:       |
#
# Author:        $[UserName]
#
# Created:       $[DateTime-'DD/MM/YYYY'-DateF
```

Press Shift+Ctrl+P for Parameter completion
Press Shift+Ctrl+M for Modifier completion

Code and Debugger Hints

The Pyscripter editor supports code and debugger hints. These hints are displayed when you let the mouse hover on a Python code identifier.
a) Code Hints

These hints display information about the definition of the identifier under the mouse i.e. module name and line number in which the identifier is defined. Additional information is displayed according to the type of the identifier, e.g. parameters of functions, superclasses of classes or the type of variables. When the source code of the module in which the identifier was defined is available, the hint presents a hyperlink which, if clicked, takes you to the definition. This works in a similar way to the Find Definition feature and and the found definition is added to the Find Definition browsing history.

b) Debugger hints

While debugging code hints are not available, but instead debugger hints are displayed. These hints show the value and type of the identifier under the cursor. Debugger hints work also with expressions e.g. sys.path[1]. For debugger expression hints place the cursor on a closing parenthesis ')' or square bracket ']'

Note: Code and/or debugger hints can be disabled from the IDE options dialog box.

Encoded Python Source Files

PySCripter supports the PEP 263 fully. The editor internally uses Unicode strings. When saved, Python files can be encoded in either utf-8 or ansi encoding.

UTF-8 encoded source files

You can select this encoding from the File Formats submenu of the Edit menu. From that menu you can select whether UTF-8 encoded source files include the BOM UTF-8 signature which is detected by the Python interpreter. This signature is also detected by PyScripter when a file is loaded and other Windows editors. Although it is not necessary you are advised to include an encoding comment such as

# -- coding: utf-8 -*-*

as the first or second line of the python script. The advantage of using UTF-8 encoded files is that they can run without modification in other computers with different default encoding. *When using UTF-8 encoding you should specify all strings that are not plain ascii as python unicode stings by adding the prefix 'u'.*

ANSI encoded files

If the UTF-8 flag of the File Formats submenu of the Edit menu is not selected, then the file is treated as an ANSI string.   To define a specific source code encoding, a magic comment must be placed into the source files either as first or second line in the file, e.g.:

```
#!/usr/bin/python
# -*- coding: <encoding name> -*-
```

More precisely, the first or second line must match the regular expression "coding[:=]\s*([-\w.]+)". The first group of this expression is then interpreted as encoding name. If the encoding is unknown to Python, an error is raised during compilation. There must not be any Python statement on the line that contains the encoding declaration. If such a comment is not present then the default system encoding is assumed. PyScripter detects such comments when it loads Python Source files and decodes them to Unicode using the appropriate encoding.

The default python encoding is controlled by a Python file called "site.py" which is located in the python lib directory (see function "setencoding" in site.py). The default encoding when python is installed is ascii, which does not support non-ascii characters (character value greater than 127). If you are planning to use non-ascii strings in Python without using the utf-8 encoding, you will need to modify site.py and enable support for a locale aware default string encoding.

IDE encoding options for new files

Pyscripter provides two IDE options controlling the encoding of new files:

- *Default line breaks for new files*
- *Default encoding for new files*

IDE option for detecting UTF-8 encoding when opening files

Another IDE option (*Detect UTF-8 when opening files*) controls whether PyScripter attempts to detect utf-8 encoding when opening files without the BOM mark. This detection is done by analyzing the first 4000 characters of the file and is imperfect. It only applies to non-Python files since utf-8 encoded Python files are required to have either the BOM mark or an encoding comment.

Editor Fonts

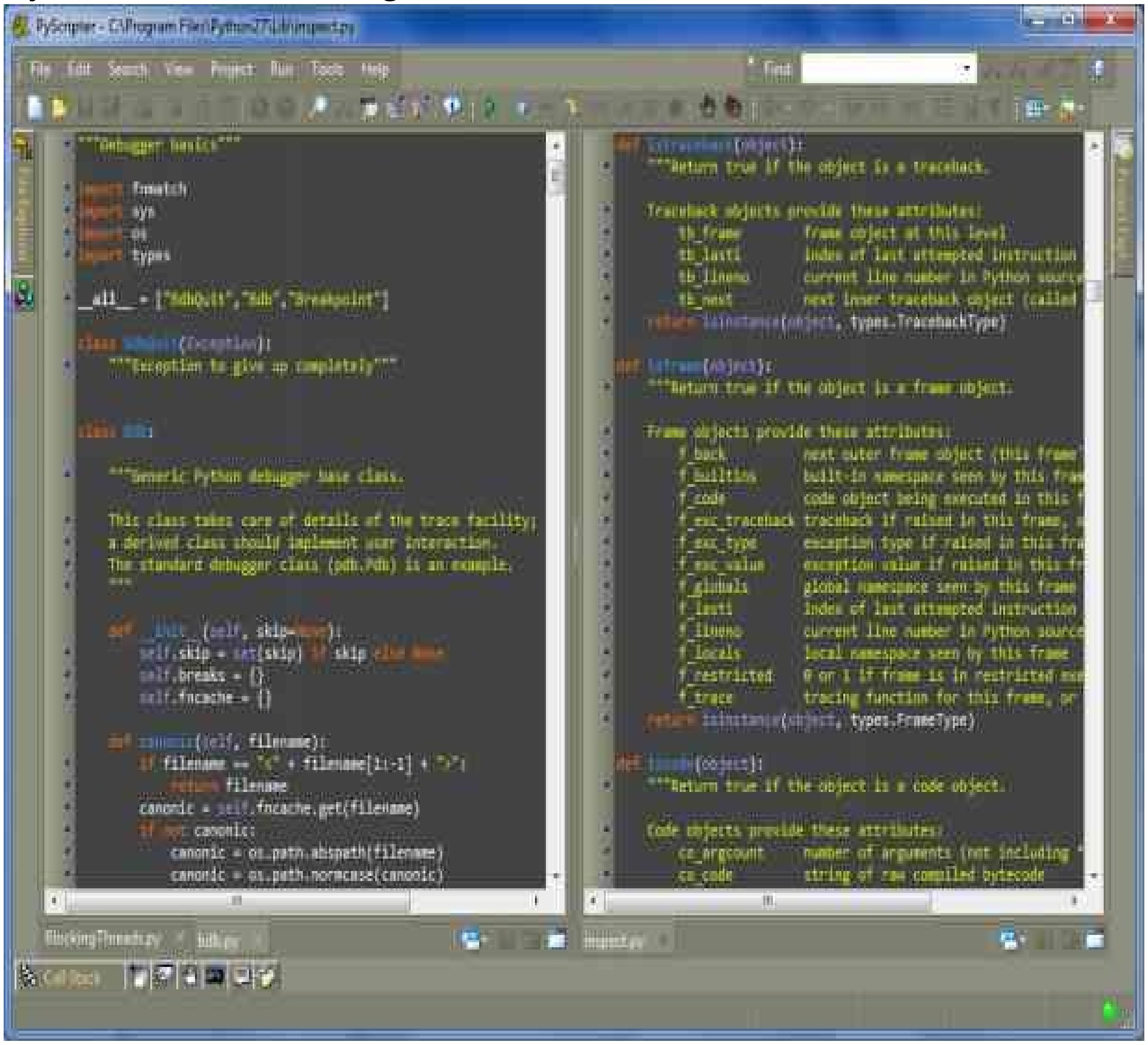
Previous Top Next

Pyscripter requires the use of monospace or non-proportional fonts. If you find the choice of fonts available in your Windows system limited you may want to have a look at http://www.lowing.org/fonts/. Microsoft has also produced a new attractive monospace font called Consolas available from Microsoft. Please note Consolas can only be installed if Visual Studio
is also installed. Another site with fixed width fonts you may want to try is http://www.proggyfonts.com/.
And by the way if you are using an LCD screen you may want to turn Clear Type on. To do so and fine tune Clear Type visit http://www.microsoft.com/typography/cleartype/tuner/Step1.aspx. (You need to visit with Microsoft Explorer). Clear Type can make a huge difference!

I personally use Consolas font size 10 with Clear Type turned on. (screenshot)

Side-by-Side File Editing

You initiate side-by-side file editing by using the "Split Workspace" commands of the View menu. These commands add a second set of editor tabs to achieve a layout such as the following:



```
"""Debugger basics"""

import fnmatch
import sys
import os
import types

__all__ = ["BdbQuit","Bdb","Breakpoint"]

class BdbQuit(Exception):
    """Exception to give up completely"""


class Bdb:

    """Generic Python debugger base class.

    This class takes care of details of the trace facility;
    a derived class should implement user interaction.
    The standard debugger class (pdb.Pdb) is an example.
    """

    def __init__(self, skip=None):
        self.skip = set(skip) if skip else None
        self.breaks = {}
        self.fncache = {}

    def canonic(self, filename):
        if filename == "<" + filename[1:-1] + ">":
            return filename
        canonic = self.fncache.get(filename)
        if not canonic:
            canonic = os.path.abspath(filename)
            canonic = os.path.normcase(canonic)
```

```
def istraceback(object):
    """Return true if the object is a traceback.

    Traceback objects provide these attributes:
        tb_frame        frame object at this level
        tb_lasti        index of last attempted instruction
        tb_lineno       current line number in Python source
        tb_next         next inner traceback object (called
    return isinstance(object, types.TracebackType)

def isframe(object):
    """Return true if the object is a frame object.

    Frame objects provide these attributes:
        f_back          next outer frame object (this frame
        f_builtins      built-in namespace seen by this fra
        f_code          code object being executed in this f
        f_exc_traceback traceback if raised in this frame, o
        f_exc_type      exception type if raised in this fra
        f_exc_value     exception value if raised in this fr
        f_globals       global namespace seen by this frame
        f_lasti         index of last attempted instruction
        f_lineno        current line number in Python source
        f_locals        local namespace seen by this frame
        f_restricted    0 or 1 if frame is in restricted mod
        f_trace         tracing function for this frame, or
    return isinstance(object, types.FrameType)

def iscode(object):
    """Return true if the object is a code object.

    Code objects provide these attributes:
        co_argcount     number of arguments (not including *
        co_code         string of raw compiled bytecode
```

You can also show two files side-by-side one above the other (horizontally aligned).

You can add tabs to the secondary tab set in a number of ways:

- Drag&drop tabs from the primary tab set (works in the opposite direction).
- Open files while the secondary tabs is active.

- Use the Recent Files menu while the secondary tab set is active.
- Use the context menu of the secondary tab set to create new modules.

When you hide the secondary editor tab set all contained tabs are moved to the primary tab set. If you just want to maximize the editing space of the primary tab set you can temporarily hide the secondary tabs by double-clicking the handle of the splitter between the two tab sets. You can later restore the previous layout by double-clicking the handle of the splitter again.

The layout of the secondary workspace and the contained tabs are persisted across PyScripter invocations.

Working with Remote Files

Previous Top Next

Pyscripter supports working with remote files i.e. files that may reside in different computers (servers) including Windows and Linux machines. You can open, edit run debug and save back these files. They work seamlessly with other PyScripter features such as the Recent File list, project files, and Run Configurations.

Requirements

To use PyScripter with remote files your computer need to have SSH client capabilities at the computer running PyScripter and an SSH server running on the remote computer. SSH is a widely used network protocol for securely connecting to remote machines. Windows 10 since the April 2018 update includes SSH. With earlier versions of Windows 10 you need to manually enable SSH through "Enable Optional Features". For other versions of Windows you can install the latest version of OpenSSH for Windows using the provided installation instructions.

Alternatively on the client side you can use the popular SSH client PuTTY.

Configuring the SSH client side

Pyscripter requires password-less authentication using rsa keys. You need to create the rsa keys and add them to the ssh-agent service which needs to be running. Your public key needs to be added to the ~/.ssh/authorised_keys file on the server side. Instructions are provided here.

Configuring the SSH server side

The SSH server service (sshd) and SSH agent service (ssh-agent) need to be running on the server side. This is most likely true for Linux machines. In Windows machines you need to start the server using the

```
net start sshd
net start ssh-agent
```

commands.   You can also configure these services to run automatically at login time.

Testing the SSH connection

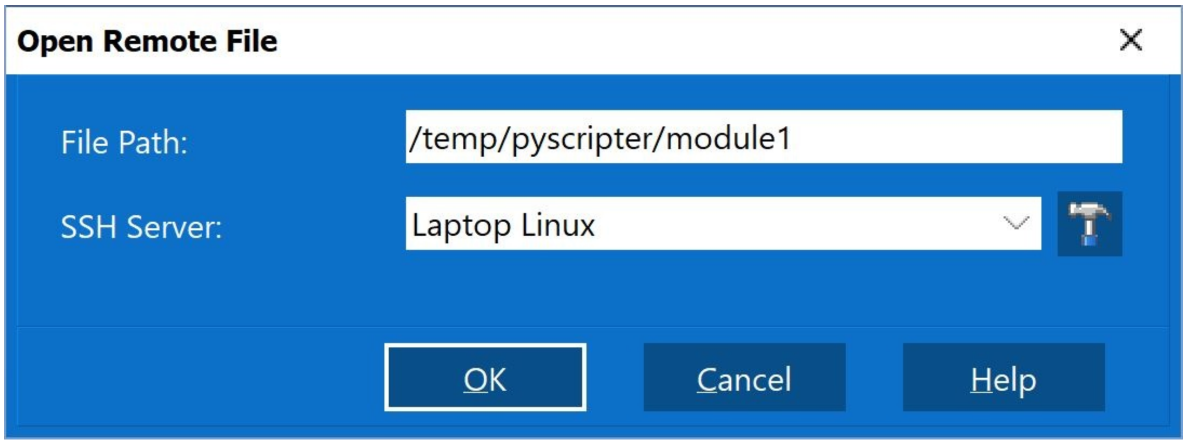
From a command prompt issue the following command:

```
ssh username@hostname
```

where username is the user name on the server side and host name is the IP address of the SSH server. If this works and you see the server shell, then PyScripter is ready to use the Server.

Opening remote files

You can open remote files using the File Menu. You are then shown the Open Remote File dialog shown below:

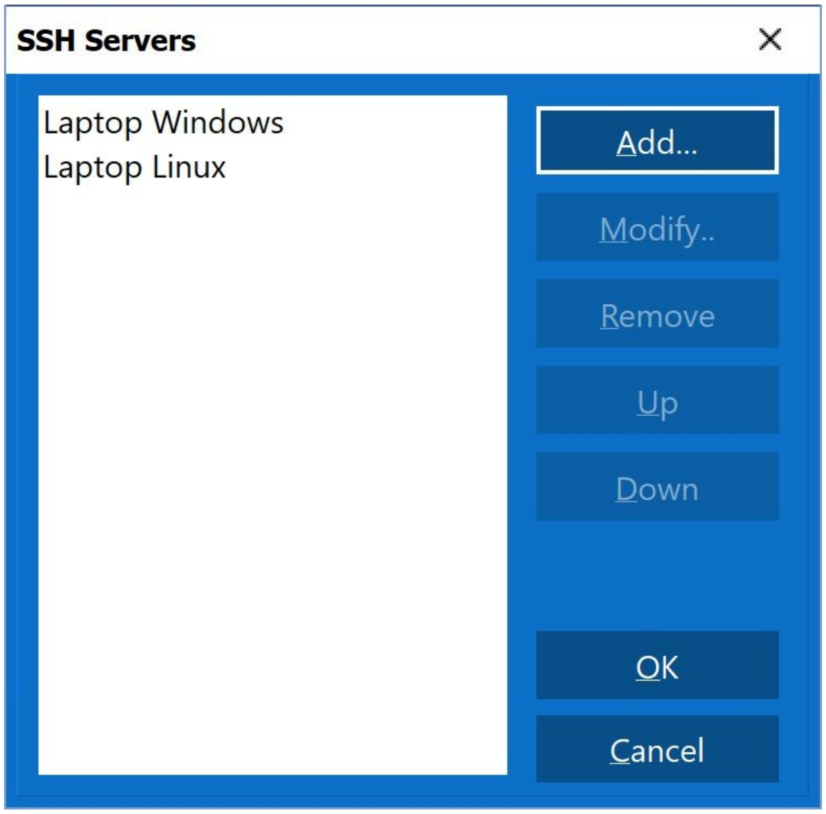


File Path: /temp/pyscripter/module1

SSH Server: Laptop Linux

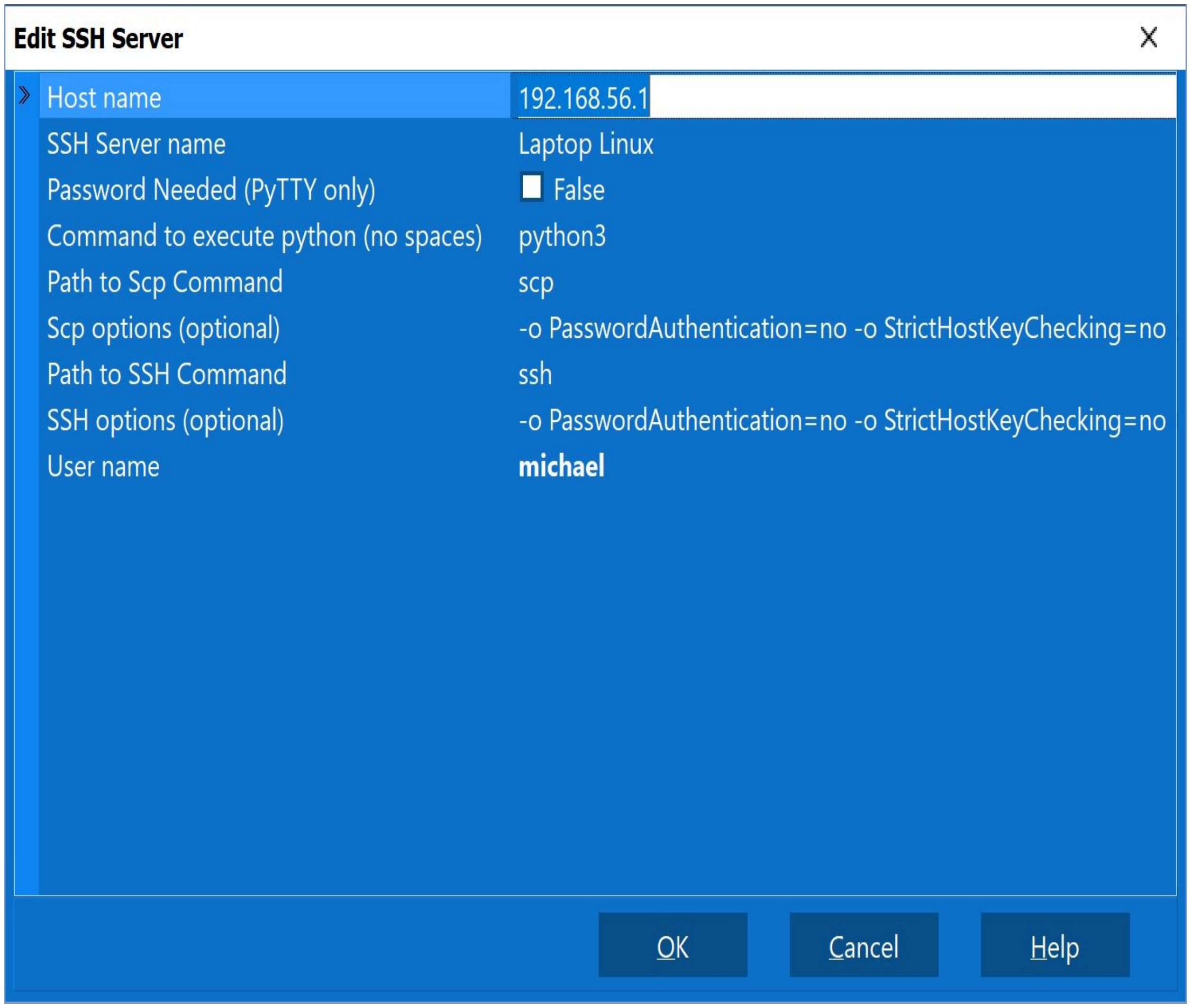
In this dialog box you provide the path to the remote file and select an SSH server from a drop-down list. You can also setup your SSH servers by pressing the button next to the SSH server field.   In PyScripter remote file names are shown in the UNC format \\server name\filepath.

Setting up SSH servers



In this dialog box you add remove or modify SSH servers.

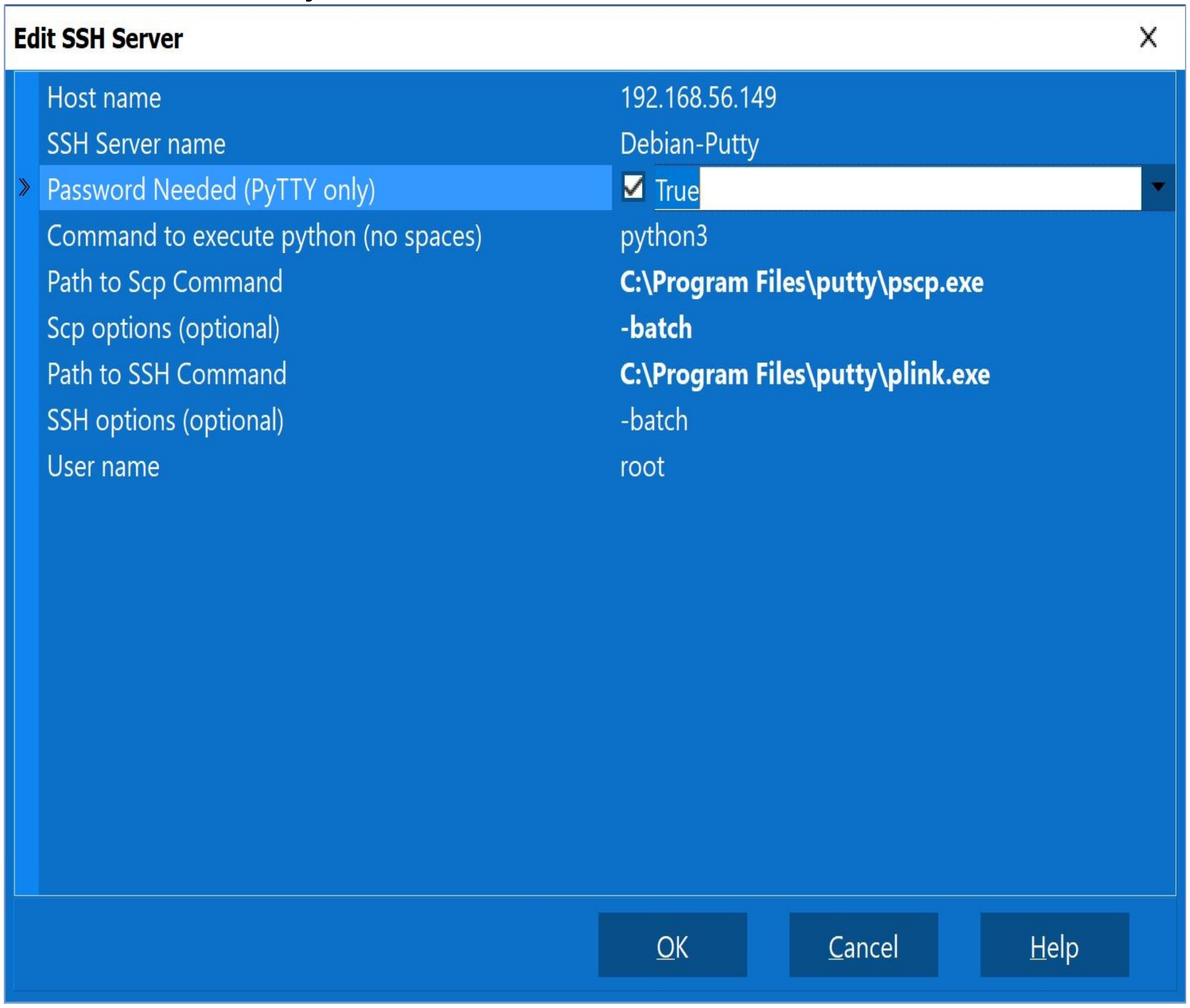
Editing SSH server information



Host name	192.168.56.1
SSH Server name	Laptop Linux
Password Needed (PyTTY only)	☐ False
Command to execute python (no spaces)	python3
Path to Scp Command	scp
Scp options (optional)	-o PasswordAuthentication=no -o StrictHostKeyChecking=no
Path to SSH Command	ssh
SSH options (optional)	-o PasswordAuthentication=no -o StrictHostKeyChecking=no
User name	**michael**

For each SSH server you need to provide a Name that will be used to identify the SSH server, as well as the user name and host name (or IP address) that will be used to connect to the server. You also need to provide the path to the scp and ssh commads and the command that will be used to execute Python on the server . Optionally you can provide additional SSH -o options that will be passed to the ssh and scp commands. For standard use leave this field empty. If you want to use password authentication (only with Putty - see below) you also need to check the Password Needed option.

Instead of OpenSSH for Windows you can use PuTTYas the SSH client. See below a typical PuTTY configuration of an SSH server. The example uses password authentication, but you can use a private/public key combination instead for password-free authentication.   In that case you need to use puttygen to create the private/public key and add -i path_to_your_private_key to the scp and ssh options or instead run pagent and add to it the private key. You also need to setup your server to accept the public key by appending it to the ~/.ssh/authorized_keys file. See here for details.



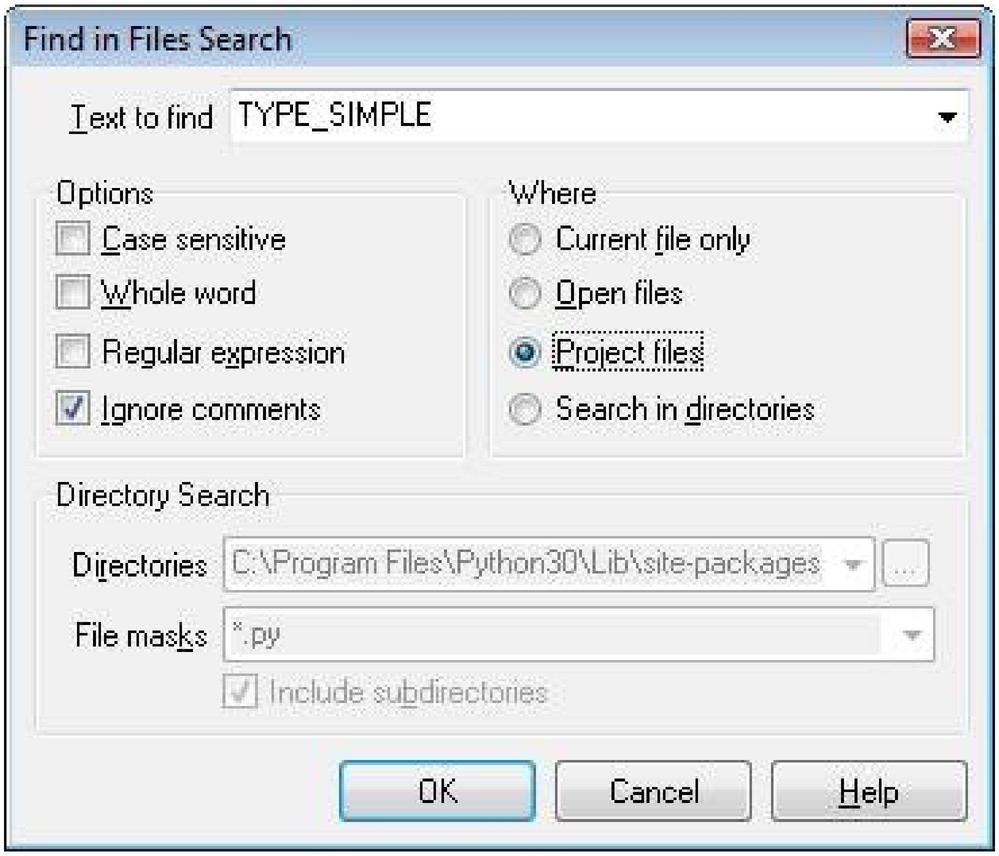
Edit SSH Server

Host name	192.168.56.149
SSH Server name	Debian-Putty
Password Needed (PyTTY only)	☑ True
Command to execute python (no spaces)	python3
Path to Scp Command	**C:\Program Files\putty\pscp.exe**
Scp options (optional)	**-batch**
Path to SSH Command	**C:\Program Files\putty\plink.exe**
SSH options (optional)	-batch
User name	root

Find in Files

Previous Top Next

PyScripter incorporates a powerful search facility that enables you to quickly locate text strings in files. Using "Find in Files", you can search the current file, all open files or all files in a directory (optionally including sub-directories).
To begin a search, select "Find in Files" from the PyScripter menu. A dialog will appear like the one below into which you can enter your search criteria. Note that the word under the cursor or the selected text when calling up this dialog, will be used as the default search string.



Text to find: TYPE_SIMPLE

Options: [ ] Case sensitive, [ ] Whole word, [ ] Regular expression, [✓] Ignore comments

Where: Current file only, Open files, [✓ Project files], Search in directories

Directories: C:\Program Files\Python30\Lib\site-packages

File masks: *.py

[✓] Include subdirectories

The various options in the search dialog are as follows:

Text to Find: The text or regular expression to search for

Options:

Case sensitive:	Search is case sensitive (a and A are treated as different characters)
Whole word:	Return matches that are whole words (delimited by whitespace or punctuation such as "().,<>-{}!@#$") Note that 0-9 and _ are treated as part of a word.
Regular expression:	The text to find is a regular expression
Ignore Comments:	Ignore matches in comments for Python files

Where (search scope):

Current file only:	Only the file that is currently in focus for editing
Open files:	All files that are currently open in the editor
Project files:	All project files
Search in directories:	All files specified by the Directory Search options (see below)

Directory Search:

Note: This portion of the dialog is only enabled if "Search in directories" is selected. Custom paremeters are supported in the directories field.

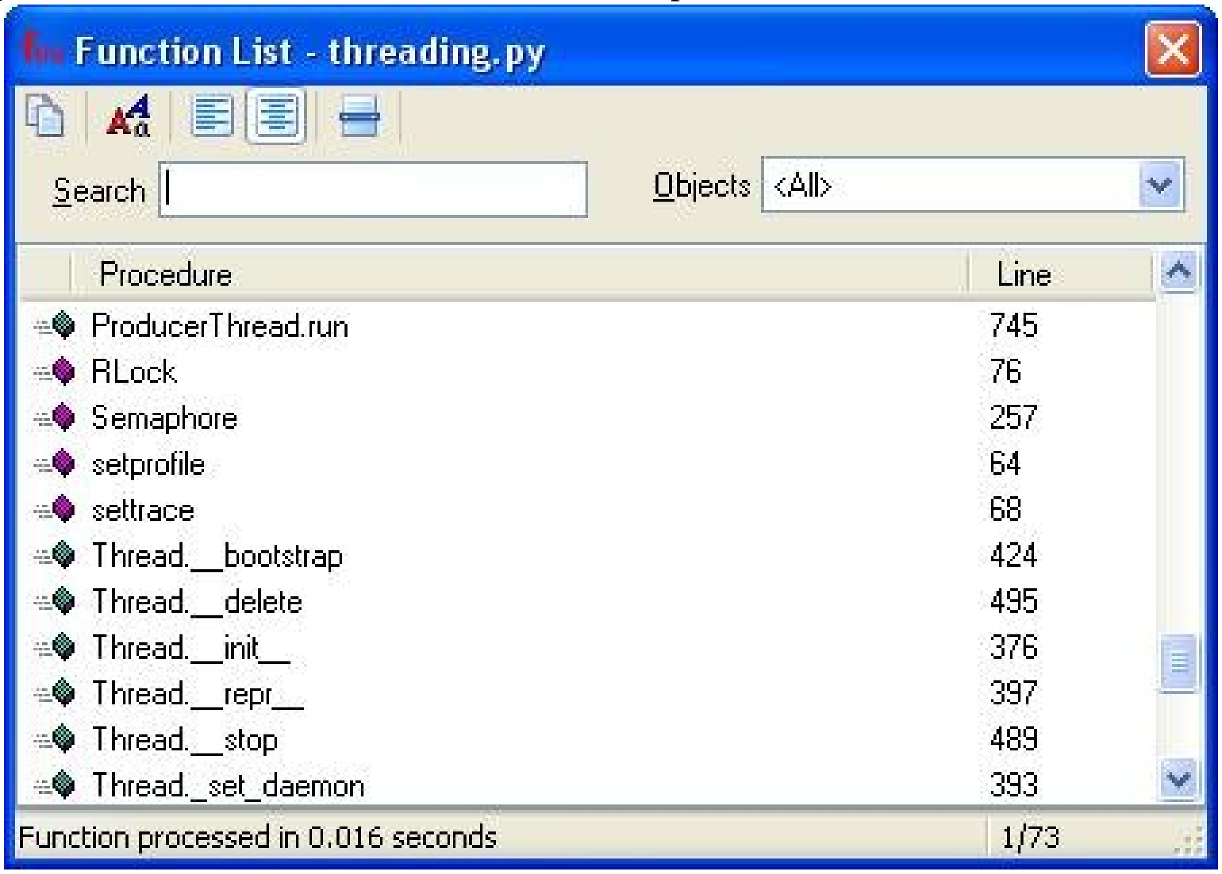
Directories:	A semicolon separated list of directories to search
"..." Button:	Allows browsing for a search directory
File masks:	Limits the search to a semicolon separated list of file extensions
Include subdirectories:	Enables recursive searching of the search directories

Once you have entered the search criteria, click the OK button or press enter to initiate the search. As the search progresses, results will be shown in the Find in Files Results window. From the results window, you can also perform multi-file search/replace on the matches.

Credits: This utility is based on code from the GExperts project (www.gexperts.org).

Find function

The Find function command pops up a dialog with a list of Python functions and methods defined in the current module and allows you to quickly jump to a given function. The function list window is pictured below:



Procedure	Line
ProducerThread.run	745
RLock	76
Semaphore	257
setprofile	64
settrace	68
Thread.__bootstrap	424
Thread.__delete	495
Thread.__init__	376
Thread.__repr__	397
Thread.__stop	489
Thread._set_daemon	393

Function processed in 0.016 seconds 1/73

To search for a function, start typing in the Search edit box. As you type, the characters will appear at the top next to the word Search. To jump to a selected function, double click or highlight it and press enter.

Two search modes are provided. In the first search mode (match only from the start), searches are conducted only on the beginning of the function name (after an optional class reference). In the second search mode (match anywhere), the search string can match at any point in the function name. For example, if you had two functions, MyClass.assign and MyClass.assignWidget, searching for "assign" would return both methods in either mode, whereas searching for widget would return neither when searching from the start, while MyClass.assignWidget would appear if searching for a match at any point.

Using the Objects combobox on the right, you can filter the function list to display all functions, only those without an associated class, or only those associated with a specific class. The Copy button will copy all function details to the clipboard.

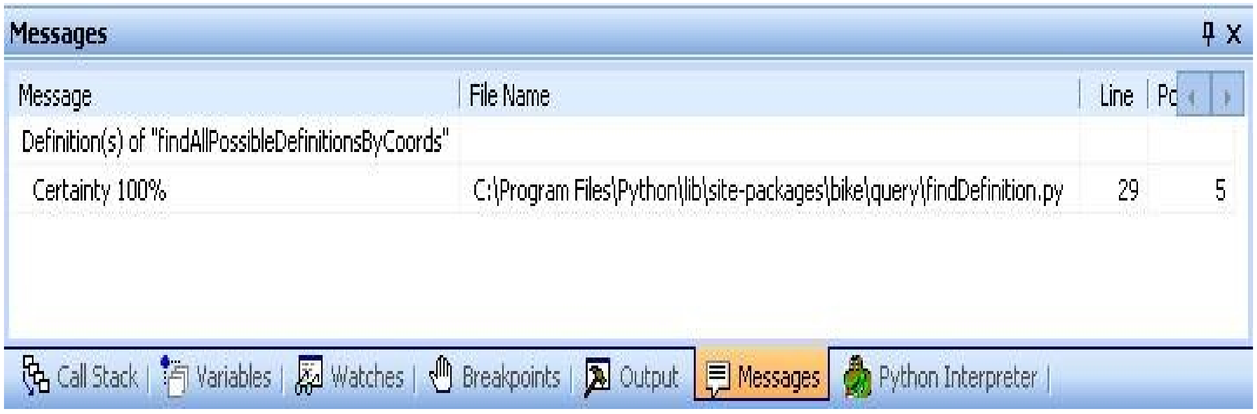
Credits: This utility is based on code from the GExperts project (www.gexperts.org).

Find Definition

This utility function allows you to find and jump to the definition of the identifier under th cursor, which may be in a different file. This feature can be invoked in two ways:

a) From the Search menu

In this case PyScripter provides detailed feedback on the found matches in the Messages Window (see below) and it reports any problems that may have occurred. If a definition is found the editor jumps to the that definition, which may be in a different file. In some cases the exact definition cannot be found with certainty and this is why the Messages Window also reports the degree of certainty for each candidate definition.



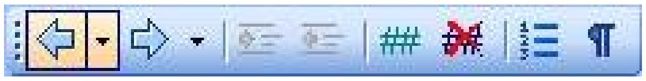
Message	File Name	Line	Pc
Definition(s) of "findAllPossibleDefinitionsByCoords"			
Certainty 100%	C:\Program Files\Python\lib\site-packages\bike\query\findDefinition.py	29	5

b) By clicking on an identifier while pressing the Ctrl key.

When you press the Ctrl key while the mouse hovers on a Python identifier, the identifier appears as a hyperlink. You can invoke the find definition function by clicking on that identifier. If the definition is found the cursor jumps to that definition, otherwise and unlike the case described above, no feedback is provided except for a beep sound.

In both case the original cursor position and the found definition is added to the Find Definition browsing history. You can move backwards and forward within the the browsing history by clicking on the Browse Back and Browse Forward buttons in the editor toolbar (the first two buttons in the toolbar shown below).

These two buttons also provide a drop-down list from which you can select to jump to a specific found definition.



Note: PyScripter provides its own powerful python source parsing engine which is used for the Find Definition and Find References operations.

Find References

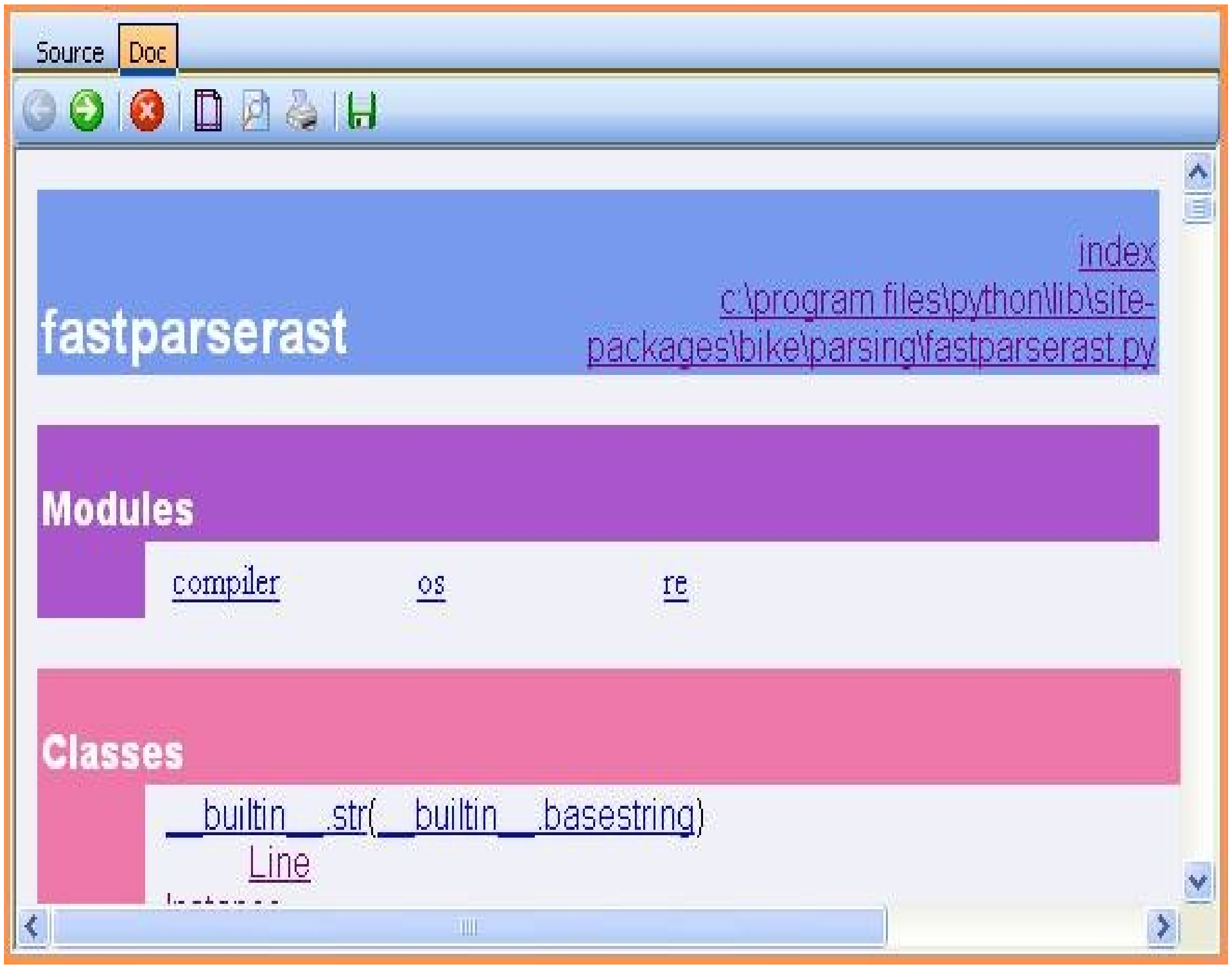
This utility function allows you to find all references of the identifier under the cursor, even if they are in different files. All files in the same directory as the file in the current editor are searched. In addition if the Python module belongs to a package all the files of that package are searched as well. This feature can be invoked from the Search menu. PyScripter provides detailed feedback on the found references in the Messages Window and it reports any problems that may have occurred.

Note: PyScripter provides its own powerful python source parsing engine which is used for the Find Definition and Find References operations.

Documentation View

Previous Top Next

PyScripter can use the standard Python module pydoc to generate HTML documentation for Python Modules. This feature is available from the "Source Code Views" submenu of the Tools menu. The HTML documentation is displayed using an internal browser. You can use the toolbar buttons to print preview, print or save the HTML file. To close the documentation view, right-click on the "Doc" tab and select "Close".



fastparserast

index
c:\program files\python\lib\site-packages\bike\parsing\fastparserast.py

Modules

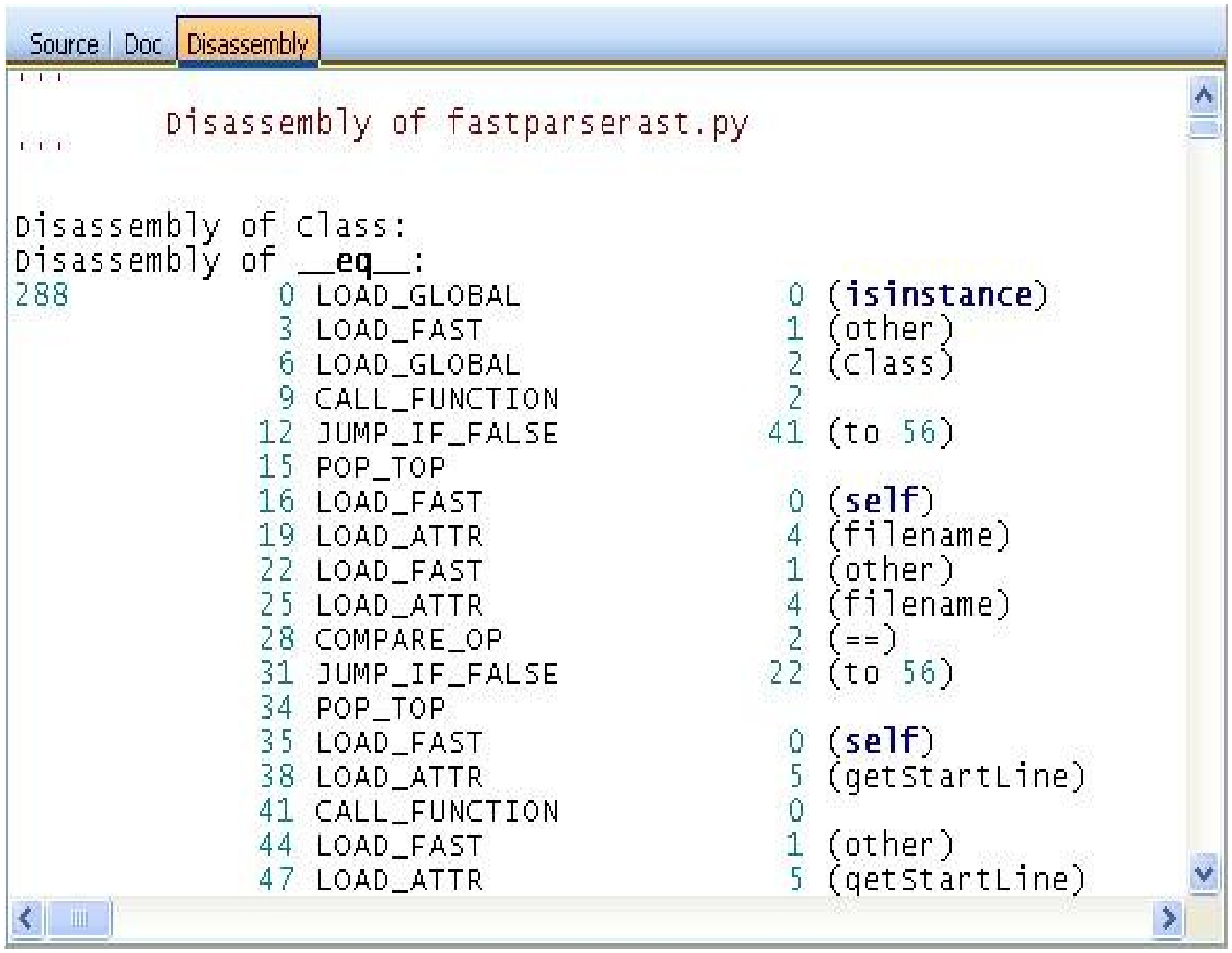
compiler   os   re

Classes

__builtin__.str(__builtin__.basestring)
    Line

Disassembly View

PyScripter can use the standard Python module dis to disassemble Python Modules. This feature is available from the "Source Code Views" submenu of the Tools menu. The disassembly is displayed in a separate editor. To close the disassembly view, right-click on the "Disassembly" tab and select "Close".



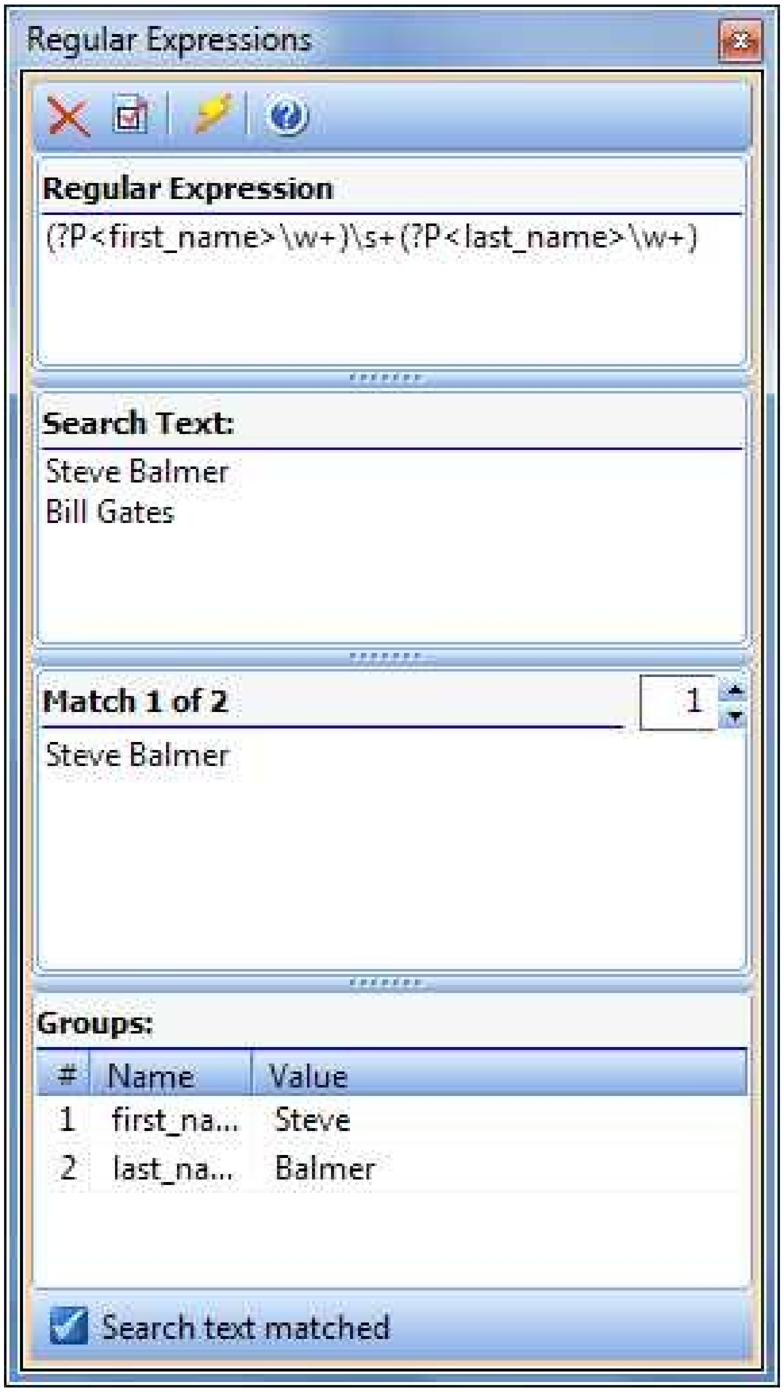
```
'''
       Disassembly of fastparserast.py
'''

Disassembly of Class:
Disassembly of __eq__:
288             0 LOAD_GLOBAL              0 (isinstance)
                3 LOAD_FAST                1 (other)
                6 LOAD_GLOBAL              2 (Class)
                9 CALL_FUNCTION            2
               12 JUMP_IF_FALSE           41 (to 56)
               15 POP_TOP
               16 LOAD_FAST                0 (self)
               19 LOAD_ATTR                4 (filename)
               22 LOAD_FAST                1 (other)
               25 LOAD_ATTR                4 (filename)
               28 COMPARE_OP               2 (==)
               31 JUMP_IF_FALSE           22 (to 56)
               34 POP_TOP
               35 LOAD_FAST                0 (self)
               38 LOAD_ATTR                5 (getStartLine)
               41 CALL_FUNCTION            0
               44 LOAD_FAST                1 (other)
               47 LOAD_ATTR                5 (getStartLine)
```

Regular Expression Testing

Previous Top Next

PyScripter provides for integrated regular expression testing. From the View menu select IDE Windows, Regular Expression Tester to show the following window.



```
(?P<first_name>\w+)\s+(?P<last_name>\w+)
```

Steve Balmer
Bill Gates

Steve Balmer

#	Name	Value
1	first_na...	Steve
2	last_na...	Balmer

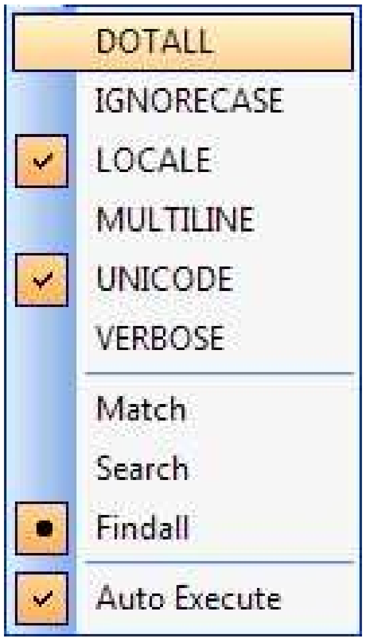
In this window you can type a regular expression and the search text and then press the Execute button to see the matched text and the value of each group of the regular expression. If you use the "findall" search type (see options below) then you can examine all matches found by using the spin edit control in the matches section.

Buttons on the Toolbar

Clear
Clears all the information entered.

Options
Allows to specify the various options of the re Python module such as IGNORECASE, VERBOSE etc. as well as whether you want to use the search, match or findall function of the regular expression objects. For more information look at the Python help file in the "re (standard module)" page.

The "Auto Execute" option determines whether the regular expression is executed every time the regular expression or the search text is changed.



Execute

Executes the search using the options specified and shows the results. There is no need to press if the "Auto Execute" option explained above is checked.

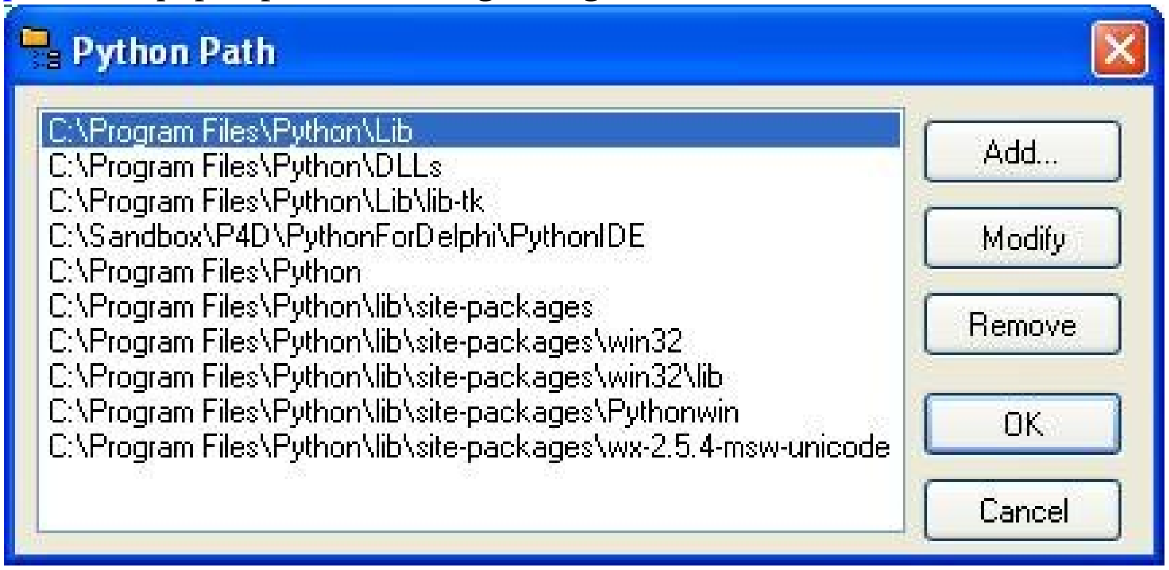
Help on re
Shows information about the re Python module and the syntax of regular expressions from the Python help file.

Note:
For more fully-featured regular expression testing you can use other more specialized programs such as Kodos or Kiki. You can easily integrate such programs with PyScripter as External Tools.

Python Path Configuration

Previous Top Next

This feature allows the configuration of the Python Path of the embedded Python interpreter. It is accessible from the Tools menu or the context menu of the File Explorer. It pops-up the following dialog:



```
C:\Program Files\Python\Lib
C:\Program Files\Python\DLLs
C:\Program Files\Python\Lib\lib-tk
C:\Sandbox\P4D\PythonForDelphi\PythonIDE
C:\Program Files\Python
C:\Program Files\Python\lib\site-packages
C:\Program Files\Python\lib\site-packages\win32
C:\Program Files\Python\lib\site-packages\win32\lib
C:\Program Files\Python\lib\site-packages\Pythonwin
C:\Program Files\Python\lib\site-packages\wx-2.5.4-msw-unicode
```

Pressing the Add.. button shows a directory selection dialog from which can select a directory to add to the path. The Modify button replaces the selected directory with one chosen using the directory selection dialog.

Tip: You can rearrange the order of the directories by dragging and dropping.

Note that the changes to the Python path apply only to the current session. To permanently change the Python path you can either modify the PYTHONPATH environment variable or modify the HKEY_LOCAL_MACHINE\SOFTWARE\Python\PythonCore\x.y\PythonPath registry setting. You can also use the PyScripter startup script python_init.py to modify the path.

Persistent Options

Previous Top Next

All user options including IDE options, IDE shortcuts, editor options, code templates and custom parameters are saved in a file called "PyScripter.ini". PyScripter searches for this file in the following locations

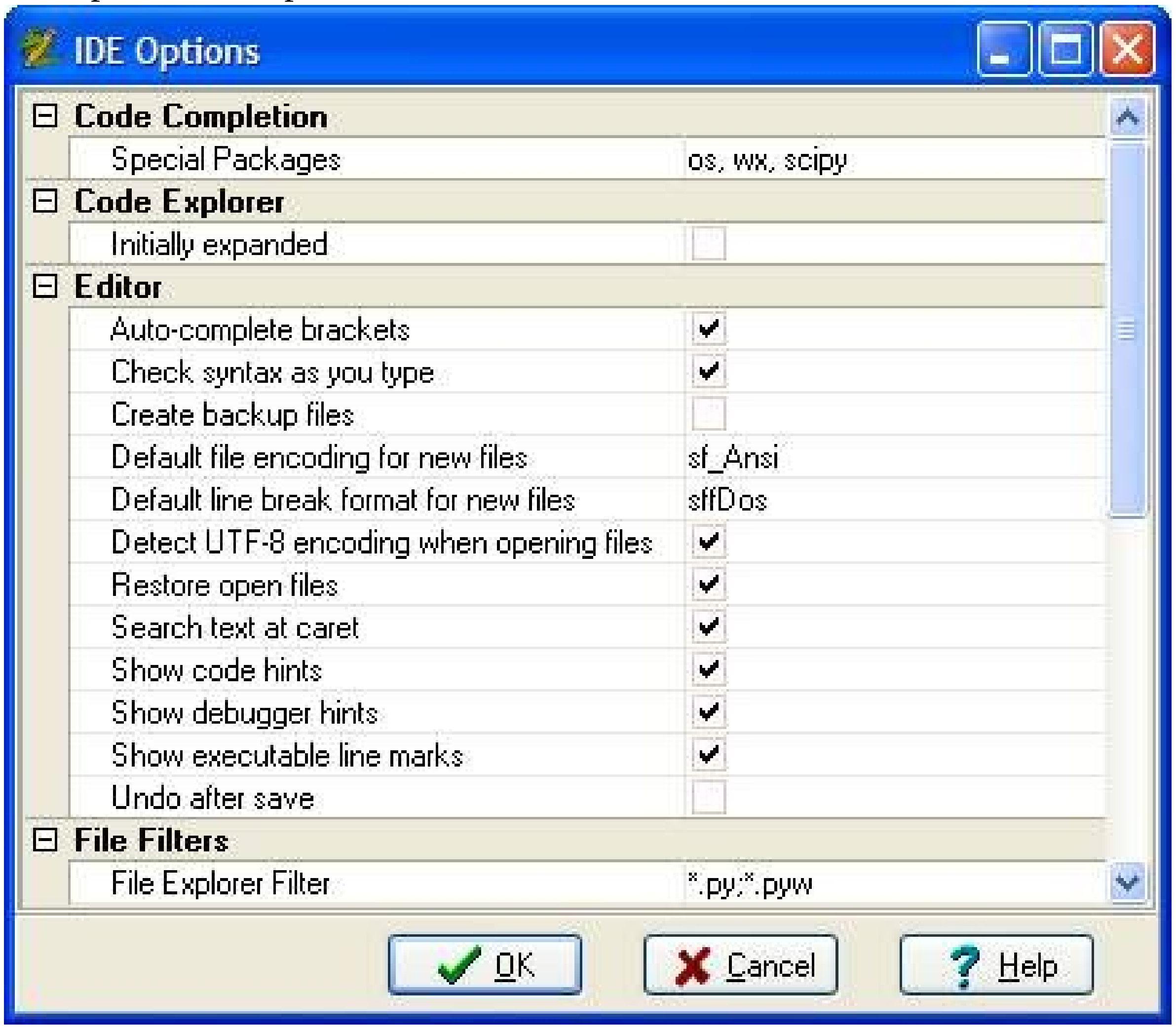
a) Exe directory
b) %APPDATA% (environment varaible)
c) %APPDATA%\Pyscripter

So the default location is the user's Application Data directory . If however the "PyScripter.ini" file exists in the same directory as the PyScripter executable then it is used in preference to the one in the user directory. This allows PyScripter installations in USB storage for example.
Application form sizes and positions (referred to as layouts) are saved in a separate settings file called "PyScripter.local.ini.

IDE Options

Previous Top Next

This dialog provides a number of options that affect the operation of PyScripter. These options are explained below.



Code Completion	
Special Packages	os, wx, scipy
Code Explorer	
Initially expanded	☐
Editor	
Auto-complete brackets	✓
Check syntax as you type	✓
Create backup files	☐
Default file encoding for new files	sf_Ansi
Default line break format for new files	sffDos
Detect UTF-8 encoding when opening files	✓
Restore open files	✓
Search text at caret	✓
Show code hints	✓
Show debugger hints	✓
Show executable line marks	✓
Undo after save	☐
File Filters	
File Explorer Filter	*.py;*.pyw

Code Completion

Auto-Completion font

Allows you to customize the size and type of font used in auto-completion.

Case Sensitive

This option determines whether the filtering of the code completion list when you type characters is case sensitive (default True).

Code completion list size

The size of the code completion window in number or lines.

Complete as you type

Code completion is invoked automatically as you type.
Complete Python keywords
Python keywords appear in the completion list when appropriate.
Complete with word-break characters
When the completion list is displayed completion with the currently selected entry occurs when word-break characters are typed (e.g. space, brackets etc.). This is in addition to completing with the Tab and Enter keys.
Auto-complete with one entry
If true, when the completion list contains one entry complete automatically without showing the list.
Editor code completion
Enable/Disable code completion in the editor.
Interpreter code completion
Enable/Disable code completion in the interpreter.

Special packages
The code and parameter completion should be one of the best you can find in any Python IDE. However,if you find that code and parameter completion is not very accurate for certain modules and packages such as wxPython and scipy you can achieve near perfect completion if you add these packages to this option (comma separated list). By default it is set to "os, wx, scipy". Special packages are imported on demand to the interpreter instead of scanning their source code.
Code Explorer
Initially expanded
If checked the Code Explorer stats with its nodes initially expanded.
Editor Options
Auto-complete brackets
If checked, when you edit Python scripts and you type an open bracket ("(", "[", "{") the corresponding closing bracket is entered automatically. When editing HTML and XML files the opening bracket is "<". It also auto-completes strings.
Auto-reload changed files
If checked, files changed on disk will be reloaded without prompting if the files have not been changed inside PyScirpter. A message is shown in the status bar and a beep sound can be heard when this happens.
Check syntax as you type
If checked, when you edit Python files Pyscripter continuously check the active file for syntax error, which are shown in a similar way in which word processors

show spelling errors. If you place the cursor on an error indicator you will see a hint explaining the problem.
Create backup files
If checked PyScripter will create backup files before overwriting existing files.
Default line breaks for new files
Controls the line break format for new files. Options: sffDos, sffUnix, sffMac, sffUnicode. The last option although available is not currently supported.
Display packages names
Display package names instead of file names for package files (__init__.py) in editor tabs.
Default file encoding for new files
Controls the encoding for new files. Options: sf_Ansi, sf_UTF8, sf_UTF8_NoBOM. See the topic on Encoded Source Files for further information.
Detect UTF-8 encoding when opening files
This option controls whether PyScripter attempts to detect utf-8 encoding when opening files without the BOM mark. This detection is done by analyzing the first 4000 characters of the file and is imperfect. It only applies to non-Python files since utf-8 encoded Python files are required to have either the BOM mark or an encoding comment.
File line limit for syntax check as you type
Files with more lines than specified will not be syntax checked as you type. (default 1000 lines)
Highlight selected word
If checked, when you select a word by double-clicking or by issuing the editor command "Select Word", all occurrences of the selected word in the editor are highlighted.
Restore open files
If checked PyScripter will restore the files which were open when the last editing session ended when it starts up.
Search text at caret
If checked, when the search function is invoked the search expression is set to the word containing the editor caret.
Show code hints
If unchecked code hints are not shown.
Show debugger hints
If unchecked debugger hints are not shown.

Show executable line marks
If checked the editor will show in the gutter special marks for executable lines.
Trim trailing spaces when saving files
If checked the editor will trim spaces at the end of the lines when saving files. This works independently of the corresponding editor option which trims trailing spaces while editing.
Undo after save
If checked, you can undo editing actions beyond the point at which you saved the file. Otherwise undo can only take you back to point at which you last saved the file.

File Explorer

File Explorer background processing
If checked (default) the File Explorer processes folder expand makrs and file impages in threads. Uncheck if you encounter errors related to File Explorer.

File Filters

File Explorer Filter
The file extensions separated by semi-colon that the File Explorer recognizes as Python files. Used when the filter function is applied.
e.g. *.py;*.pyw
Open Dialog Python Filter
The Python file filter for the File Open dialog. Multiple filters should be separated by vertical bars ("|").
e.g. Python Files (*.py;*.pyw)|*.py;*.pyw
Open Dialog CSS Filter
The CSS file filter for the File Open dialog.
Open Dialog CPP Filter
The C/C++ file filter for the File Open dialog.
Open Dialog HTML Filter
The HTML file filter for the File Open dialog.
Open Dialog JavaScript Filter
The JavaScript file filter for the File Open dialog.
Open Dialog PHP Filter
The PHP file filter for the File Open dialog.
Open Dialog XML Filter
The XML file filter for the File Open dialog.
Open Dialog YAML Filter
The YAML file filter for the File Open dialog.

IDE

Check for updates automatically

If checked you will be notified when new version of the PyScripter become available for download.

Days between update checks

You can define how often you want to check for PyScripter updates.

Dock animation interval

This and the following option, control the animation when you show a hidden docked form. The Dock animation interval controls the interval in milliseconds, between successive redraws of the shown window. It needs to be greater than or equal to 1.

Dock animation move width

The Dock animation move width controls the width in pixels, by which the animated form is moved. To disable the animation set this parameter to a large value, e.g. 1000.

File Change Notification

Controls whether the user is notified about changes in the edited files (possible values: fcnFull, fcnNoMappedDrives, fcnDisabled). Default: fcnNoMappedDrives.

Editor tab position

Controls whether the editor tabs should appear at the top or at the bottom (possible values: toBottom, toTop).

File template for new Python scripts

The "New Python Module" command checks the value of this option and, if it is not empty and is the name of an existing File Template, it uses the template for the new module. By default points to the provided File Template for Python scripts.

Mask FPU exceptions

Floating point operations that result in special numbers such as Nan, + or - Infinity etc, normally raise exceptions. However some packages such as Scipy would raise such exceptions when they get imported into Python and they wouldn't be usable. Keep this option checked if you want to use such packages with PyScripter.

Show tab close button

Shows/hides the close buttons on the editor tabs.

Smart Next/Previous Page

(Shift+)Ctrl+Tab can be used to move to the (previous) next page. If this option is checked these keys move through the pages according to the Z-order of the pages (as the Alt+Tab does in Windows). Otherwise they move according to the visual order of the pages.

Style Main Window Border

If true the main application window border is styled to match the selected style. If false it has the standard windows border.

Use Python colors in IDE

If checked the colors of the python highlighter (Background color from the space attribute and foreground color of the identifier attribute) are used in the IDE windows. When a dark highlighter is used this option helps achieve a uniform "dark" look. (default false)

Project Explorer

Initially expanded

If checked the Project Explorer stats with its nodes initially expanded.

Python Interpreter

Always use sockets

Pyscripter can use sockets or Windows named pipes for communicating with server. Named pipes can only be used in pywin32 is installed in the active python version. The main advantage of named pipes is that it avoids firewall issues related to socket connections. If this option is checked sockets are used in preference to named pipes. (default true)

Clear output before run

If checked the interpreter output is cleared before running scripts. (default false)

Interpreter history size

Specifies the size of the interpreter command line history (default 50)

Jump to error on Exception

If set, when an exception occurs the file in which the exception occurred opens and the error position is displayed. (default true)

Post mortem on exception

If this option is checked when an unhandled exception occurs while running or debugging a script, PyScripter enters the post-mortem analysis mode.

Pretty print output

If checked the standard python module pprint is used to format interpreter output.

Python Engine Type

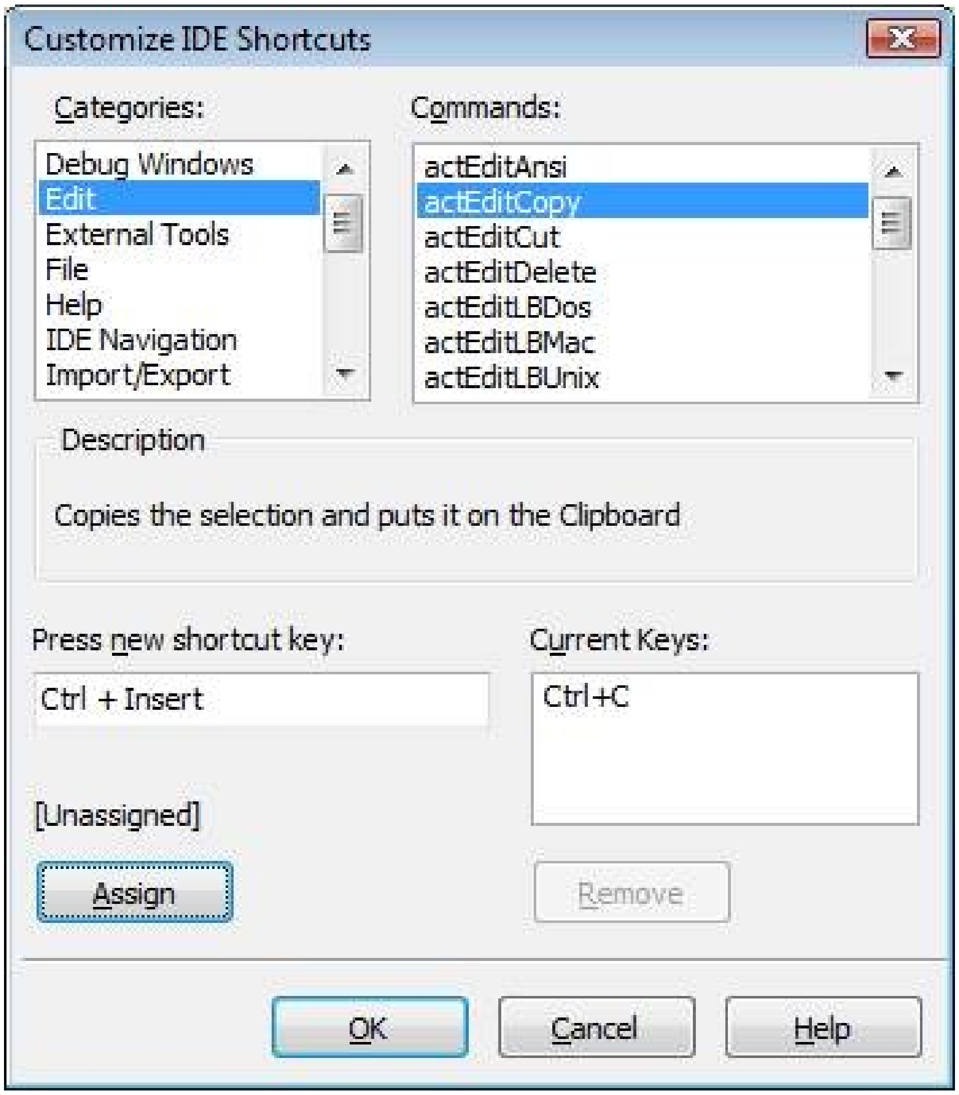
Controls which engine type is used. Possible values (peInternal, peRemote) See (Remote Python Engines for details).

Reinitialize before run
If set and a remote Python engine is used, it is re-initialized before running or debugging a script. This is necessary when using GUI applications (Tkinter, wxPython, etc.). It is set by default.
Save environment before run
It saves environment options including open files. layout etc. before running scripts so that you can recover if the IDE crashes.
Save files before run
If checked all open files are saved before running scripts
Save interpreter history
If checked the interpreter history is saved at program exit and restored when PyScripter is started again.
Step into open files only
If checked the debugger will not step into files that are not currently open. Defaults to false.
Timeout for running scripts
Time in ms for running scripts timeout. If different than zero, when the elapsed time since the start of running a Python script exceeds the timeout value, you are given the opportunity to interrupt the script.   Due to Python limitations this does not work when the script loops indefinitely inside a function.
UTF-8 in Interactive Interpreter
If checked the input to the interactive interpreter is converted to UTF-8 before is fed to python. The comments related UTF-8 encoded source files apply to the interactive interpreter as well.
Shell Integration
File Explorer Context Menu
If checked, it adds a menu item "Edit with Pyscripter" to the Windows File Explorer context menu for Python files. If unchecked this option removes such context menu if it is present.
SSH
Disable Variables Window with SSH
The updating of the Variables window requires data transers between PyScripter and remote Python Engines. With slow SSH connections this could slow down PyScipter. If checked, the Variables Window is desabled while using an SSH engine.
Scp command
This option allows you to specify a path to an scp command. This could be

useful if you have multiple versions of scp installed and want to specify which one to use. The defalut value 'scp' assumes scp can be found on the system path. You can overwrite this value in the SSH server configuration.
Scp options
This option allows you to specify options to used with the scp command. You can overwrite this value in the SSH server configuration.
SSH command
This option allows you to specify a path to an ssh command. This could be useful if you have multiple versions of ssh installed and want to specify which one to use. The defalut value 'ssh' assumes ssh can be found on the system path. You can overwrite this value in the SSH server configuration.
SSH options
This option allows you to specify options to used with the ssh command. You can overwrite this value in the SSH server configuration.

IDE Shortcuts

You can customize the IDE keyboard shortcuts using the "IDE Shortcuts..." command of the Options submenu of the Tools menu. In the dialog box shown below you can add or remove shortcuts for the the various commands. Note that you can have multiple shortcuts for each command.



Copies the selection and puts it on the Clipboard

Press new shortcut key: Ctrl + Insert

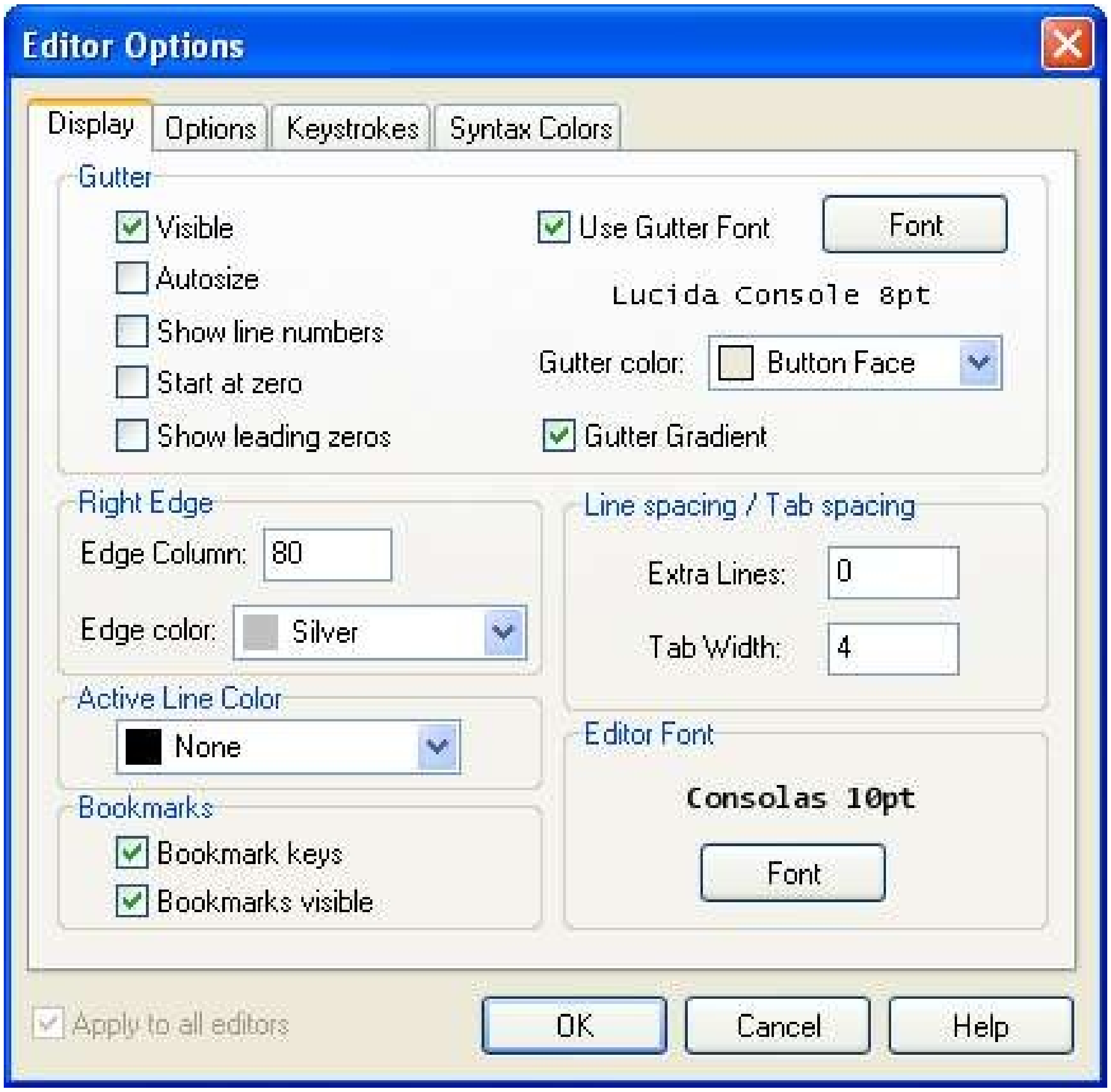
[Unassigned]

Current Keys: Ctrl+C

Editor Options

A number of editor options are user-configurable. To customize the editor options select "Editor Options..." from the Tools|Options menu which displays the Editor Options dialog. This dialog contains four tabs and the options in each tab are explained below. You can apply the changes to either the Active editor if any or else to all editors, by checking the relevant checkbox at the bottom of the dialog. Changes applying to all editor windows affect also the key bindings and the syntax highlighter colors interactive interpreter, but not the other options. Use the context menu of the interactive interpreter to changes the other editor options of the interpreter window.

Display tab



☑ Visible
☐ Autosize
☐ Show line numbers
☐ Start at zero
☐ Show leading zeros
☑ Use Gutter Font
Lucida Console 8pt
Gutter color: Button Face
☑ Gutter Gradient
Edge Column: 80
Edge color: Silver
Extra Lines: 0
Tab Width: 4
None
Consolas 10pt
☑ Bookmark keys
☑ Bookmarks visible
☑ Apply to all editors

Gutter (The Margin at the left hand side from the Editor)

Visible:

If not checked the right margin with the line numbers will not be visible.

.

Autosize:

Autosize the width to the line numbers.

Gutter Gradient:

If checked a gutter gradient is painted according to the selected theme.

Gutter color:

The colour used when the gutter gradient is disabled.

Show line numbers:

Self explanatory.

Show leading zeros:

If checked it shows leading zeros in line numbers.

Zero start:

The first line number will be zero.

Gutter Font:

Changes the Gutter font

Right Edge (a gray line showing the right margin):

Edge column:

Enter the count of characters where the right edge should appear. Use zero to hide this line.

Edge color:

Changes color of the right edge.

Active Line Color:

Select the color for highlighting the active editor line or None if you do not want to highlight it.

Bookmarks:

Bookmark keys:
Enable the bookmark shortcuts (see the Keystrokes tab).

Bookmarks visible:
Show bookmarks in the editor gutter.

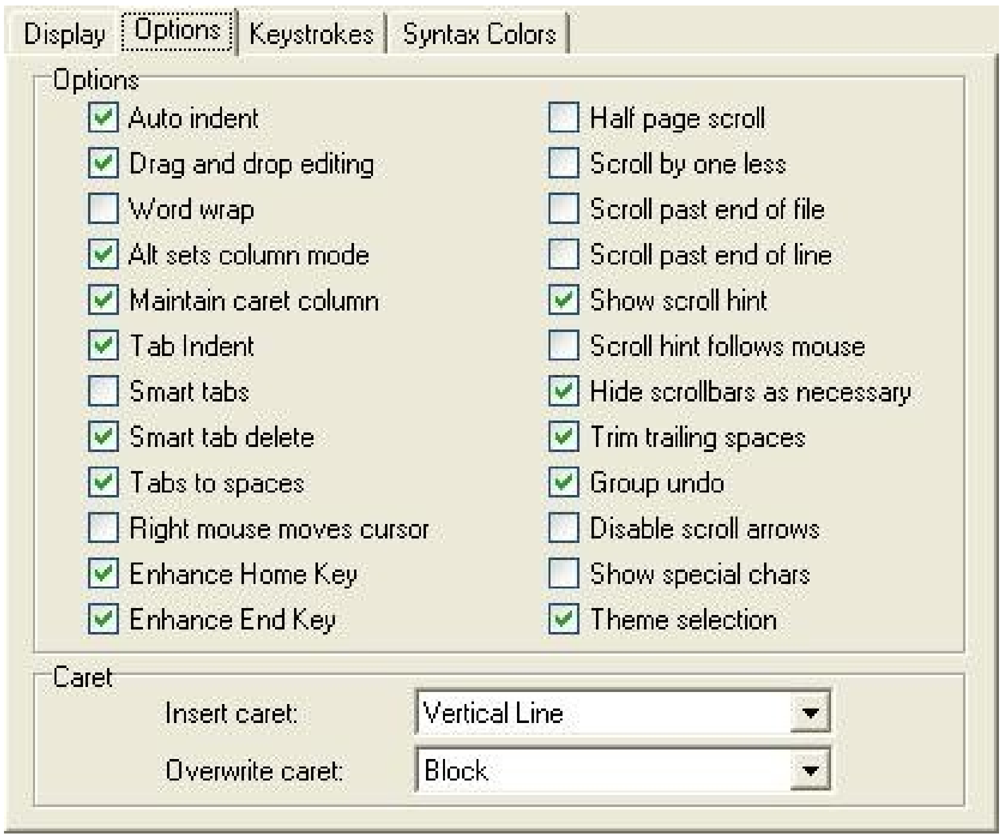
Line Spacing/Tab spacing:

Extra lines:
Extra spacing between the lines.

Tab width:
The width of tabs in characters.

Editor Font:
Changes the editor font.

Options tab
Various other editor options.



Options

[✓ Auto indent]
[✓ Drag and drop editing]
Word wrap
[✓ Alt sets column mode]
[✓ Maintain caret column]
[✓ Tab Indent]
Smart tabs
[✓ Smart tab delete]
[✓ Tabs to spaces]
Right mouse moves cursor
[✓ Enhance Home Key]
[✓ Enhance End Key]
Half page scroll
Scroll by one less
Scroll past end of file
Scroll past end of line
[✓ Show scroll hint]
Scroll hint follows mouse
[✓ Hide scrollbars as necessary]
[✓ Trim trailing spaces]
[✓ Group undo]
Disable scroll arrows
Show special chars
[✓ Theme selection]

Caret

Insert caret: Vertical Line
Overwrite caret: Block

ALT sets column mode:

If activated then the Alt key sets the editor in column selection mode. i.e. you'll be able to select columns from a text.

Auto indent:

Enables automatic indentation whereby a new line keeps the indent of the previous line.

Disable scroll arrows:

Disables the scroll bar arrow buttons when you can't scroll in that direction any more.

Drag and Drop editing:

If activated you will be able to drag text and drop it to another position.

Enhanced Home key:

If activated and you use the Home key one time, the caret is placed at the first occurrence of a non white-space, the second time it's placed in column 1.

Enhanced End key:

End key behaves in a way analogous to the Enhanced Home.

Group Undo:

Concequtive editing actions are undone/redone together

Half page scroll:

When scrolling with PageDown and PageUp keys, scrol of a half screen instead of a full.

Hide scrollbars as necessary:

If enabled, the scrollbars will only show when necessary. If you have selected "Scroll Past end of line", then it the horizontal bar will always be there.

Insert/overwrite caret:

Select the caret to be displayed in insert and overwrite editor modes.

Maintain caret column:

When moving through lines w/o the option "Scroll past end of line", keeps the column position of the cursor.

Right mouse moves cursor:

When clicking with the right mouse for a popup menu, move the cursor to that location.

Scroll by one less:

Scrolls page by one line less than the page length.

Scroll hint follows mouse:
The scroll hint follows the mouse when scrolling vertically.

Scroll past end of file:
Cursor can go after the theoretical end of file.

Scroll past end of line:
You'll be able to place the caret beyond the text (width).

Show scroll hint:
Shows a hint window with the numbers of the displayed lines when you scroll the editor.

Show special chars:
Show special chars such as TABS and new lines.

Smart tab delete:
Similar to Smart tabs, but applies to character deletion with the BackSpace key.

Smart tabs:
When tabbing, the cursor will go to the next non-white space character of the previous line.

Tab Indent
When some text is selected the Tab key indents and the Shift-Tab key unindents the selection.

Tabs to Space:
Replaces tabs with spaces while you're typing.

Theme selection:

If selected a theme specific color is used for the background of selected text.. Otherwise the Windows Highlight color is used. Note that if the foreground color of the editor is not black or the background color is not white this option is ignored.

Trim trailing spaces:

Removes all spaces at the end of the lines.

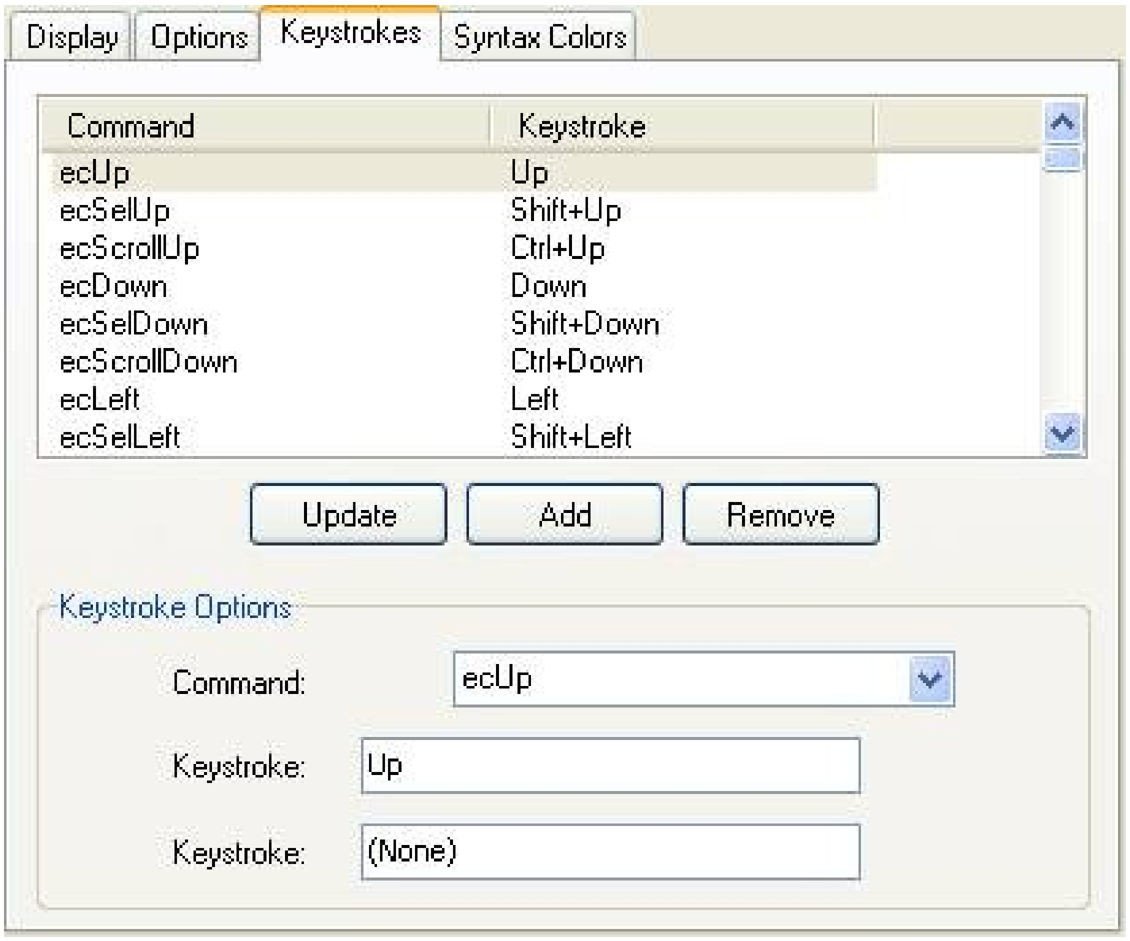
Word wrap:

Enable/disable word wrap.

Keystrokes tab

This tab allows you to customize the editor keyboard shortcuts.

Note that the second keystroke is for defining a sequence of keystrokes as a shortcut (e.g. Ctrl+K, I)

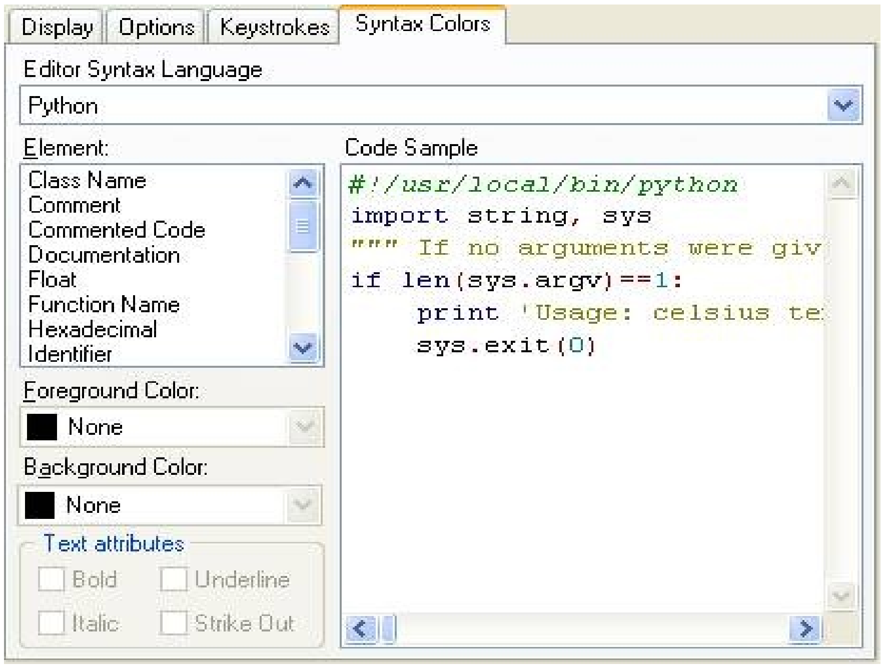


Command	Keystroke
ecUp	Up
ecSelUp	Shift+Up
ecScrollUp	Ctrl+Up
ecDown	Down
ecSelDown	Shift+Down
ecScrollDown	Ctrl+Down
ecLeft	Left
ecSelLeft	Shift+Left

Command: ecUp
Keystroke: Up
Keystroke: (None)

Syntax Colors tab

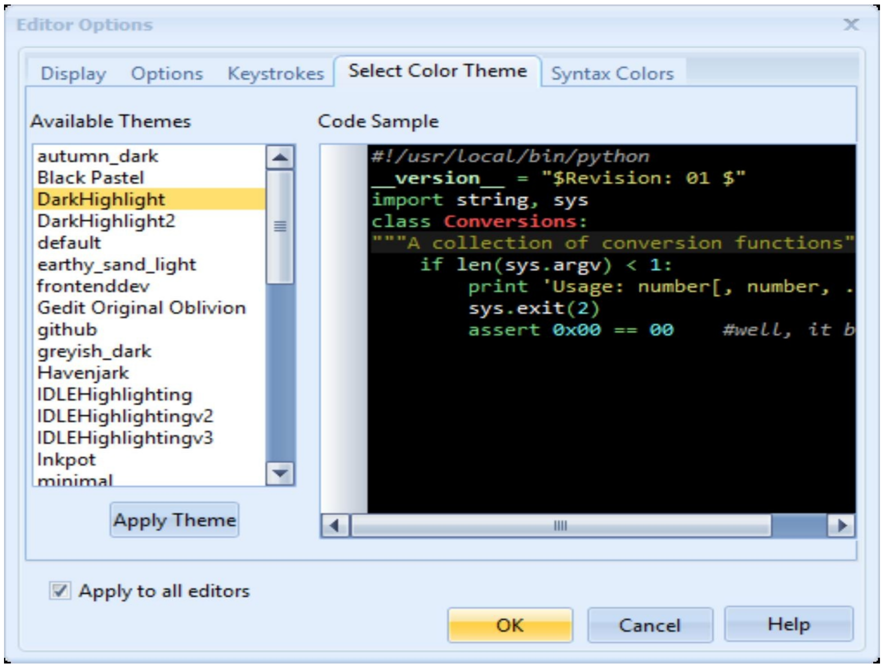
The options in this tab allow you to customize the syntax highlighting for Python and other file types.



```
#!/usr/local/bin/python
import string, sys
""" If no arguments were giv
if len(sys.argv)==1:
    print 'Usage: celsius te
    sys.exit(0)
```

Color theme tab

In this tab you can preview and apply available syntax highlighting schemes themes. These themes are located in the %APPDATA%\PyScripter\Highlighters directory.



```
#!/usr/local/bin/python
__version__ = "$Revision: 01 $"
import string, sys
class Conversions:
"""A collection of conversion functions"
    if len(sys.argv) < 1:
        print 'Usage: number[, number, .
        sys.exit(2)
        assert 0x00 == 00     #well, it b
```

Import/Export Settings

Previous Top Next

Using the commands under Tools, Options, Import Export, you can import and export shortcuts and syntax highlighting settings. This allows PyScripter users to share shortcuts and syntax highlighting settings. If you have customized these settings to match those of popular IDEs and editors such as Visual Studio and IDLE, you are encouraged to submit them to pyscripter@google.com, so that they will be made available to all users in a future version of PyScripter.

Command-line Options

When invoked without any arguments PyScripter will load the latest version of Python and restore the files which were open when the last editing session ended. If no files were open an empty Python module is created.   This behavior can be changed with command-line arguments:

PyScripter [--pythonversion] filename1 filename2 ...

where

pythonversion can be PYTHON23, PYTHON24 etc.

if pythonversion is provided on the command-line PyScripter tries to use that version if it is available and PyScripter has been compiled for a version lower to or equal to that version.
If one or more filenames are provided on the command-line they are opened when PyScripter starts.
To open a file at a specific line and column use "filename (lll:ccc)", i.e. the filename should be followed by a space and enclosed in parentheses the line and column numbers separated by colon. The whole expression should be enclosed in double quotes. The expression in double quotes should match the regular expression (.+) \((\d+):(\d+)\)$
Other command line flags:
--PROJECT filename

Open a specific PyScripter project file

--*PYTHONDLLPATH*

In order to allow PyScripter to work with unregistered version of Python such as Portable Python, another command line argument is provided PYTHONDLLPATH. When such an argument is provided the registry search is bypassed and the Python DLL found in that path is used instead.
e.g. PyScripter --PYTHON25 --PYTHONPATHDLL "E:\PortablePython"
See Using PyScripter with Unregistered Python for more information

--NEWINSTANCE or -N

If set a new instance of Pyscripter is started. This to prevent the default behavior which is to activate an existing instance of PyScripter is one is running.

NOTE

Since version 1.85 PyScripter options are marked with two dashes "--".

Further information on --PYTHONDLLPATH

There are two types of Python installation

a) For all users

    Python creates registry entries at HKEY_LOCAL_MACHINE\SOFTWARE\Python\PythonCore\2.x with installation info and puts the dll in c:\Windows\System32. (*This is no longer the case since python version 3.5*)

b) For a single user

    Python creates registry entries at HKEY_CURRENT_USER\SOFTWARE\Python\PythonCore\2.x with installation info and does not put the dll in c:\Windows\System32.

PyScripter without any command line flags looks at the registry to find the latest version of Python and then for an all user installation tries to load the relevant Python dll from the system path. For a single user installation tries to load the DLL from the Install path that is in the registry.

When PyScripter is used with a --PYTHONxx flag then it does the above but searching only for the specific version. The Registry lookup does not take place when Python is used with the --PYTHONDLLPATH. Instead PyScripter tries to load the Python dll from the specified path.

The --PYTHONDLLPATH flag should be used with the --PYTHONxx flag. See help topic Using PyScripter with Unregistered Python for an example of using PyScripter with portable Python.

The %PYTHONHOME% environment variable is not used by PyScripter directly but by Python to find the installed libraries. See the Python documentation for its use.

Startup Python Scripts

Previous Top Next

PyScripter users can create two startup python files that can be run at PyScripter startup.

1. pyscripter_init.py
   This script is run once in the space of the internal Python engine at start-up. It can be used to set various PyScripter options and to customize the PyScripter user environment.

2. python_init.py
   This file is executed when a Python engine, internal or remote, is initialized and every time the engine is reinitialized. It can be used to customize the Python engine, for example by always importing certain units.

PyScripter searches for startup python files first at the PyScripter.exe directory, and if it is not found there at %APPDATA%\Pyscripter, where %APPDATA% is the environment variable. The PyScripter installation program places these files without any content at %APPDATA%\Pyscripter, if the files are not there already.

You can edit the startup files by using the Tools Menu command "Edit Startup Files".

,

Using PyScripter with Unregistered Python

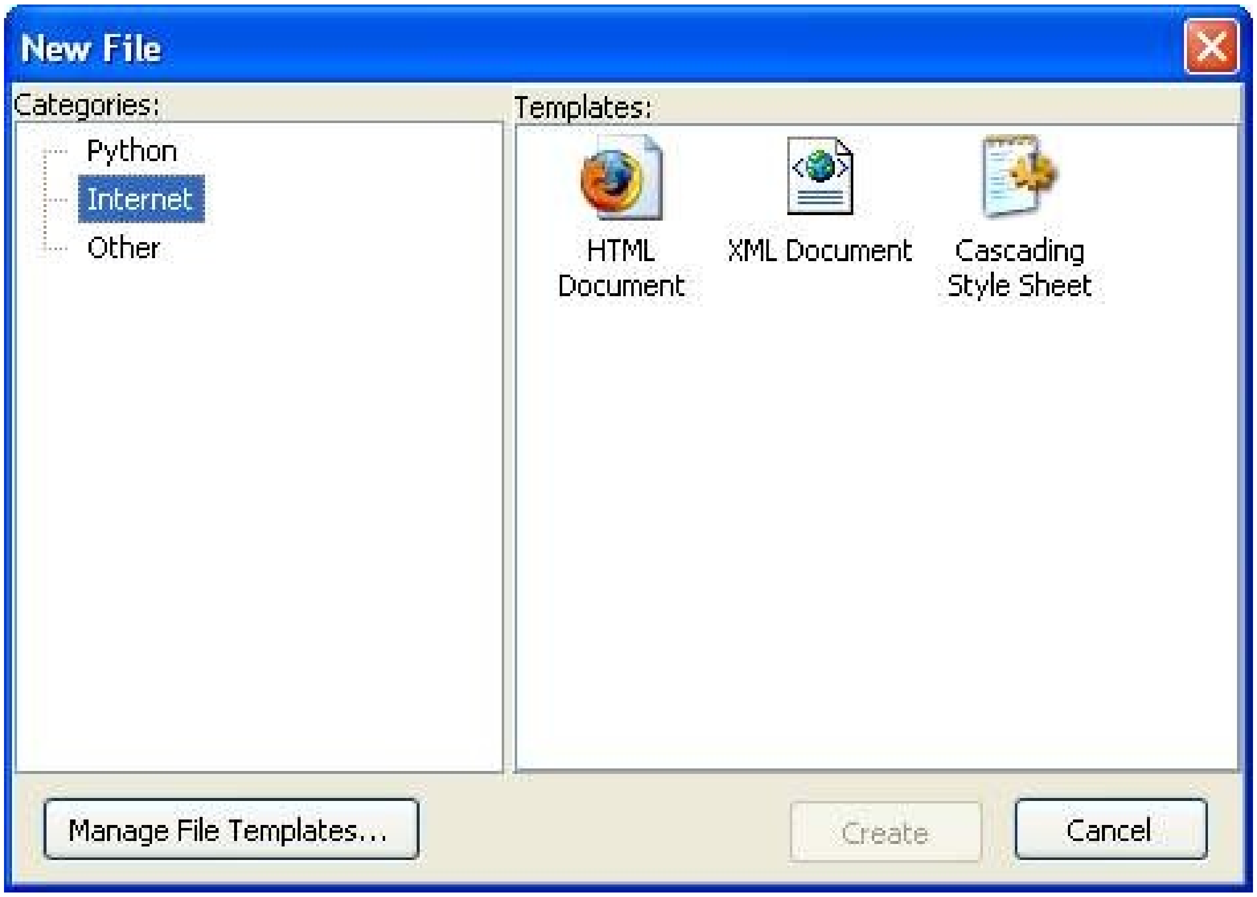
Previous Top Next

Since version 3.4 you can use the Setup Python dialog box to easily add and use unregistered versions to PyScripter. Virtual environments, both venv and virtualenv as well as conda distributions are supported out-of-the-box.

File Templates

The New File Dialog

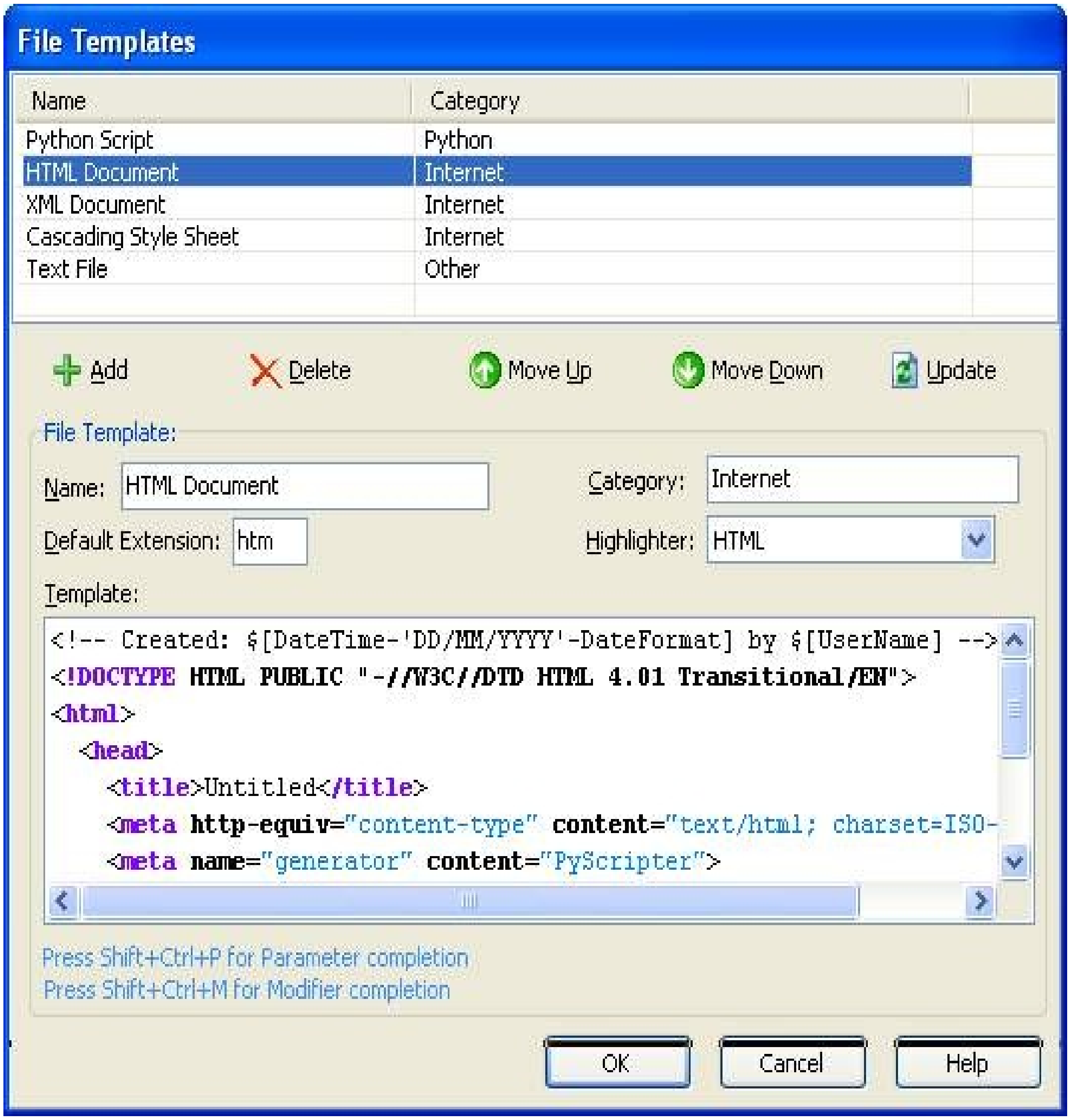
This dialog allows to create a new file based one of the many pre-defined file templates. You can access the dialog (shown below) from the File Menu selecting New, File...



In this dialog select a category in the tree view on the left side and a specific template on the right side. Then press the "Create" button to create a new file based on the selected template. The "Manage File Templates..." button allows you to customize the available templates (see below).

File Template Customization

You can customize file templates through the File Templates dialog shown below. This dialog is accessible either via the New File dialog shown above or through the Options submenu of the Tools Menu.



Name	Category
Python Script	Python
HTML Document	Internet
XML Document	Internet
Cascading Style Sheet	Internet
Text File	Other

File Template:

Name: HTML Document

Category: Internet

Default Extension: htm

Highlighter: HTML

Template:

```
<!-- Created: $[DateTime-'DD/MM/YYYY'-DateFormat] by $[UserName] -->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
  <head>
    <title>Untitled</title>
    <meta http-equiv="content-type" content="text/html; charset=ISO-
    <meta name="generator" content="PyScripter">
```

Press Shift+Ctrl+P for Parameter completion
Press Shift+Ctrl+M for Modifier completion

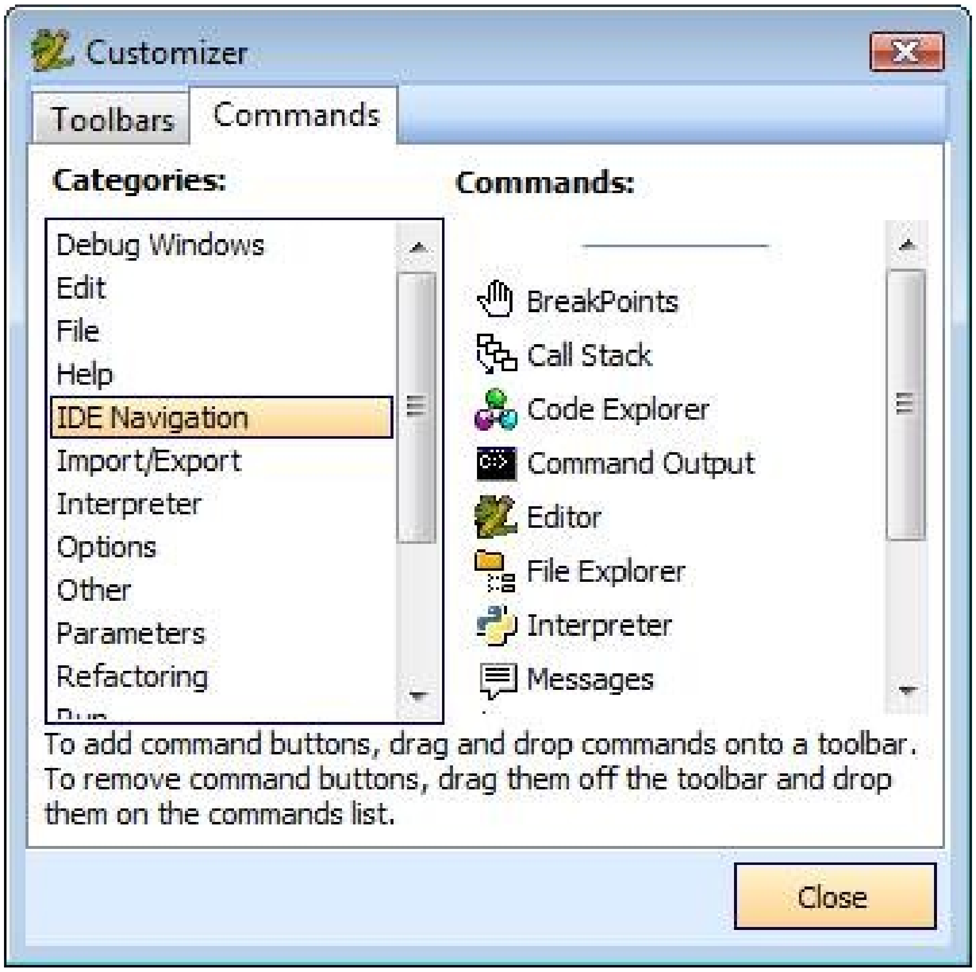
With this dialog you can modify add new templates or modify/delete existing templates. Each template has the following properties:

Name:	The name of the template that appears in the New File dialog
Category:	The category under which this template will be listed.
Default Extension:	The default extension that will be added to the filename when a file based on this template is saved.
Highlighter:	The syntax highlighter that will be used for files based on this template.
Template:	The actual text that will be inserted into new files created from this template. The template text can contain custom parameters which are expanded upon the activation of the template. If the character "|" is present in the template, after the insertion of the template text, the cursor is placed at the position of that character and the character is deleted.

Toolbar Customization

Previous Top Next

PyScripter allows you to customize the toolbars by dragging and dropping command buttons on the toolbars, as you do in Microsoft Office for example. You can invoke the customization dialog shown below either through the View, menu (Toolbars submenu), or from the context menu of the toolbars background.



To add command buttons, drag and drop commands onto a toolbar. To remove command buttons, drag them off the toolbar and drop them on the commands list.

Localization

Previous Top Next

PyScripter uses gettext for creating localizing the user interface. The files that contains the strings be translated is located in the directory:

C:\Program Files\PyScripter\locale

assuming that PyScripter is located at C:\Program Files\PyScripter\.

There are two files of interest to translators in that directory

1. default.po
2. languages.po

The first contains the strings of PyScripter and the second the localized names of different languages.

To create a translation for a new language:

1. Create a new directory

   C:\Program Files\PyScripter\locale\##\LC_MESSAGES\

   In this path, ## represents the two-letter ISO 639-1 language code.

2. Copy the two po files in that directory

3. Translate the two files using a gettext editor. Poedit is the recommended editor.

4. Compile the po files to mo files (Poedit can do that) and you are set. Use the View Menu to change the language and test your translation.

5. Please submit your translation files to pyscripter@gmail.com for inclusion in the next PyScripter distribution.

Tip: Use the translation memory of Poedit to speed up the translation process.

Customizable Styles

Previous Top Next

In version 3.0, PyScripter introduced an updated style (skin) engine that allows the loading of external themes. Style information is stored in files with the "vsf" extension. Available style files are located in the directory %APPDATA%\PyScripter\Styles (%APPDATA% is the directory pointed by the corresponding environment variable).

You can change the active style by using the "View, Select Style" command. Your choice is then saved and used in future activations of PyScripter.

Initialization Scripts

PyScripter can run initialization scripts that customize the program or the Python engine in different ways.

Two different scripts can be provided:

a) pyscripter_init.py

This file is run once after the program is loaded and can be used for intance to modify Pyscripter IDE options. It is run in the namespace of the internal interpreter.

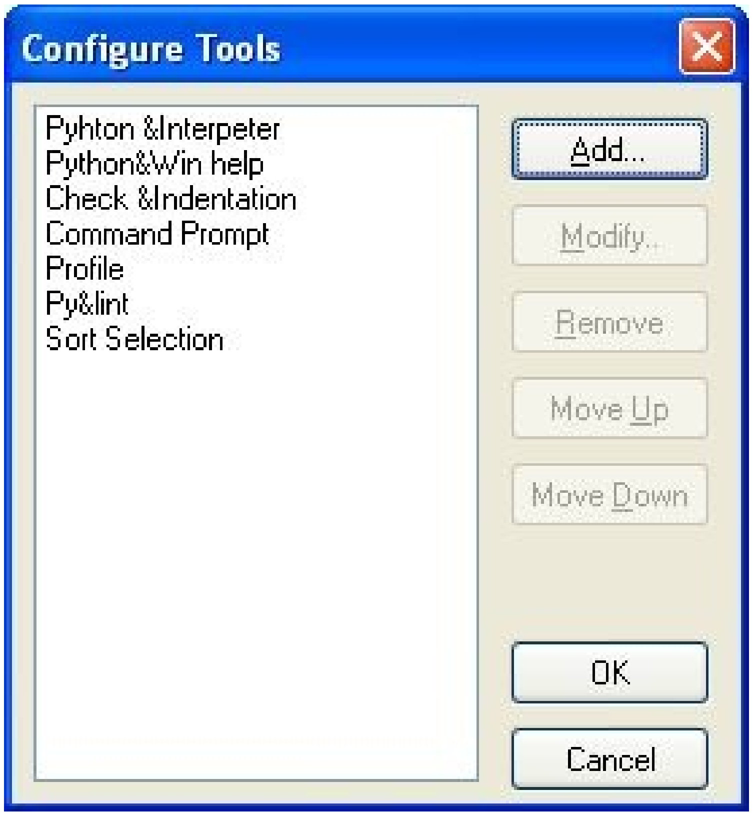
b) python_init.py

This file is run every time a Python engine is initialized. It can be used to customize the Python environment by importing certain units for instance.

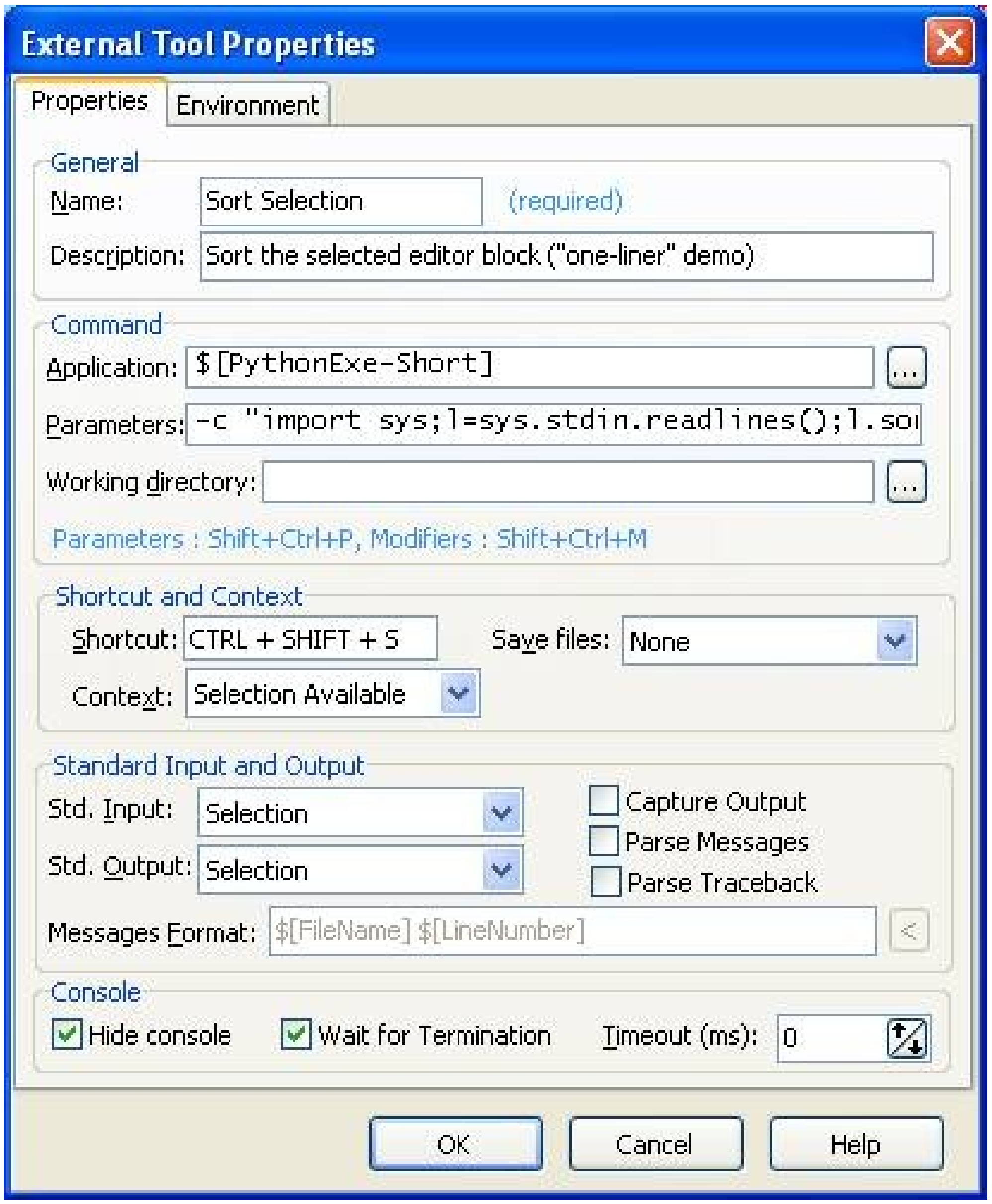
Both of these files should be located in the directory %APPDATA%\Pyscripter where %APPDATA% refers to the directory defined by the environment variable.

External Tools

PyScripter offers the ability to define External Tools that can be run independently or interact with the IDE editor.   There is great flexibility in specifying such tools allowing you to integrate your favorite Python utilities or command-line programs tightly with PyScripter.
You can define or modify external tools through the "Tools|Configure Tools.." menu option. The definition of the tools is persisted to the PyScritper.ini file.



In the above dialog box you can modify existing external tools or create new ones using the the External Tool properties dialog shown below:


External Tool Properties

Properties | Environment

General
Name: Sort Selection (required)
Description: Sort the selected editor block ("one-liner" demo)

Command
Application: $[PythonExe-Short]
Parameters: -c "import sys;l=sys.stdin.readlines();l.so
Working directory:
Parameters : Shift+Ctrl+P, Modifiers : Shift+Ctrl+M

Shortcut and Context
Shortcut: CTRL + SHIFT + S
Save files: None
Context: Selection Available

Standard Input and Output
Std. Input: Selection
Std. Output: Selection
☐ Capture Output
☐ Parse Messages
☐ Parse Traceback
Messages Format: $[FileName] $[LineNumber]

Console
☑ Hide console
☑ Wait for Termination
Timeout (ms): 0

A large number of options are available in the External Tool properties dialog which are explained below:

Tool Configuration Options

Caption	The caption of the Menu Item corresponding to this tool
Description	The hint of the Menu Item corresponding to this tool
Application	The full name of file to execute

Parameters	Command-line parameters
Working Directory	The working directory
Shortcut	Menu shortcut for the external tool
Context	Specifies when can the tool be executed • Always Enabled, • Active File, • Active Python File • Selection Available
Save Files	Save files option. Possible values • None, • Active File, • All Files
Standard Input	Feed standard input to the running process. Possible values: • None: No standard input • Word at cursor • Current line • Selection • Active file
Standard output	Send standard output to • None: Do nothing • Replace word at cursor • Replace current line • Replace selection • Replace active file • Place in new file
Parse Messages	Parse File/Line/LinePos info from output and put it in the Messages Window. Useful for integrating command-line tools.
Message Format	Regular expression for parsing messages. You should use the predefined grep expression macros $[FileName], $[LineNumber], $[ColumnNumber] for specifying the grep expression.
Parse Traceback	Parse TraceBack and Syntax Errors from Python output and put it in the Messages Window.

Capture Output	Capture command line output and place it in the Output Window
Hide Console	Hide Console or External Tool window
Wait for Termination	Non-blocking wait for termination of the External tool. Required for Parse Messages, Parse Traceback and other options
Timeout	Give the user the opportunity to terminate the External tool after Timeout ms. A value of zero disables this feature.
Environment	The Environment tab in tool properties allows you to run the external tool with customized environment variables.

Custom parameters (Shift+Ctrl+P) and modifiers (Shft+Ctrl+M) are available when specifying the Application, the Parameters and the Working directory.

A few external tools that demonstrate the possibilities opened by this feature of PyScripter are offered by default:

- PythonWin help (Shows the PythonWin help file)
- Python Interpreter (Runs a separate python shell)
- Command Prompt (starts a console)
- Sort Selection (a python one-liner that demonstrates the use of Standard Input and Standard Output options)
- Profiler (profiles the active python script using the standard Python profile module)
- PyLint - python source code checking tool (http://www.logilab.org/projects/pylint)
- Advanced Search and Replace (using re.sub)

You can delete or modify these tools as well as create new ones.

Parameters

Previous Top

Custom Parameters are implemented using ideas and code from the Syn Editor project (http:\\syn.sf.net). It is a very powerful feature allowing the developement of custom command line tools and facilitating autocompletion.

Parameter syntax
Parameter is any ParameterValue, that is enclosed by the parameter delimiters (currently "$[" and "]") e.g.

```
$[ProgramFiles]
```

Predefined system parameters (variables, that can be replaced in the command line, scripts, templates and in inserted text) are changed and extended. Now they include:

Python Paths

Python32Dir	- Installation directory of Python version 3.2
Python31Dir	- Installation directory of Python version 3.1
Python30Dir	- Installation directory of Python version 3.0
Python26Dir	- Installation directory of Python version 2.6
Python25Dir	- Installation directory of Python version 2.5
Python24Dir	- Installation directory of Python version 2.4
Python23Dir	- Installation directory of Python version 2.3
Python32Exe	- Executable of Python version 3.2
Python31Exe	- Executable of Python version 3.1
Python30Exe	- Executable of Python version 3.0
Python26Exe	- Executable of Python version 2.6
Python25Exe	- Executable of Python version 2.5
Python24Exe	- Executable of Python version 2.4

Python23Exe	- Executable of Python version 2.3
PythonDir	- Installation directory of active Python version
PythonExe	- Executable of active Python version
PythonVersion	- Version of active Python

Some parameters, that represent system folders for current configuration:

ProgramFiles	- Program Files folder for current configuration
CommonFiles	- Common Files folder for current configuration
Windows	- Windows folder for current configuration
WindowsSystem	- Windows System folder for current configuration
WindowsTemp	- Windows Temp folder for current configuration
MyDocuments	- My Documents folder for current user configuration
Desktop	- Desktop folder for current user configuration

Some parameters, that represent other system information:

CurrentDir	- Current Directory
DateTime	- Current date and time in the default short date-time format
UserName	- the name of the user currently logged onto the system
Paste	- returns contents of the clipboard
CmdLineArgs	- returns the active command line arguments

Some parameters, that represent PyScripter specific information:

ActiveDoc	- file name of the Active Document
ActiveScript	- file name of the Script you are about to run/debug
Project	- file name of the Active Project
OpenFiles	- file names of all open files, separated with space
ModFiles	- file names of modified files, separated with space
Exe	- parameter, pointing to PyScripter executable file

And some other useful:

SelectFile	- opens a FileOpen dialog for file selection
SelectedFile	- returns the last selected file
SelectDir	- opens "Browse for folder" dialog for folder selection
SelectedDir	- returns the last selected path

Custom parameter value can contain also other parameters and modifiers. They are calculated when the value is required, not when the parameter is loaded from file, so they always points to the actual other parameter value. For example

```
QuotedSelText=$[ActiveDoc-SelText-Quote]
```

will return quoted selected text of the active document

Parameter modifiers

We have also the so-called "Parameter modifiers" - Small words, that are placed after parameter, separated by '-' They actually represent one parameter functions, which modify value, returned from parameter. For the moment valid are:

Path	- extracts file path from filename (with '\' at the end)
Dir	- extracts file dir from filename (without '\' at the end)
Name	- extracts only file name
Ext	- extracts file extension
ExtOnly	- extracts file extension without dot before it
NoExt	- returns full file name without extension
Drive	- extracts only file drive
Full	- expands file name to absolute file name
UNC	- expands file name to absolute UNC file name
Long	- returns long file name
Short	- returns short file name
Sep	- adds '\' at the end, if there is no
NoSep	- removes '\' from the end, if there is any
Type	- returns file type of the file, as shown in the explorer
Text	- returns file text
EdText	- returns file text (but actual text from editor, if file is changed)
CurWord	- returns current word of the given file, if is open in editor
CurLine	- returns current line of the given file, if is open in editor
CurLineNumber	- returns current line of the given file, if is open in editor
SelText	- returns selected text in the given file, if is open in editor
Select	- selects text (modifier is valid only for "Insert test" tools)
Env	- returns environment variable (the parameter must contain

its name)

Reg	- returns registry key value for given registry key name
UpperCase	- converts parameter value to uppercase
LowerCase	- converts parameter value to lowercase
Quote	- adds quotes to parameter value
UnQuote	- removes quotes from parameter value
Date	- returns date portion from date time parameter
Time	- returns time portion from date time parameter
FileDate	- returns file date
DateCreate	- returns file creation date
DateWrite	- returns file last write date
DateAccess	- returns file last access date
DateFormat	- formats date with given format (the prior modifier must contain desired format)

Parameter syntax is extended so you can type

```
$[Parameter=DefaultValue]
```

which means, that if this Parameter is not found, Default Value will be used.

```
$[Parameter?Question]
```

will open a Input Box and ask with 'Question' for a value of Parameter.

```
$[Parameter=DefaultValue?Question]
```

will open a Input Box and ask with 'Question' for a value of Parameter, but if there is no value, will offer DefaultValue as default.

```
$['Some value']
```

is returned as it is (only parameters in SomeValue are replaced with their values) - this is useful, if you want to pass specific value to modifier -for example

```
$['31.01.2002'-'YYYY/MM/DD'-DateFormat]
```

will return you 2002.01.31

ParameterValue can be conditional parameter. Format is

```
$[(ParameterCondition)TrueValue:FalseValue]
```

where TrueValue and FalseValue are any valid ParameterValue. Symbol ':' is not required - if it is missing, its assumed that this means empty FalseValue. ParameterCondition can contain one or two ParameterValues and one of operations "=", "<>", "<", ">", "<=", ">=", "IS NULL" or "IS NOT NULL" or text to be asked in dialog (in single quotes) and "?" (question mark symbol) after it. In this case value will depend from the user input.

```
Condition1=$[($[Project] IS NULL)'There is no project open':'Project file is $[Project]']
Condition2=$[('Answer Yes or No'?)'Your answer was Yes':'Your answer was No']
```

Parameters can be placed AutoComplete items and Tools too.

Editor Shortcuts
You can use parameters in the IDE editor, the external tool configuration dialog and the Code Template definition. To facilitate the entry of parameters and modifiers PyScripter provides parameter and modifier completion (selection from a pop-up list) using the following shortcuts.

- Shft+Ctrl+P provides Parameter completion
- Shft+Ctrl+M provides Modifier completion
- Shft+Ctrl+R replaces all parameters with their values

LIST OF DEFINED PARAMETERS AND MODIFIERS

These are all parameters and their values at the moment of writing of this file.If you want to see their values, copy the following into an editor and select "Replace parameters" from "Edit menu"

System parameters:

CurrentDir=$[CurrentDir]
ProgramFiles=$[ProgramFiles]
CommonFiles=$[CommonFiles]
Windows=$[Windows]
WindowsSystem=$[WindowsSystem]
WindowsTemp=$[WindowsTemp]
MyDocuments=$[MyDocuments]
Desktop=$[Desktop]
Exe=$[Exe]
ActiveDoc=$[ActiveDoc]
ActiveScript=$[ActiveScript]
Project=$[Project]
ModFiles=$[ModFiles]
DateTime=$[DateTime]
UserName=$[UserName]
SelectFile=$[SelectFile]
SelectedFile=$[SelectedFile]
SelectDir=$[SelectDir]
SelectedDir=$[SelectedDir]
Paste=$[Paste]
OpenFiles=$[OpenFiles]

ParameterModifiers:

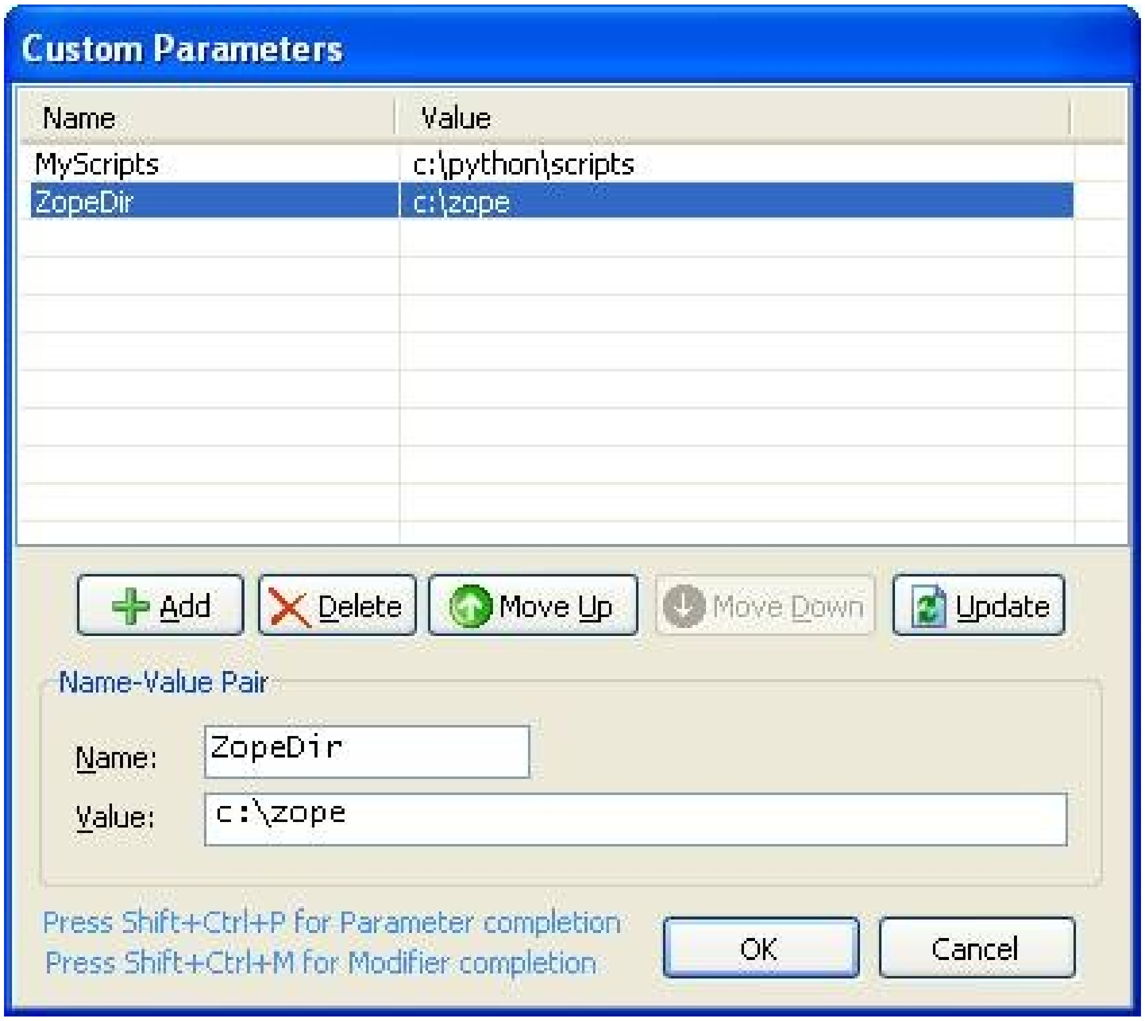
ActiveDoc-CurLine=$[ActiveDoc-CurLine]
ActiveDoc-CurWord=$[ActiveDoc-CurWord]
ActiveDoc-SelText=$[ActiveDoc-SelText]
'PATH'-Env=$['PATH'-Env]
PYTHON24DIR=$['HKLM\SOFTWARE\Python\PythonCore\2.4\Ins
Reg]
'0'-Param=$['0'-Param]
Exe-DateAccess=$[Exe-DateAccess]
Exe-DateCreate=$[Exe-DateCreate]
Exe-DateWrite=$[Exe-DateWrite]
Exe-FileDate=$[Exe-FileDate]
Exe-FileDate-'DD/MM/YYYY HH:NN:SS'-DateFormat=$[Exe-FileDate-'DD/MM/YYYY HH:NN:SS'-DateFormat]

```
Exe-FileDate-Date=$[Exe-FileDate-Date]
Exe-FileDate-Time=$[Exe-FileDate-Time]
Exe-Dir=$[Exe-Dir]
Exe-Dir-Sep=$[Exe-Dir-Sep]
Exe-Path=$[Exe-Path]
Exe-Path-NoSep=$[Exe-Path-NoSep]
Exe-Drive=$[Exe-Drive]
Exe-Ext=$[Exe-Ext]
Exe-Full=$[Exe-Full]
Exe-Long=$[Exe-Long]
Exe-LowerCase=$[Exe-LowerCase]
Exe-Name=$[Exe-Name]
Exe-NoExt=$[Exe-NoExt]
Exe-Quote=$[Exe-Quote]
Exe-Short=$[Exe-Short]
Exe-Type=$[Exe-Type]
Exe-UNC=$[Exe-UNC]
Exe-UpperCase=$[Exe-UpperCase]

Day=$[DateTime-'DD'-DateFormat]
Month=$[DateTime-'MM'-DateFormat]
Year=$[DateTime-'YYYY'-DateFormat]
Hour=$[DateTime-'HH'-DateFormat]
Minutes=$[DateTime-'NN'-DateFormat]
Seconds=$[DateTime-'SS'-DateFormat]
```

Defining your own custom parameters

You can define your own custom parameters by selecting the "Custom Parameters..." menu command under Tools|Options. The Custom Parameters dialog is displayed below:



Name	Value
MyScripts	c:\python\scripts
ZopeDir	c:\zope

Name-Value Pair

Name: ZopeDir

Value: c:\zope

Press Shift+Ctrl+P for Parameter completion
Press Shift+Ctrl+M for Modifier completion