Wing IDE Reference Manual

Wingware, the feather logo, Wing IDE, Wing IDE 101, Wing IDE Personal, Wing IDE Professional, and "The Intelligent Development Environment" are trademarks or registered trademarks of Wingware in the United States and other countries.

Disclaimers: The information contained in this document is subject to change without notice. Wingware shall not be liable for technical or editorial errors or omissions contained in this document; nor for incidental or consequential damages resulting from furnishing, performance, or use of this material.

Hardware and software products mentioned herein are named for identification purposes only and may be trademarks of their respective owners.

Copyright (c) 1999-2014 by Wingware. All rights reserved.

```
Wingware
P.O. Box 400527
Cambridge, MA  02140-0006
United States of America
```

Contents

Wing IDE Reference Manual **1**

Introduction 1

1.1. Product Levels 1

1.2. Licenses 1

1.3. Supported Platforms 2

1.4. Supported Python versions 2

1.5. Technical Support 3

1.6. Prerequisites for Installation 3

1.7. Installing Wing IDE 3

1.8. Running Wing IDE 4

1.9. Installing your License 4

1.10. User Settings Directory 6

1.11. Upgrading 7

Compatibility Notes 7

1.11.1. Migrating from older versions of Wing 8

Licensing 8

Converting Projects 8

Changes in Wing 5 8

1.11.2. Fixing a Failed Upgrade 9

1.12. Installation Details and Options 10

1.12.1. Linux Installation Notes 10

1.12.2. Remote Display on Linux 11

1.12.3. Installing Extra Documentation 12

1.12.4. Source Code Installation 12

1.13. Removing Wing IDE 12

1.14. Command Line Usage 12

Customization 14

2.1. Keyboard Personalities 14

2.1.1. Key Equivalents 15

2.1.2. Key Maps 16

2.1.3. Key Names 18

2.2. User Interface Options 19
2.2.1. Display Style and Colors 19
Color Configuration 19
Add Color Palettes 20
2.2.2. Windowing Policies 20
2.2.3. User Interface Layout 20
2.2.4. Altering Text Display 22
2.3. Preferences 22
2.3.1. Preferences File Layers 22
2.3.2. Preferences File Format 23
2.4. Syntax Coloring 23
Minor Adjustments 23
Comprehensive Changes 24
Automatic Color Adjustment 24
2.5. Perspectives 24
2.6. File Filters 26
Project Manager 27
3.1. Creating a Project 27
3.2. Removing Files and Directories 28
3.3. Saving the Project 28
3.4. Sorting the View 29
3.5. Navigating to Files 29
3.5.1. Keyboard Navigation 30
3.6. Sharing Projects 30
3.7. Project-wide Properties 31
Environment 32
Debug 32
Options 33
Extensions 34
Testing 34
3.7.1. Environment Variable Expansion 35
3.8. Per-file Properties 35

File Attributes 35
Editor 36
Debug/Execute 37
Testing 37
3.9. Launch Configurations 37
Shared Launch Configurations 38
Working on Different Machines or OSes 38
Source Code Editor 39
4.1. Syntax Colorization 39
4.2. Right-click Editor Menu 40
4.3. Navigating Source 40
4.4. File status and read-only files 41
4.5. Transient, Sticky, and Locked Editors 41
4.6. Auto-completion 42
4.7. Source Assistant 44
4.7.1. Docstring Type and Validity 44
4.7.2. Python Documentation Links 45
4.7.3. Working with Runtime Type Information 45
4.7.4. Source Assistant Options 45
4.8. Auto-editing 46
4.9. Bookmarks 48
4.10. File Sets 49
Binding File Sets to Keys 49
Shared File Sets 50
4.11. Code Snippets 50
User Interface 50
Contexts 51
Key Bindings 51
Execution and Data Entry 51
Auto-completion 52
Snippet Syntax 52
Indentation and Line Endings 53

Cursor Placement 54
Snippet Directory Layout 54
File Types 54
Contexts 54
Configuration 54
Commands 54
Scripting Snippets 55
4.12. Indentation 55
4.12.1. How Indent Style is Determined 55
4.12.2. Indentation Preferences 55
4.12.3. Indentation Policy 56
4.12.4. Auto-Indent 57
4.12.5. The Tab Key 57
4.12.6. Checking Indentation 59
4.12.7. Changing Block Indentation 59
4.12.8. Indentation Manager 59
4.13. Folding 60
4.14. Brace Matching 61
4.15. Support for files in .zip or .egg files 61
4.16. Keyboard Macros 62
4.17. Notes on Copy/Paste 62
Smart Copy 62
4.18. Auto-reloading Changed Files 63
4.19. Auto-save 63
Search/Replace 63
5.1. Toolbar Quick Search 63
5.2. Keyboard-driven Mini-Search/Replace 64
5.3. Search Tool 65
5.4. Search in Files Tool 66
5.4.1. Replace in Multiple Files 66
5.5. Find Points of Use 67
5.6. Wildcard Search Syntax 68

Refactoring 68
6.1. Rename Symbol 68
6.2. Move Symbol 69
6.3. Extract Function / Method 69
6.4. Introduce Variable 70
Diff/Merge Tool 70
Diff/Merge Options 71
Source Code Browser 71
8.1. Display Choices 72
8.1.1. Browse Project Modules 72
8.1.2. Browsing Project Classes 73
8.1.3. Viewing Current Module 73
8.2. Display Filters 73
8.2.1. Filtering Scope and Source 73
8.2.2. Filtering Construct Type 73
8.3. Sorting the Browser Display 74
8.4. Navigating the Views 74
8.5. Browser Keyboard Navigation 74
Interactive Python Shell 75
9.1. Python Shell Auto-completion 76
9.2. Python Shell Options 76
OS Commands Tool 76
10.1. OS Command Properties 77
Unit Testing 79
11.1. Project Test Files 80
11.2. Running Tests 80
11.3. Running unittest Tests From the Command Line 81
Debugger 81
12.1. Quick Start 82
12.2. Specifying Main Entry Point 83
12.2.1. Named Entry Points 83
12.3. Debug Properties 84

12.4. Setting Breakpoints 84
Breakpoint Types 84
Breakpoint Attributes 84
Breakpoints Tool 85
Keyboard Modifiers for Breakpoint Margin 85
12.5. Starting Debug 85
12.6. Debugger Status 87
12.7. Flow Control 87
12.8. Viewing the Stack 88
12.9. Viewing Debug Data 89
12.9.1. Stack Data View 89
12.9.1.1. Popup Menu Options 90
12.9.1.2. Filtering Value Display 91
12.9.2. Watching Values 92
12.9.3. Evaluating Expressions 93
12.9.4. Problems Handling Values 93
12.10. Debug Process I/O 94
12.10.1. External I/O Consoles 95
12.10.2. Disabling Debug Process I/O Multiplexing 95
12.11. Interactive Debug Probe 96
12.11.1. Managing Program State 97
12.11.2. Debug Probe Options 97
12.12. Debugging Multi-threaded Code 98
12.13. Managing Exceptions 98
Exception Reporting Mode 99
Reporting Logged Exceptions 100
Exception Type Filters 100
12.14. Running Without Debug 101
Advanced Debugging Topics 101
13.1. Debugging Externally Launched Code 101
13.1.1. Importing the Debugger 101
13.1.2. Debug Server Configuration 103

13.1.3. Debugger API 104
13.1.4. Debugging Embedded Python Code 106
13.2. Remote Debugging 106
13.2.1. File Location Maps 108
13.2.1.1. File Location Map Examples 108
13.2.2. Remote Debugging Example 110
13.2.3. Installing the Debugger Core 111
13.3. Attaching and Detaching 112
13.3.1. Access Control 112
13.3.2. Detaching 112
13.3.3. Attaching 113
13.3.4. Identifying Foreign Processes 113
13.3.5. Constraints 113
13.4. OS X Debugging Notes 114
System-Provided Python 114
MacPorts Python 114
Debugging 32-bit Python on a 64-bit System 114
13.5. Debugger Limitations 114
Integrated Version Control 117
14.1. Version Control Tool Panel 118
14.2. Common Version Control Operations 118
14.3. Bazaar 120
14.4. CVS 120
14.5. Git 120
14.6. Mercurial 121
14.7. Perforce 122
14.8. Subversion 122
14.9. Version Control Configuration 123
14.9.1. Configuring SSH 123
14.9.2. Configuring Subversion 124
14.9.3. Configuring CVS 125
Source Code Analysis 126

15.1. How Analysis Works 126
15.2. Static Analysis Limitations 127
15.3. Helping Wing Analyze Code 127
15.4. Analysis Disk Cache 129
PyLint Integration 129
Scripting and Extending Wing IDE 131
17.1. Scripting Example 131
17.2. Getting Started 132
Naming Commands 133
Reloading Scripts 133
Overriding Internal Commands 133
17.3. Script Syntax 134
Script Attributes 134
ArgInfo 135
Commonly Used Types 135
Commonly Used Formlets 135
Magic Default Argument Values 137
GUI Contexts 137
Top-level Attributes 138
Importing Other Modules 138
Internationalization and Localization 138
Plugins 139
17.4. Scripting API 140
17.5. Advanced Scripting 140
Example 140
How Script Reloading Works 141
Trouble-shooting Guide 141
18.1. Trouble-shooting Failure to Start 142
18.2. Speeding up Wing 143
18.3. Trouble-shooting Failure to Debug 143
18.3.1. Failure to Start Debug 143
18.3.2. Failure to Stop on Breakpoints or Show Source Code 145

18.3.3. Failure to Stop on Exceptions 146
18.3.4. Extra Debugger Exceptions 147
18.4. Trouble-shooting Other Known Problems 147
18.5. Obtaining Diagnostic Output 148
Preferences Reference 149
User Interface 149
Projects 157
Files 159
Editor 163
Debugger 180
Source Analysis 190
Version Control 193
IDE Extension Scripting 197
Network 198
Internal Preferences 198
Core Preferences 198
User Interface Preferences 200
Editor Preferences 203
Project Manager Preferences 205
Debugger Preferences 205
Command Reference 208
20.1. Top-level Commands 209
Application Control Commands 209
Dock Window Commands 221
Document Viewer Commands 222
Global Documentation Commands 224
Window Commands 224
Wing Tips Commands 224
20.2. Project Manager Commands 225
Project Manager Commands 225
Project View Commands 227
20.3. Editor Commands 228

Editor Browse Mode Commands 228
Editor Insert Mode Commands 229
Editor Non Modal Commands 229
Editor Panel Commands 230
Editor Replace Mode Commands 230
Editor Split Commands 231
Editor Visual Mode Commands 231
Active Editor Commands 232
General Editor Commands 249
Shell Or Editor Commands 261
Bookmark View Commands 261
Snippet Commands 262
Snippet View Commands 262
20.4. Search Manager Commands 263
Toolbar Search Commands 263
Search Manager Commands 265
Search Manager Instance Commands 267
20.5. Unit Testing Commands 267
Unit Testing Commands 267
20.6. Version Control Commands 269
Subversion Commands 269
Git Commands 270
Bazaar Commands 270
C V S Commands 271
Mercurial Commands 272
Perforce Commands 272
Version Control Command Map 273
20.7. Debugger Commands 274
Debugger Commands 274
Debugger Watch Commands 279
Call Stack View Commands 280
Exceptions Commands 280

Breakpoint View Commands 280
20.8. Script-provided Add-on Commands 280
Debugger Extensions Script 280
Django Script 281
Django Script 282
Editor Extensions Script 282
Emacs Extensions Script 286
Pylintpanel Script 286
Testapi Script 287
Key Binding Reference 287
21.1. Wing IDE Personality 287
21.2. Emacs Personality 301
21.3. VI/VIM Personality 319
21.4. Visual Studio Personality 346
21.5. OS X Personality 361
21.6. Eclipse Personality 374
21.7. Brief Personality 404
License Information 417
22.1. Wing IDE Software License 417
22.2. Open Source License Information 424

Introduction

Thanks for choosing Wingware's Wing IDE! This manual will help you get started and serves as a reference for the entire feature set.

The manual is organized by major functional area of Wing IDE, including customization, project manager, source code editor, search/replace features, refactoring, diff/merge, source code browser, python shell, OS commands tool, unit testing, debugger, version control, source code analysis, PyLint integration, ad scripting and extending Wing IDE. Several appendices provide trouble-shooting assistance, document the key bindings & command set, provide pointers to resources and tips for Wing and Python users, and list the full software license.

The rest of this chapter describes how to install and start using Wing IDE. See also the quick start guide and tutorial.

1.1. Product Levels

This manual is for the Wing IDE Professional product level of the Wing IDE product line, which currently includes Wing IDE Professional, Wing IDE Personal, and Wing IDE 101.

Wing IDE Professional is the full-featured Wing IDE product, and may be licensed for commercial or non-commercial uses. Wing IDE Personal is for non-commercial use only and contains a subset of the features found in Wing IDE Professional. Both products are commercial products for sale from our website; Wing IDE Personal is not a free download.

Wing IDE 101 is a heavily scaled back IDE that was designed for teaching entry level computer science courses. It is free to download and use for educational and personal use.

Wing IDE Professional, Wing IDE Personal, and Wing IDE 101 are independent products and may be installed at the same time on your system without interfering with each other.

For a list of features in each product level, please refer to http://wingware.com/wingide/features.

1.2. Licenses

Wing IDE requires a separate license for each developer working with the product. For the full license text, see the Software License.

License Activation

To run for more than 10 minutes, Wing IDE requires activation of a time-limited trial or permanent purchased license. Time-limited trials last for 10 days and can be renewed two times, for a total or 30 days.

An activation ties the license to the machine through a series of checks of the hardware connected to the system. This information is never transmitted over the internet. Instead an SHA hash of some of the values is passed back and forth so that the machine will be identifiable without us knowing anything specific about it.

The machine identity metrics used for activation are designed to be forgiving so that replacing parts of your machine's hardware or upgrading the machine will usually not require another activation. By the same token, activating multiple times on the same machine (for example if the activation file is lost) usually does not increase your activation count.

Licenses come with ten activations per year by default and additional activations can be obtained from the self-serve license manager or by emailing sales at wingware.com. As a fall-back in cases of emergency where we cannot be contacted and you don't have an activation, Wing IDE will run for 10 minutes at a time without any license at all, or a trial license can be used until any license problem is resolved.

See Installing Your License for more information on obtaining and activating licenses.

1.3. Supported Platforms

This version of Wing IDE is available for Microsoft Windows, Linux, and Mac OS X.

Microsoft Windows

Wing IDE supports Windows XP, 2003 Server, Vista, Windows 7, and Windows 8 for Intel processors. Earlier versions of Windows are not supported and will not work.

Linux/Intel

Wing IDE runs on Linux versions with glibc version 2.6 or later (such as Ubuntu 10+, RHEL 6.4+, and Debian 5.0+).

Mac OS X

Wing IDE runs on Mac OS X 10.6+ as a native application.

1.4. Supported Python versions

Wing supports CPython 2.5 through 3.4, Stackless Python 2.5 through 3.2, and cygwin Python 2.5 through 2.7. Wing can also be used with PyPy, IronPython, and Jython, but the debugger will not work with these implementations of Python.

Wing's debugger is pre-built for each of these versions of Python with and without **--with-pydebug**. Both 32-bit and 64-bit compilations are supported. CPython **--with-framework** builds are also supported on OS X. If necessary, it is possible for customers to compile Wing's debugger against other custom versions of Python.

Before installing Wing, you may need to download Python and install it if you do not already have it on your machine.

On Windows, Python must be installed using one of the installers from the python.org (or by building from source if desired).

On Linux, most distributions come with Python. Installing Python is usually only necessary on a custom-built Linux installation.

On OS X, an Python built by Apple is installed by default. Other Python versions are available from python.org and from MacPorts, Fink, or Homebrew

1.5. Technical Support

If you have problems installing or using Wing IDE, please submit a bug report or feedback using the **Submit Bug Report** or **Submit Feedback** items in Wing IDE's **Help** menu.

Wingware Technical Support can also be contacted by email at support at wingware.com, or online at http://wingware.com/support.

Bug reports can also be sent by email to bugs at wingware.com. Please include your OS and product version number and details of the problem with each report.

If you are submitting a bug report via email, see Obtaining Diagnostic Output for more information on how to capture a log of Wing IDE and debug process internals. Whenever possible, these should be included with email-based bug reports.

1.6. Prerequisites for Installation

To run Wing IDE, you will need to obtain and install the following, if not already on your system:

- A downloaded copy of Wing IDE
- A supported version of Python
- A working TCP/IP network configuration (for the debugger; no outside access to the internet is required)

1.7. Installing Wing IDE

Before installing Wing IDE, be sure that you have installed the necessary prerequisites. If you are upgrading from a previous version, see Upgrading first.

Note: The installation location for Wing IDE is referred to as **WINGHOME**. On OS X this is the name of Wing's **.app** folder.

Windows

Install Wing IDE by running the downloaded executable. Wing's files are installed by default in **C:\Program Files\Wing IDE 5.0**, but this location may be modified during installation. Wing will also create a User Settings Directory in the location appropriate for your version of Windows. This is used to store preferences and other settings.

The Windows installer supports a **/silent** command line option that uses the default options, including removing any prior install of version 5.0 of Wing IDE. If a prior install is removed, a dialog with a progress bar will appear. You can also use a **/dir=<dir name>** option to specify an alternate installation directory.

Linux

Use the RPM, Debian package, or tar file installer as appropriate for your system type. Installation from packages is at **/usr/lib/wingide5** or at the selected location when installing from the tar file. Wing will also create a User Settings Directory in **~/.wingide5**, which is used to store preferences and other settings.

For more information, see the Linux installation details.

Mac OS X

On OS X, Wing is installed simply by opening the distributed disk image and dragging to the Applications folder, and optionally from there to the task bar.

1.8. Running Wing IDE

For a quick introduction to Wing's features, refer to the Wing IDE Quickstart Guide. For a more gentle in-depth start, see the Wing IDE Tutorial.

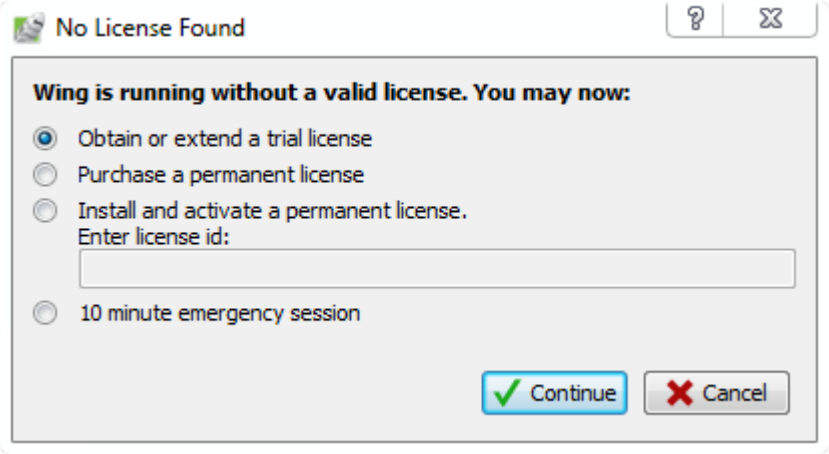
On Windows, start Wing IDE from the Program group of the Start menu. You can also start Wing from the command line with **wing.exe** (located inside the Wing IDE installation directory).

On Linux/Unix, just execute **wing5.0** (or **wing** located inside the Wing IDE installation directory).

On Mac OS X, start Wing IDE by double clicking on the app folder or from the command line using **wing** command inside the Wing IDE **.app** folder.

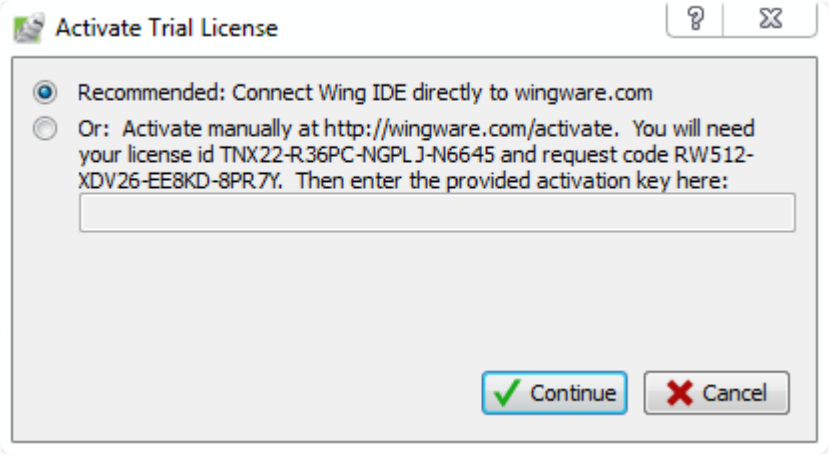
1.9. Installing your License

Wing IDE requires a time-limited trial or permanent license and the license needs to be activated on each machine (see the Licenses section for general information). When Wing IDE is first started, you can obtain a trial licence, purchase a permanent license, install & activate a permanent license, or use Wing for up to 10 minutes without any license:



Trial Licenses

Trial licenses allow evaluation of Wing IDE for 10 days, with an option to extend the evaluation twice for up to 30 days total (or more on request). The most convenient way to obtain a trial license is to ask Wing IDE to connect directly to **wingware.com** (via http, TCP/IP port 80). After the trial license is obtained, Wing will not attempt to connect to **wingware.com** (or any other site) unless you submit feedback or a bug report through the **Help** menu.



If you're unable or unwilling to connect Wing IDE directly to **wingware.com**, you can go to http://wingware.com/activate and enter the license id and activation request number obtained from Wing. After entering this information, you will be given an activation key which you can enter into Wing's dialog box to complete the activation. This is exactly the same exchange of information that occurs when Wing IDE connects directly to **wingware.com** to obtain a trial license.

If activation fails, Wing will provide a way to configure an http proxy. Wing tries to detect and use proxies by default but in some cases they will need to be manually configured. Please ask your network administrator if you do not know what proxy settings to use. See also how to determine proxy settings.

If you run into problems or need additional evaluation time, please email us at sales at wingware.com.

Permanent Licenses

Permanent licenses and upgrades may be purchased in the online store at http://wingware.com/store. Permanent licenses include free upgrades through the 5.* version series. Wing IDE Professional licenses also allow access to the product source code via http://wingware.com/downloads (requires signed non-disclosure agreement).

Activating on Shared Drives

When Wing is installed on a shared drive (for example a USB keydrive, or on a file server), the User Settings Directory where the license activation is stored may be accessed from several different computers.

In this case, Wing must be activated once on each computer. The resulting extra activations will be stored as **license.act1**, **license.act2**, and so forth, and Wing will automatically select the appropriate activation depending on where it is running.

Obtaining Additional Activations

If you run out of activations, you can use the self-serve license manager or email us at sales at wingware.com to obtain additional activations on any legitimately purchased license.

Deactivating a License

If you wish to deactivate and remove your license number from a machine, click **License** in Wing IDE's **About** dialog box and then **Deactivate**. This will remove the license activation and quit Wing.

Note that this just removes your license number from the machine. If you are out of activations you will still need to follow the instructions in **Obtaining Additional Activations** above.

1.10. User Settings Directory

The first time you run Wing, it will create your **User Settings Directory** automatically. This directory is used to store your license, preferences, auto-save files, recent lists, and other files used internally by Wing. If the directory cannot be created, Wing will exit.

The settings directory is created in a location appropriate to your operating system. The location is listed as your **Settings Directory** in the **About Box** accessible from the **Help** menu.

These are the locations used by Wing:

Linux/Unix -- **~/.wingide5** (a sub-directory of your home directory)

Windows -- In **Wing IDE 5** within the per-user application data directory. The location varies by version of Windows. For Windows 2000 and XP running on **c:** with an English localization the location is:

```
c:\Documents and Settings\${username}\Application Data\Wing IDE 5
```

For Vista running on **c:** with an English localization the location is:

```
c:\Users\${username}\AppData\Roaming\Wing IDE 5
```

Wing also creates a Cache Directory that contains the source analysis cache. This is often but not always in the same location as the above. On Windows, this directory is usually in the per-user directory under **Local Settings** on 2000 and XP and under **Local** on Vista. This directory is also listed in the **About Box**.

1.11. Upgrading

If you are upgrading within the same minor version number of Wing (for example from 5.0 to 5.0.x) this will replace your previous installation. Once you have upgraded, your previous preferences and settings should remain and you should immediately be able to start using Wing.

If you are upgrading across major releases (for example from 4.1 to 5.0), this will install the new version along side your old version of Wing.

New major releases of Wing will read and convert any existing Wing preferences, settings, and projects. Projects should be saved to a new name for use with the new major release since they cannot be read by earlier versions.

To install an upgrade, follow the steps described in Installing

Compatibility Notes

The following compatibility notes may be useful to users of earlier versions of Wing IDE:

- **File Sets** from Wing IDE 4 are now called **File Filters**. **File Sets** in Wing 5 are a new feature for creating named sets of files.
- Auto-editing is on by default except for auto-entering spaces and block management with the colon key
- Preferences and some menus have been reorganized
- File Properties have changed to make use of **Launch Configurations**
- **OS Commands** has been redesigned to use **Named Entry Points** and **Launch Configurations**.
- The **Python Shell** now starts with current directory set according to the project or main debug file configuration
- A few key bindings have changed

- Accelerators may have changed from Wing 4 and are no longer available on OS X because Qt follows Mac user interface guidelines, except when the **Display Style** preference is used to select non-native display
- More project data is stored in the shared branch of two-file projects (such as **Python Path** and **OS Commands** settings)
- Support for Python < 2.5 has been dropped
- New projects created with Wing Pro are always shared (two file) projects. Only the .wpr file should be checked into revision control.
- For other compatibility details, see the **Changes** sections of the Change Log on the downloads page.

1.11.1. Migrating from older versions of Wing

Moving to Wing IDE 5 from earlier versions should be easy. The first time you start Wing IDE 5, it will automatically convert your preferences from any older version of Wing IDE and place them into your User Settings Directory.

Wing IDE 5 can be installed and used side by side with older versions of Wing and operates completely independently.

Licensing

Licenses for Wing IDE 4 and earlier must be upgraded before they can be activated for Wing IDE 5. This can be done in the online store.

Converting Projects

Wing IDE 4 and earlier project files will be converted as they are opened and marked untitled so they can be saved under a new name. You should not overwrite your old project files if you plan to continue using an earlier version of Wing IDE with them. Wing IDE 5 project files cannot be read by earlier versions of Wing.

Changes in Wing 5

Wing 5 changes some features and defaults found in earlier versions of Wing:

- Auto-editing is on by default except for auto-entering spaces and block management with repeated presses of the colon key
- Preferences and some menus have been reorganized
- File Sets from Wing IDE 4 are now called File Filters. File Sets in Wing 5 are a new feature for creating named sets of files.
- File Properties have changed to make use of Launch Configurations
- OS Commands has been redesigned to use Launch Configurations and Named Entry Points
- The Python Shell starts with current directory set according to the project or main debug file configuration

- A few key bindings have changed
- Accelerators may have changed from Wing 4 and are no longer available on OS X because Qt follows Mac user interface guidelines, except when the Display Style preference is used to select non-native display
- More project data is stored in the shared branch of two-file projects (Python Path, OS Commands)
- Support for Python <= 2.2 has been dropped on Windows and support for Python <= 2.5 has been dropped on OS X
- New projects created with Wing Pro are always Shared (Two File) projects

1.11.2. Fixing a Failed Upgrade

In rare cases upgrading may fail to overwrite old files, resulting in random or bizarre behaviors and crashing. The fix for this problem is to completely uninstall and manually remove remaining files before installing the upgrade again.

Windows

To uninstall on Windows, run the Add/Remove Programs control panel to uninstall Wing IDE. Then go into the directory where Wing was located and manually remove any remaining folders and files.

Mac OS X

On Mac OS X, just drag the entire Wing IDE application folder to the trash.

Linux Debian

If you installed Wing IDE for Linux from Debian package, issue the command **dpkg -r wingide5**. Then go into **/usr/lib/wingide5** and remove any remaining files and directories.

Linux RPM

If you installed Wing IDE for Linux from RPM, issue the command **rpm -e wingide5**. Then go into **/usr/lib/wingide5** and remove any remaining files and directories.

Linux Tar

If you installed Wing IDE for Linux from the tar distribution, find your Wing installation directory and run the **wing-uninstall** script located there. Once done, manually remove any remaining files and directories.

If this procedure does not solve the problem, try moving aside the User Settings Directory and then starting Wing. If this works, try restoring files from the old user settings directory one by one to find the problem. Key files to try are **license.act***, **preferences** and **recent***. Then submit a bug report to **support@wingware.com** with the offending file.

1.12. Installation Details and Options

This section provides some additional detail for installing Wing and describes installation options for advanced users.

1.12.1. Linux Installation Notes

On Linux, Wing can be installed from RPM, Debian package, or from tar archive. Use the latter if you do not have root access on your machine or wish to install Wing somewhere other than **/usr/lib/wingide5**. Be sure to use the 64-bit packages if you are on a 64-bit system.

Installing from RPM:

Wing can be installed from an RPM package on RPM-based systems, such as RedHat and Mandriva. To install, run **rpm -i wingide5-5.0.9-1.i386.rpm** as root or use your favorite RPM administration tool to install the RPM. Most files for Wing are placed under the **/usr/lib/wingide5** directory and the **wing5.0** command is placed in the **/usr/bin** directory.

Installing from Debian package:

Wing can be installed from a Debian package on Debian, Ubuntu, and other Debian-based systems.

You will need to install **enscript** before installing Wing, if it's not already on your system.

To install, run **dpkg -i wingide5_5.0.9-1_i386.deb**

as root or use your favorite package administration tool to install. Most files for Wing are placed under the **/usr/lib/wingide5** directory and the **wing5.0** command is placed in the **/usr/bin** directory.

Installing from Tar Archive:

Wing may also be installed from a tar archive. This can be used on systems that do not use RPM or Debian packages, or if you wish to install Wing into a directory other than **/usr/lib/wingide5**. Unpacking this archive with **tar -zxvf wingide-5.0.9-1-i386-linux.tar.gz** will create a **wingide-5.0.9-1-i386-linux** directory that contains the **wing-install.py** script and a **binary-package.tar** file.

Running the **wing-install.py** script will prompt for the location to install Wing, and the location in which to place the executable **wing5.0**. These locations default to **/usr/local/lib/wingide** and **/usr/local/bin**, respectively. The install program must have read/write access to both of these directories, and all users running Wing must have read access to both.

Debugging 32-bit Python on 64-bit Systems

On a 64-bit system where you need to debug 32-bit Python, you will need to install the 32-bit version of Wing. This version can also debug 64-bit Python.

Installing the 32-bit version of Wing may require installing some compatibility packages as follows:

On 64-bit Ubuntu and Debian systems, you need to first install the 32 bit compatibility libraries. This is the **ia32-libs** package on Ubuntu. Then install the 32-bit Wing with the command **dpkg -i --force-architecture --force-depends wingide5_5.0.9-1_.i386.deb** The package contains what you need to run your debug process with 64-bit Python but Wing itself runs as a 32-bit application.

On CentOS 64-bit systems, installing the **libXtst.i386** package with **yum** provides the necessary 32 bit support.

On Arch linux, the necessary package is instead named **lib32-glibc**.

1.12.2. Remote Display on Linux

Wing for Linux can be displayed remotely by enabling X11 forwarding in ssh as described here.

In summary: You need to send the **-X** option to ssh when you connect from the machine where you want windows to display to the machine where Wing will be running, and you need to add **X11Forwarding yes** to your ssh configuration (usually in **~/.ssh/config**) on the machine where Wing will be running.

Speeding up the Connection

To improve performance, in most cases you should leave off the **-C** option for ssh, even though it is often mentioned in instructions for setting up X11 forwarding. The compression that is enabled with **-C** is only useful over extremely slow connections and otherwise increases latency and reduces responsiveness of the GUI.

Another option to try is **-Y** (trusted X11 port forwarding) instead of **-X** (untrusted X11 port forwarding) as this may reduce overhead as well. However, this disabled security options so it's a good idea to understand what it does before using it.

If you are displaying to Windows, the choice of X11 server software running on Windows can make a huge difference in performance. If the GUI seems very slow, try a different X11 server.

Other Options

Other options for displaying Wing remotely from Linux include:

- XRDP -- implements the protocol for Windows Remote Desktop.
- NoMachine -- Another free remote desktop toolkit.

1.12.3. Installing Extra Documentation

On Windows, Wing looks for local copies of Python documentation in the **Doc** directory of the Python installation(s), either in CHM or HTML format.

If you are using Linux or OS X, the Python manual is not included in most Python installations, so you may wish to download and install local copies.

To do this, place the top-level of the HTML formatted Python manual (where **index.html** is found) into **python-manual/#.#** within your Wing IDE installation. Replace **#.#** with the major and minor version of the corresponding Python interpreter (for example, for the Python 2.7.x manual, use **python-manual/2.7**).

Once this is done, Wing will use the local disk copy rather than going to the web when the Python Manual item is selected from the Help menu.

1.12.4. Source Code Installation

Source code is available to licensed users of Wing IDE Professional (non-evaluation licenses only) who have completed a non-disclosure agreement. Upon receipt of this agreement, you will be provided with instructions for obtaining and working with the product source code.

1.13. Removing Wing IDE

Windows

On Windows, use the Add/Remove Programs control panel, select **Wing IDE 5** and remove it.

Linux/Unix

To remove an RPM installation on Linux, type **rpm -e wingide5**.

To remove an Debian package installation on Linux, type **dpkg -r wingide5**.

To remove a tar archive installation on Linux/Unix, invoke the **wing-uninstall** script in the Wing IDE installation directory. This will automatically remove all files that appear not to have been changed since installation, It will ask whether it should remove any files that appear to be changed.

Mac OS X

To remove Wing from Mac OS X, just drag its application folder to the trash.

1.14. Command Line Usage

Whenever you run **wing5.0** from the command line, you may specify a list of files to open. These can be arbitrary text files and a project file. For example, the following will open project file **myproject.wpr** and also the three source files **mysource.py**, **README**, and **Makefile**:

```
wing5.0 mysource.py README Makefile myproject.wpr
```

(on Windows, the executable is called **wing.exe**)

Wing determines file type by extension, so position of the project file name (if any) on the command line is not important. A line number may be specified for the first file on the command line by appending **:<line-number>** to the file name (for example, **README:100** will position the cursor at the start of the README file).

The following valid options may be specified anywhere on the command line:

--prefs-file -- Add the file name following this argument to the list of preferences files that are opened by the IDE. These files are opened after the system-wide and default user preferences files, so values in them override those given in other preferences files.

--new -- By default Wing will reuse an existing running instance of Wing IDE to open files specified on the command line. This option turns off this behavior and forces creation of a new instance of Wing IDE. Note that a new instance is always created if no files are given on the command line.

--reuse -- Force Wing to reuse an existing running instance of Wing IDE even if there are no file names given on the command line. This just brings Wing to the front.

--system-qt -- *(Posix only)* This option causes Wing to try to use the system-wide install of Qt rather than its own version of Qt. Running in this mode will cause Wing to pick up on system-wide theme defaults, but may result in crashing or display problems due to incompatibilities in Qt and related libraries.

--private-qt -- *(Posix only)* This option causes Wing to use its private install of Qt rather than any system-provided Qt.

--verbose -- *(Posix only)* This option causes Wing to print verbose error reporting output to **stderr**. On Windows, run **console_wing.exe** instead for the same result.

--use-sqlite-dotfile-locking -- *(Posix only)* Use sqlite dotfile locking when opening databases which are located under the **~/.cache** directory or in the **$XDG_CACHE_DIR**.

--use-winghome -- *(For developers only)* This option sets WINGHOME to be used during this run. It is used internally and by developers contributing to Wing IDE. The directory to use follows this argument.

--use-src -- *(For developers only)* This option is used to force Wing to run from Python source files even if compiled files are present in the **bin** directory, as is the case after a distribution has been built.

--orig-python-path -- *(For developers only)* This option is used internally to indicate the original Python path in use by the user before Wing was launched. The path follows this argument.

--squelch-output -- *(For developers only)* This option prevents any output of any kind to **stdout** and **stderr**. Used on Windows to avoid console creation.

Customization

There are many ways to customize Wing IDE in order to adapt it to your needs or preferences. This chapter describes the options that are available to you.

> ***Note***
>
> **These are some of the areas of customization that are available:**
>
> - The editor can run with different personalities such as VI/Vim, Emacs, Visual Studio, Eclipse, and Brief emulation
> - The action of the tab key can be configured
> - The auto-completer's completion key(s) can be altered
> - The layout, look, color, and content of the IDE windows can be configured
> - Editor syntax colors can be configured
> - Keyboard shortcuts can be added, removed, or altered for any Wing command
> - File filters can be defined to control some of the IDE features
> - Code snippets can be defined and bound to keys
> - Perspectives can be used to save and restore user interface state
> - Scripts can be written in Python to extend the IDE's functionality
> - Many other options are available through preferences

2.1. Keyboard Personalities

The default keyboard personality for Wing implements most common keyboard equivalents found in a simple graphical text editor. This uses primarily the graphical user interface for interacting with the editor and limits use of complex keyboard-driven command interaction.

> ***Note***
>
> **Emulation of Other Editors**
>
> The first thing any Vim, Emacs, Visual Studio, Eclipse, or Brief user will want to do is to set the keyboard personality to emulate their editor of choice. This is done with the **Edit > Keyboard Personality** menu or with the **User Interface > Keyboard > Personality** preference.

Under the Vim and Emacs personalities, key strokes can be used to control most of the editor's functionality, using a textual interaction 'mini-buffer' at the bottom of the IDE window where the current line number and other informational messages are normally displayed.

Related preferences that alter keyboard behaviors include **Tab Key Action** and **Completion Keys** for the auto-completer.

It is also possible to add, alter, or remove individual keyboard command mappings within each of these personalities. See the following sub-sections for details.

2.1.1. Key Equivalents

The command a key will invoke may be modified by specifying a custom key binding. A custom key binding will override any binding for a particular key found in the keymap. Custom key bindings are set via the **Custom Key Bindings** preference.

To add a binding, click the insert button, then press the key to be bound in the **Key** field, and enter the name of the command to invoke in the **Command** field. Commands are documented in the Command Reference.

Key bindings may consist of multiple key strokes in a row, such as **Ctrl-X Ctrl-U** or **Esc X Y Z**.

If multiple comma-separated commands are specified, the key binding will execute the first available command in the listed. For example, specifying **debug-restart, debug-continue** as the command will first try to restart an existing debug session, and if no debug session exists it will start a new one.

To disable a key binding, leave the command field blank.

Some commands take arguments, which can be specified in the binding, for example by using **show-panel(panel_type="debug-probe")** or **enclose(start="(", end=")")``in the ``Command** field. Any unspecified arguments that do not have a default defined by the command will be collected from the user, either in a dialog or in the data entry area at the bottom of the IDE window.

Key bindings defined by default or overridden by this preference will be shown in any menu items that implement the same command. In cases where a command is given more than one key equivalent, only the last equivalent found will be displayed (although both bindings will work from the keyboard).

The Alt Key on OS X

On OS X, the **Alt Key** preference allows selecting one or both of the **Option** keys or the **Command** keys to act as the **Alt** key modifier for keyboard bindings.

This preference should be set by the user according to keyboard type, so that key bindings in Wing do not prevent entering characters with one or both of the **Option** keys.

For example, on German keyboards, **[**, **]**, and other common symbols are entered with the **Option** key. In those cases, setting the preference to "Left Option key" frees up the right **Option** key for entering characters and dedicates the left **Option** key to triggering key bindings.

When **Command** is used for **Alt**, the binding for **Alt-<key>** takes precedence over any binding for **Command-<key>** for the same key.

2.1.2. Key Maps

Wing ships with several key equivalency maps found at the top level of the Wing IDE installation, including **keymap.normal**, **keymap.emacs**, **keymap.vi**, and others. These are used as default key maps for the corresponding editor personalities, as set with the **User Interface > Keyboard > Keyboard Personality** preference.

For developing entirely new key bindings, or in other cases where the **Custom Key Bindings** preference is not sufficient, it is possible to create a custom key equivalency map and use it as your default map through the **Key Map File** preference.

In a key map file, each key equivalent is built from names listed in the Key Names section. These names can be combined as follows:

1. A single unmodified key is specified by its name alone, for example **'Down'** for the down arrow key.
2. Modified keys are specified by hyphenating the key names, for example **'Shift-Down'** for the down arrow key pushed while shift is held down. Multiple modifiers may also be specified, as in **'Ctrl-Shift-Down'**.
3. Special modifiers are defined for Vim mode: **Visual**, **Browse**, **Insert**, and **Replace**. These correspond with the different editor modes, and will only work if the Keyboard Personality preference has been set to **VI/Vim**.
4. Multi-key combinations can be specified by listing multiple key names separated by a space. For example, to define a key equivalent that consists of

first pushing **ctrl-x** and then pushing the **a** key by itself, use **'ctrl-x a'** as the key sequence.

The command portion of the key equivalency definition may be any of the commands listed in section Command Reference. See the examples below for usage options.

Examples

Here is an example of adding a key binding for a command. If the command already has a default key binding, both bindings will work:

```
'Ctrl-X P': 'debug-attach'
```

This example removes a key equivalent entirely:

```
'Ctrl-C Ctrl-C': None
```

These can be combined to changes the key binding for a command without retaining its default key binding:

```
'Ctrl-C Ctrl-C': None
'Ctrl-G': 'debug-continue'
```

Wing always retains only the last key binding for a given key combination. This example binds Ctrl-X to 'quit' and no other command:

```
'Ctrl-X': 'debug-stop'
'Ctrl-X': 'quit'
```

If multiple commands are specified separated by commas, Wing executes the first command that is available. For example, the following will either restart the debug process whether or not one is currently running:

```
'Ctrl-X': 'debug-restart, debug-continue'
```

Command arguments can be specified as part of the binding. Any unspecified arguments that do not have a default will be collected from the user in a dialog or in the data entry area at the bottom of the IDE window:

```
'Ctrl-X P': 'show-panel(panel_type="debug-probe")'
```

If Keyboard Personality is set to VI/Vim, modifiers corresponding to the editor modes restrict availability of the binding to only that mode:

```
'Visual-Ctrl-X': 'cut'
```

2.1.3. Key Names

Key modifiers supported by Wing IDE for key bindings are:

- **Ctrl** -- Either Control key.
- **Shift** -- Either Shift key. This modifier is ignored with some key names, as indicated below.
- **Alt** -- Either Alt key. Not recommended for general use since these bindings tend to conflict with accelerators and operating system or window manager operations. On OS X, the **Alt Key** preference is used to configure which keys invoke Alt-key bindings.
- **Command** -- Macintosh Command/Apple key. This may be mapped to other keys on other systems, but is intended for use on the Macintosh.

On Linux it is possible to remap the function of the Control, Alt, command, and windows keys. In those cases, the Ctrl and Alt modifiers will refer to the keys specified in that mapping.

Basic Keys such as the digit keys and core western alphabet keys are specified as follows:

0 1 2 3 4 5 6 7 8 9 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Most punctuation can be specified but any Shift modifier will be ignored since these keys can vary in location on different international keyboards. Allowed punctuation includes:

` ~ ! @ # $ % ^ & * ( ) - _ + = [ ] { } \ | ; : ' " / ? . > , <

Special Keys can also be used:

Escape, Space, BackSpace, Tab, Linefeed, Clear, Return, Pause, Scroll_Lock, Sys_Req, Delete, Home, Left, Up, Right, Down, Prior, Page_Up, Next, Page_Down, End, Begin, Select, Print, Execute, Insert, Undo, Redo, Menu, Find, Cancel, Help, Break, Mode_switch, script_switch, Num_Lock,

F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F11, L1, F12, L2, F13, L3, F14, L4, F15, L5, F16, L6, F17, L7, F18, L8, F19, L9, F20, L10, F21, R1, F22, R2, F23, R3, F24, R4, F25, R5, F26, R6, F27, R7, F28, R8, F29, R9, F30, R10, F31, R11, F32, R12, F33, R13, F34, R14, F35, R15,

Additional Key Names that also work but ignore the Shift modifier since they tend to appear in different locations on international keyboards:

AE, Aacute, Acircumflex, Adiaeresis, Agrave, Ampersand, Any, Apostrophe, Aring, AsciiCircum, AsciiTilde, Asterisk, At, Atilde, Backslash, Bar, BraceLeft, BraceRight, BracketLeft, BracketRight, Ccedilla, Colon, Comma, Dollar, ETH, Eacute, Ecircumflex, Ediaeresis, Egrave, Equal, Exclam, Greater, Iacute, Icircumflex, Idiaeresis, Igrave, Less, Minus, Ntilde, NumberSign, Oacute, Ocircumflex, Odiaeresis, Ograve, Ooblique, Otilde, ParenLeft, ParenRight, Percent, Period,

Plus, Question, QuoteDbl, QuoteLeft, Semicolon, Slash, Space, THORN, Uacute, Ucircumflex, Udiaeresis, Ugrave, Underscore, Yacute, acute, brokenbar, cedilla, cent, copyright, currency, degree, diaeresis, division, exclamdown, guillemotleft, guillemotright, hyphen, macron, masculine, mu, multiply, nobreakspace, notsign, onehalf, onequarter, onesuperior, ordfeminine, paragraph, periodcentered, plusminus, questiondown, registered, section, ssharp, sterling, threequarters, threesuperior, twosuperior, ydiaeresis, yen

2.2. User Interface Options

Wing provides many options for customizing the user interface to your needs. Preferences can be set to control the number and type of windows, layout of tools and editors, text fonts and colors, type of toolbar, and the overall display style (including ability to select background color).

2.2.1. Display Style and Colors

By default Wing runs with native look and feel for each OS. It is possible to override this using **Display Style** preference. The options include using the native style, using a selected style (for example Windows while running on OS X), or using a style where foreground and background color can be selected.

Color Configuration

The colors used in the user interface are selected with the **Color Palette** preference. This affects editor background color and the color of markers on text such as the selection, debug run marker, caret line highlight, bookmarks, diff/merge annotations, and other configurable colors. Palettes also define 20 additional colors that appear in preferences menus that are used for selecting colors.

The defaults set by the color palette preference can be overridden on a value by value basis in preferences. For example, the **Text Selection Color** preference is used to change the text selection color to a value other than the one specified in the selected color palette. Each such preference allows selection of a color from the current color palette, or selection of any color from a color chooser dialog.

To set also the background color of areas other than the editor, first select a **Color Palette** and then set the **Display Style** preference to **Match Palette**. The foreground and background colors for the GUI will be set by the current color palette.

The colors used for syntax highlighting code in the editor are configured separately, as described in Syntax Coloring.

Add Color Palettes

Additional color palettes can be defined and stored in the **palettes** sub-directory of the user settings directory. This directory must be created if it does not already exist. Example palettes are included in your Wing IDE installation in **resources/palettes**. After adding a palette in this way, Wing must be restarted before it is available for use.

2.2.2. Windowing Policies

Wing IDE can run in a variety of windowing modes. This is controlled by the **Windowing Policy** preference, which provides the following options:

- **Combined Toolbox and Editor Windows** -- This is the default, in which Wing opens a single window that combines the editor area with two toolbox panels.
- **Separate Toolbox Windows** -- In this mode, Wing IDE moves all the tools out to a separate shared window.
- **One Window Per Editor** -- In this mode, Wing IDE creates one top-level window for each editor that is opened. Additionally, all tools are moved out to a separate shared toolbox window and the toolbar and menu are moved out to a shared toolbar/menu window.

The windowing policy is used to describe the initial configuration and basic action of windows in the IDE. When it is changed, Wing will reconfigure your projects to match the windowing policy the first time they are used with the new setting.

However, it is possible to create additional IDE windows and to move editors and tools out to another window or among existing windows without changing from the default windowing policy. This is described below.

2.2.3. User Interface Layout

When working in the default windowing policy, Wing's main user interface area consists of two toolboxes (by default at bottom and right) and an area for source editors and integrated help.

Clicking on an already-active toolbox tab will cause Wing to minimize the entire panel so that only the toolbox tabs are visible. Clicking again will return the toolbox to its former size. The F1 and F2 keys toggle between these modes. The command **Maximize Editor Area** in the **Tools** menu (Shift-F2) can also be used to quickly hide both tool areas and toolbar.

In other windowing modes, the toolboxes and editor area are presented in separate windows but share many of the configuration options described below.

Configuring the Toolbar

Wing's toolbar can be configured by altering the size and style of the toolbar icons in the toolbar, and whether or not text is shown in addition to or instead of icons. This is controlled with the **Toolbar Icon Size** and **Toolbar Icon Style** preferences.

Alternatively, the toolbar can be hidden completely with the **Show Toolbar** preference.

Configuring the Editor Area

The options drop down menu in the top right of the editor area allows for splitting and joining the editor into multiple independent panels. These can be arranged horizontally, vertically, or any combination thereof. When multiple splits are shown, all the open files within the window are available within each split, allowing work on any combination of files and/or different parts of the same file.

The options drop down menu can also be used to change between tabbed editors and editors that show a popup menu for selecting among files (the latter can be easier to manage with large number of files) and to move editors out to a separate window or among existing windows when multiple windows are open.

Configuring Toolboxes

Each of the toolboxes can be also be split or joined into any number of sub-panels along the long axis of the toolbox by clicking on the options drop down icon in the tab area of the toolbox (right-clicking also works). The number of tool box splits Wing shows by default depends on your monitor size.

The options drop down menu can also be used to duplicate tools, or move them around among the splits or out to separate windows.

The size of each panel and the panel splits can also be altered by dragging on the dividers between them.

All available tools are enumerated in the Tools menu, which will display the most recently used tool of that type or will add one to your window at its default location, if none is already present.

Creating Additional Windows

In addition to moving existing editors or tools to new windows, it is also possible to create new tool windows (initially with a single tool) and new document windows (with editor and toolbars if applicable to the selected windowing policy) from the Windows menu.

Wing IDE will remember the state of windows as part of your project file, so the same window layout and contents will be restored in subsequent work sessions.

2.2.4. Altering Text Display

Wing tries to find display fonts appropriate for each system on which it runs, but many users will want to customize the font style and size used in the editor and other user interface areas. This can be done with the **Source Code Font/Size** and **Display Font/Size** preferences.

For information on altering colors used for syntax highlighting in the editor, see Syntax Coloring.

2.3. Preferences

Wing has many preferences that control features of the editor, unit tester, debugger, source browser, project manager, and other tools.

To alter these, use the **Preferences** item in the **Edit** menu (or **Wing IDE** menu on OS X). This organizes all available preferences by category and provides access to documentation in tooltips that are displayed when mousing over the label area to the left of each preference. Any non-default values that are selected through the **Preferences Dialog** are stored in the user's preferences file, which is located in the User Settings Directory.

All preferences are documented in the Preferences Reference.

2.3.1. Preferences File Layers

Wing's preferences manager runs on a layered set of preferences files, as follows:

1. For each preference, Wing defines a hardwired default internally.
2. An installation-wide preferences file may be placed inside the Wing IDE installation directory (or on OS X the **Contents/Resources** directory within Wing's **.app** folder).
3. An individual user preferences file is stored in the User Settings Directory.
4. Additional preferences files may be specified on the command line with one or more **--prefs-file** options. For example:

```
wing5.0 --prefs-file /path/to/myprefs
```

The values given in later files in this list override values found in earlier ones. For example, the user-specific preferences file take precedence over any values in the **WINGHOME/preferences** file, and a file specified with **--prefs-file** would override values in the user-specific preferences file.

When preferences are changed, Wing writes the changes to the lowest file present on the above list, either the last file specified with **--prefs-file** or the preferences file in the User Settings Directory. Wing will never modify the installation-wide preferences file.

If a preference is set to a default value, as obtained from the preceding files in the above list, then Wing removes the value from the writeable preferences file. This means that the effective value of a preference can change in later IDE sessions even if the last file on the list above is unchanged. This is by design to allow inheriting centrally managed default values.

2.3.2. Preferences File Format

While we recommend using the **preferences GUI** to alter preferences, some users may wish to edit the underlying text files manually.

The preferences file format consists of a series of sections separated by bracketed headers such as **[user-preferences]**. These headers are used internally to identify from which file a value was read, when there are multiple preferences files active.

The body of each section is a sequence of lines, each of which is a **name=value** pair. All of these are read in from each preferences file, with later like-named settings overwriting earlier ones.

Each preference name is in *domain.preference* form, where *domain* is the IDE subsystem affected and *preference* is the name of the specific preference (for example, **edit.tab-size** defines the source editor's tab size).

Preference values can be any Python expression that will evaluate to a number, string, tuple, list, or dictionary (the data type is defined by each preference and will be verified as the file is read into Wing). Long lines may be continued by placing a backslash (****) at the end of a line and comments may be placed anywhere on a line by starting them with **#**.

If you wish to write preferences files by hand, refer to the Preferences Reference for documentation of all available preferences.

2.4. Syntax Coloring

There are two ways to configure syntax highlighting in Wing: Minor adjustments can be made in preferences, and comprehensive configuration can be achieved by creating a syntax color specification file.

Minor Adjustments

For minor tweaks to syntax coloring in the editor, use **Syntax Formatting** in the **Edit > Syntax Coloring** preference group. For each supported file type, and each lexical state for the file type, it is possible to set the foreground and background colors, to use bold or italic font, and to fill the end of line character so it appears as a solid block of color.

Comprehensive Changes

For more comprehensive changes to syntax coloring, textual syntax coloring specifications can be placed into the **syntax** directory within the User Settings Directory. This directory must be created if it is not already present.

To override syntax colors only for a particular **Color Palette**, place the syntax file in a sub-directory of the **syntax** directory whose name matches the palette specification file name. For example, use **syntax/black-background/python.stx** to specify colors to use in Python files only with the Black Background color palette.

The syntax coloring configuration files can be modeled on the system-wide defaults, which are stored in **resources/syntax** within your Wing IDE installation directory. Any values missing from these files cause Wing to fall back to the system-wide defaults. Wing must be restarted to pick up changes made in these files.

Note that any non-default syntax coloring preferences will take precedence over syntax files found in the user settings directory or system-wide.

Automatic Color Adjustment

If the currently selected **Color Palette** uses a non-white background for the editor, or if the **Background Color** in the **Editor > Syntax Coloring** preference group is set to a color other than white, then Wing will automatically adjust all configured foreground colors when necessary to ensure that the text remains visible. This avoids the need to create completely new color configurations for different editor background colors.

This feature is disabled when using a palette-specific syntax configuration file, as describe above, since in that case the colors are being designed for a specific background color.

2.5. Perspectives

Wing IDE Professional allows you to create and switch between subsets of the IDE's tools, as appropriate for particular kinds of work, such as editing, testing, debugging, working on documentation, and so forth.

These subsets, or perspectives, are named and then accessed from the **Tools** menu, which provides a sub-menu for switching between them. The current perspective is shown in brackets in the lower left of Wing's window.

Perspective Manager

The **Tools** menu also contains an item for displaying the **Perspective Manager**. The Perspective Manager shows the name of each perspective, whether or not the perspective is shared, whether or not the perspective is auto-saved, the perspective style, and the key binding (if any) that is assigned to it.

The name of a perspective can be changed by clicking on the name within the list and editing it in place.

When perspectives are shared, they are stored in the shared perspectives file, which is configured with the **Shared Perspective File** preference, instead of in the project file. This makes the shared perspectives available across all projects, or potentially to multiple users. When multiple instances of Wing share this file, Wing will watch for changes and auto-reload the set of perspectives into each instance of Wing, as another instance makes changes. Note that when a shared perspective is un-shared, it is moved into the project currently open in the instance of Wing that un-shared it.

When the **Auto-save Perspectives** is set to **Configured by Perspective**, the Perspective Manager will include a column to specify whether the perspective should be auto-saved before transitioning to another perspective. This is described in more detail below.

The perspective style can be used to control how much state is stored in the perspective: By default Wing stores only the overall layout of the GUI and set of tools present. Setting this to "Tools and Editors" will cause the perspective to control also which editors are open. Setting it to "All Visual State" will store also the detailed state of the tools and editors, including scroll position, selection, search strings, tree expansion states, and so forth.

When a key binding is defined, that key sequence will cause Wing to switch to the associated perspective.

Perspective Manager Context Menu

The Perspective Manager provides the following functionality in its context (right-click) menu:

- **New** creates a new untitled perspective with the current state of the application.
- **Duplicate** makes a copy of the selected perspective, including its stored application state.
- **Delete** removes the selected perspective.
- **Set Key Binding** displays a dialog in which the key binding desired for the perspective can be typed. This key sequence will cause Wing to switch to that perspective.
- **Update with Current State** replaces the stored state for the selected perspective with the current application state.
- **Restore Saved State** loads the state stored in the selected perspective without making that perspective current.

Preferences

The Perspective Manager's **Configure** button displays the preferences that control how perspectives work. These include:

- **Auto-save Perspectives** -- Selects when the current GUI state should be auto-saved into a perspective before switching to another perspective. **Always** will always auto-save all perspectives, **Never** disables auto-save entirely, **Prompt** causes Wing to prompt each time when leaving a perspective, and **Configured by Perspective** allows the behavior to be controlled for each perspective, in the Manage Perspectives dialog. The default is **Always** so that the last application state is always restored when returning to the perspective. Disabling auto-save can be useful for perspectives that should always start with a previously stored fixed state.
- **Shared Perspective File** -- This is used to specify where shared perspectives are stored on disk. The default is a file **perspectives** in the User Settings Directory.

Auto-Perspectives

Auto-perspectives can be used to automatically switch between the built-in perspectives **edit** and **debug** when debugging is started and stopped. When this is enabled, Wing by default will show fewer tools when editing and most of the debugging tools only while debugging. If the user alters which tools are shown from the defaults, this will be remembered the next time debug is started or stopped.

Auto-perspectives are off by default and can be turned on with the **Automatic Perspectives** attribute under the **Debug** tab in **Project Properties**.

Once this is enabled, Wing will save the unnamed pre-existing perspective as **user** and will display the appropriate perspective **edit** or **debug** with its default tool set. Note that the perspectives **edit** and **debug** are not created until the first time debugging is started. After that, they appear in the **Goto Perspective** sub-menu in the **Tools** menu and in the perspective manager.

Restoring Default Toolset

In Wing IDE Pro, the **Tools** menu item **Restore Default Toolset** will restore the tools appropriate for the current perspective. If this is any of the built-in perspectives **edit**, **debug**, or **diff** and the **Automatic Perspectives** preference is turned on, then the tool set will differ from that which is used for user-defined perspectives or when automatic perspectives are disabled.

2.6. File Filters

Wing allows you to define file filters that can be used in various ways within the IDE, such as for searching particular batches of files and adding only certain kinds of files to a project.

To view or alter the defined file filters, use **File Filters** in the **Files > File Types** preferences group.

When adding or editing a filter, the following information may be entered:

- **Name** -- The name of the filter
- **Includes** -- A list of inclusion criteria, each of which contains a type and a specification. A file will be included by the filter if any one of these include criteria matches the file.
- **Excludes** -- A list of exclusion criteria, any of which can match to cause a file to be excluded by the filter even if one or more includes also matched.

The following types of include and exclude criteria are supported:

- **Wildcard on Filename** -- The specification in this case is a wildcard that must match the file name. The wildcards supported are those provided by Python's fnmatch module.
- **Wildcard on Directory Name** -- The specification in this case is a wildcard that must match the directory name.
- **Mime Type** -- The specification in this case names a MIME type supported by Wing IDE. If additional file extensions need to be mapped to a MIME type, use the **Extra File Types** preference to define them.

Once defined, filters are presented by name in the Search in Files tool's **Filter** menu, and in the Project tool's **Directory Properties**.

Any problems encountered in using the file filters are reported in the Messages area.

Project Manager

The **Project manager** provides a convenient index of the files in your software project and collects information needed by Wing's debugger, source code analysis tools, version control integration, and other facilities.

To get the most out of Wing's debugger and source analysis engine, you may in some cases need to set up **Python Executable**, **Python Path**, and other values in Project-Wide Properties and/or Per-File Properties.

3.1. Creating a Project

To create a new project, use the **New Project** item in the **Project** menu. This will prompt you to save any changes to your currently open project and will create a new untitled project.

When you create a new project, you will often want to alter some of the Project Properties to point Wing at the version of Python you want to use, set

PYTHONPATH so Wing's source analyzer and debugger can find your files, and set any other necessary runtime environment for your code.

To add files to your project, use the following items in the Project menu:

- **Add Existing Directory** allows you to specify a directory to include in the project. In many cases, this is the only operation needed to set up a new project, and it is the recommended approach. You will be able to specify a filter of which files to include, whether to include hidden & temporary files, and whether to include subdirectories. The list of files in the project will be updated as files matching the criteria are added and removed from the disk.
- **Add Current File** will add the current editor file to the project if it is not already there.
- **Add Existing File** will prompt you to select a single file to add to the project view. This may also result in adding a new directory to the project manager window, if that file is the first to be added for a directory.
- **Add New File** is used to create a new file and simultaneously add it to your project.

A subset of these options can be accessed from the context menu that appears when right-clicking your mouse on the surface of the project manager window.

3.2. Removing Files and Directories

To remove a specific file or directory, select it and use the Remove From Project menu item in the right-click context menu from the surface of the Project Manager window, or by selecting an item on the project and using Remove Selected Entry in the Project menu.

If the removed file or directory is part of another directory that has been added to the project, the removal is remembered as an exclusion that can be cleared from **Directory Properties**, which are accessed by right clicking on the parent directory in the Project tool.

3.3. Saving the Project

To save a new project, use **Save Project As** in the **Project** menu. Once a project file has been saved the first time, it will be auto-saved whenever you close the project, start a debug session, or exit Wing.

You can also save a copy of your project to another location or name with **Save Project As...** in the **Project** menu.

> ***Note***
>
> **Moving Project Files**
>
> When moving a project file on disk, doing so in a file browser or from the command line may partially break the project if it is moved relative to the position of files that it includes. Using **Save Project As...** in Wing instead will properly update the relative paths that the project manager uses to locate files in the project.

3.4. Sorting the View

The project can be set to show your files in one of several modes, using the **Options** menu in the top right of the project view:

- **View As Tree** -- This displays the project files in true tree form. The tree structure is based on the partial relative path from the project file.
- **View As Flattened Tree** -- This view (the default) shows files organized according to their location on disk. Each directory is shown at the top level with path names shown as partial relative paths based on the location of the project file. If you alter the location of the project file with **Save Project As...**, these paths will be updated accordingly.

Several sorting options are available to sort items within their directory by name, mime type, or extension. The **List Files Before Directories** option may be used to control whether files or directories are shown first in the tree view.

3.5. Navigating to Files

Files can be opened from the project manager window by double clicking or middle clicking on the file name, or right-clicking and using the Open in Wing IDE menu item.

Files may also be opened using an external viewer or editor by right-clicking on the file and using the Open in External Viewer item. On Windows and Mac OS X, this opens the file as if you had double clicked on it. On Linux, the preferences **File Display Commands** and **Extra Mime Types** can be used to configure how files are opened.

You can also execute Makefiles, Python source code, and any executable files by selecting the **Execute Selected** item from the popup menu. This executes outside of the debugger with any input/output occurring in the OS Commands tool. Doing so also adds the command to the OS Commands tool, where its runtime environment can be configured.

3.5.1. Keyboard Navigation

Once it has the focus, the project manager tree view is navigable with the keyboard, using the up/down arrow keys, page up and page down, and home/end.

Use the right arrow key on a parent to display its children, or the left arrow key to hide them.

Whenever a file is selected, pressing enter will open that item into an editor in Wing IDE.

3.6. Sharing Projects

The default **Project Type**, accessed from **Project Properties > Options** in the **Project** menu, is **Shared (Two Files)**. This stores sharable project data in a file with extension **.wpr** and user-specific project data in a file with extension **.wpu**. Only the **.wpr** file should be checked into revision control or shared with other users or machines. This file is designed to work across all supported OSes and avoids storing values that are likely to be user-specific.

If the project type is changed to **Single User (One File)** only a single **.wpr** file will be written, with all project data, and the **.wpu** file will be removed from disk.

Only **Single User (One File)** projects can be read by Wing IDE Personal.

Making Project Files More Sharable

In most cases sharing the ***.wpr** file will just work. File paths are stored relative to the project's location on disk, so they will work in different configurations.

If revision control conflicts do arise among different users of a **.wpr** file, environment variables can be used to make the project work for all users and on all machines. The environment variable values can either be inherited from outside of Wing or set using **Environment** in **Project Properties**. The values for the **Environment** property are stored in the **.wpu** file and thus may vary by user.

File Format

Both the **.wpr** and **.wpu** files use the same textual file format that is used for the preferences file. See section Preferences File Format for more information on the format itself.

Changing Which Attributes are Shared

Which project properties are stored in the main project file may be set by modifying the **.wpr** file with a text editor and setting the **proj.shared-attribute-names** attribute to a list of attribute names to add or remove from the default set. Each entry in this list is an attribute name preceded by **-** to move a shared attribute to the non-shared file, or **+** to move a non-shared attribute to the shared file. This specification is applied to the default set of shared attributes in order to determine which attributes to share in this project.

The following example would move the commands defined in the OS Commands tool into the user-specific file and would share the Python Executable and Python defined in Project Properties in the **.wpr** file:

```
proj.shared-attribute-names = [
  '-console.toolbox',
  '+proj.pyexec',
  '+proj.pypath',
]
```

Note that sharing the Python Executable and Python Path works only if the value are valid on all the machines where the project is used. This can be easier to achieve if the values use environment variable references such as **${WING:PROJECT_DIR}/a/b/c** for a path entry.

The default set of shared attributes is:

```
proj.shared-attribute-names
proj.directory-list
proj.file-list
proj.file-type
proj.main-file
proj.home-dir
testing.test-file-list
testing.auto-test-file-specs
testing.test-framework
debug.named-entry-points
proj.launch-config
debug.launch-configs
console.toolbox
```

Note that only non-empty and non-default values are stored in the project file. For example, **proj.file-list** will be missing if no files are individually added to the project.

The names of other potentially sharable attributes can be found in the **.wpu** file.

3.7. Project-wide Properties

Each project has a set of top-level properties that can be accessed and edited via the **Properties** item in the Project menu. These can be used to configure the Python environment used when debugging, executing, or testing code, and for the source code analysis engine, which drives Wing's auto completion, source index, and other capabilities. Project properties are also provided to set options for the project and to enable and configure extensions for revision control, Zope, and other tools.

Any string value for a property may contain environment and special variable references, as described in Environment Variable Expansion.

Environment

To get the most out of Wing, it is important to set these values in the Environment tab correctly for your project:

Python Executable -- When the **Custom** radio button is checked and the entered field is non-blank, this can be used to set the full path to the Python executable that should be used when debugging source code in this project. When **Use default** is selected, Wing tries to use the default Python obtained by typing **python** on the command line. On OS X, Wing prefers the latest Apple-provided Python. If this fails, Wing will search for Python in **/usr/local** and **/usr** (on Linux and OS X) or in the registry (on Windows). To use Wing with **virtualenv** just set this property to the Python executable provided by **virtualenv**. An easy way to get the full path to use here is to type the following in the Python that you wish to use: **import sys; print(sys.executable)**. This can also be typed into the IDLE that is associated with the Python installation.

Python Path -- The **PYTHONPATH** is used by Python to locate modules that are imported at runtime with the **import** statement. When the **Use default** checkbox in this area is checked, the inherited **PYTHONPATH** environment variable is used for debug sessions. Otherwise, when **Custom** is selected, the specified **PYTHONPATH** is used.

Environment -- This is used to specify values that should be added, modified, or removed from the environment that is inherited by debug processes started from Wing IDE and is used to expand environment variable references used in other properties. Each entry is in **var=value** form and must be specified one per line in the provided entry area. An entry in the form **var=** (without a value) will remove the given variable so it is undefined. Note that you are operating on the environment inherited by the IDE when it started and not modifying an empty environment. When the *Use inherited environment* choice is set, any entered values are ignored and the inherited environment is used without changes.

Debug

The following properties are defined in the Debug tab:

Main Entry Point -- This defines where execution starts when the debugger is launched from the IDE. The default is to start debugging in the current editor file. Alternatively, use this property to define a project-wide main entry point so that debug always started in that file regardless of which file is current in the editor. The entry point can either be a selected file in order to debug that files with the environment specified in its File Properties, or a named entry point to select a file and use a different environment to debug it.

Initial Directory -- When the **Use default** radio button is checked, the initial working directory set for each debug session will be the directory where the debugged file is located. When **Custom** is selected, the specified directory is used

instead (use **$(WING:PROJECT_DIR)** for the project's directory). This property also sets the initial directory for the Python Shell, determines how Wing resolves partial paths on the Python Path for the purposes of static analysis, and is used for other features in the IDE that require a starting directory for a sub-process. For these, Wing will use the directory of the main debug file in the project as the default initial directory, or the directory of the project file if there is no main debug file defined.

Build Command -- This command will be executed before starting debug on any source in this project. This is useful to make sure that C/C++ extension modules are built, for example in conjunction with an external **Makefile** or **distutils** script, before execution is started. The build is configured through and takes place in the OS Commands tool.

Python Options -- This is used to select the command line options sent to the Python interpreter while debugging. The default of **-u** sets Python into unbuffered I/O mode, which ensures that the debug process output, including prompts shown for keyboard input, will appear in a timely fashion.

Debug Server Port -- This can be used to alter the TCP/IP port on which the debugger listens, on a per-project basis. In this way, multiple instances of Wing using different projects can concurrently accept externally initiated debug connections. See Advanced Debugging Topics for details.

Automatic Perspectives -- When enabled, Wing will create and automatically switch between **Edit** and **Debug** perspectives when debugging is stopped and started. See Perspectives for details.

Options

These project options are provided:

Project Type -- This can be used to select whether or not the project will be shared among several developers. When shared, the project will be written to two files, one of which can be shared with other developers. See Project Types for details.

Default Encoding sets the default text encoding to use for files when the encoding cannot be determined from the contents of the file. This applies to all files opened when the project is open, whether or not they are part of the project. By default, this falls back to the value set by the **Default Encoding** preference.

Project Home Directory sets the base directory for the project. This overrides the project file location as the directory on which to base relative paths shown in the Project view and elsewhere. It is also used as the directory in which the Python Shell subprocess is launched and for the starting directory when the **Default Directory Policy** preference is set to **Current Project**.

Preferred Line Ending and **Line Ending Policy** control whether or not the project prefers a particular line ending style (line feed, carriage return, or carriage return + line feed), and how to enforce that style, if at all. By default, projects do not enforce a line ending style but rather insert new lines to match any existing line endings in the file.

Preferred Indent Style and **Indent Style Policy** control whether or not the project prefers a particular type of indentation style for files (spaces only, tabs only, or mixed tabs and spaces), and how to enforce that style, if at all. By default, projects do not enforce an indent style but rather insert new lines to match any existing indentation in the file.

Strip Trailing Whitespace controls whether or not to automatically remove whitespace at the ends of lines when saving a file to disk.

Extensions

The Extensions tab of Project Properties is used to control add-ons on a per-project basis:

Enable Django Template Debugging enables Django-specific functionality that makes it possible for Wing's debugger to stop at breakpoints and step through Django template files.

Matplotlib Event Loop Support enabled Matplotlib-specific functionality that updates plots continuously when working interactively in the Python Shell.

Enable Zope2/Plone Support, **Zope2 Instance Home**, and **Zope2 Host** enable legacy support for older Zope installations. They are needed because Zope 2.x implements import magic that works differently from Python's default **import** and thus adding the instance home directory to **PYTHONPATH** is not sufficient. Wing's source analyzer needs this extra clue to properly find and process the Zope instance-specific sources.

When this option is activated, Wing will also offer to add the relevant Zope2/Plone files to the project, and to install the control panel for configuring and initiating debug in Zope2/Plone. See the Zope How-To for details.

Testing

Test File Pattern can be used to specify which files in the project should be shown in the Testing tool. See the Testing chapter for details.

Default Test Framework defines the testing framework to use by default, unless another is chosen using File Properties on the test file.

Environment can be used to select environment for running unit tests that differs from the Project-wide settings, and for setting any command line arguments to send to unit tests.

3.7.1. Environment Variable Expansion

Any string value for a property may contain environment variable references using the **$(name)** or **$ {name}** notation. These will be replaced with the value of the environment variable when used by the IDE. If the environment variable is not set, the reference will be replaced by an empty string. The system environment, as modified by the project-wide or per-file environment property (if defined), is used to expand variable references.

Special Environment Variables

The following special variable names are defined by Wing IDE for use in the **$(name)** or **${name}** form:

- **WING:FILENAME** -- full path of current file
- **WING:FILENAME_DIR** -- full path of the directory containing the current file
- **WING:LINENO** -- current line number in the current file
- **WING:SCOPE** -- x.y.z-formatted name of the current scope in the current file (if Python)
- **WING:PROJECT** full path of current project (including the project file name)
- **WING:PROJECT_DIR** -- full path of the directory containing the current project
- **WING:PROJECT_HOME** -- full path of the **Project Home** directory, as set in **Project Properties** (by default this is the same as WING:PROJECT_DIR)
- **WING:SELECTION** -- the text selected on the current editor, if any

These may evaluate to an empty string when there is no current file name.

3.8. Per-file Properties

Per-file properties can be set by right-clicking on a source file and selecting the **Properties** menu item in the popup, by right-clicking on a file in the project view and selecting **File Properties**, or by opening a file and using the **Current File Properties...** item in the Source menu. For Debug and Python Settings, values entered here will override any corresponding project-wide values when the selected file is the current file or the main entry point for debugging.

Any string value for a property may contain environment and special variable references, as described in Environment Variable Expansion.

File Attributes

File Type -- This property specifies the file type for a given file, overriding the type determined automatically from its file extension and/or content. This property is recommended only when the **Extra File Types** preference cannot be used to specify encoding based on filename extension.

Encoding -- This can be used to specify the encoding with which a file will be saved. When it is altered for an already-open file, Wing will offer to reload the file

using the new encoding, to only save subsequently using the new encoding, or to cancel the change. Choose to reload if the file was opened with the wrong encoding. For already-open files, the encoding attribute change is only saved if the file is saved. If it is closed without saving, the encoding attribute will revert to its previous setting. The encoding cannot be altered with this property if it is being defined by an encoding comment in a Python, HTML, XML, or gettext PO file. In this case, the file should be opened and the encoding comment changed. Wing will save the file under the newly specified encoding.

Important: Files saved under a different encoding without an encoding comment may not be readable by other editors because there is no way for them to determine the file's encoding if it differs from the system or disk default. Wing stores the selected encoding in the project file, but no mark is written in the file except for those encodings that naturally use a Byte Order Mark (BOM), such as utf_16_le, utf_16_be, utf_32_le, or utf_32_be. Note that standard builds of CPython cannot read source files encoded in utf16 or utf32.

Line Ending Style -- Specifies which type of line ending (line feed, carriage return, or carriage return and line feed) is used in the file. When altered, the file will be opened and changed in an editor. The change does not take effect until the file is saved to disk.

Indent Style -- This property can be used in non-Python files to change the type of indent entered into the file for newly added lines. For Python files, the only way to alter indentation in a file is with the **Indentation manager**.

Read-only on Disk -- This property reflects whether or not the file is marked read-only on disk. Altering it will change the file's disk protections for the owner of the file (on Posix, group/world permissions are never altered).

Editor

These properties define how the file is displayed in the editor:

Show Whitespace -- This allows overriding the **Show White Space** preference on a per-file basis.

Show EOL -- This allows overriding the **Show EOL** preference on a per-file basis.

Show Indent Guides -- This allows overriding the **Show Indent Guides** preference on a per-file basis.

Ignore Indent Errors -- Wing normally reports potentially serious indentation inconsistency in Python files. This property can be used to disable this check on a per-file basis (it is also available in the warning dialog).

Ignore EOL Errors -- When the project's **Line Ending Policy** is set to warn about line ending mismatches, this property can be used to disable warnings for a particular file.

Debug/Execute

This tab is used to select the environment used when debugging or executing the file, and to set run arguments. By default, the project-wide environment will be used with the specified run arguments. Alternatively, use the **Environment** property to select a different environment defined by a launch configuration.

Testing

The testing tab contains a subset of the fields described in Project-Wide Properties.

3.9. Launch Configurations

Most users of Wing IDE will use the project-wide environment for debugging, executing, and testing code.

In some cases, multiple environments are needed in the same project, for example to run unit tests in different environments, or to set a different environment for specific entry points. To support this, launch configurations can be managed from the **Project > Launch Configurations** menu item. This manager is initially empty. Right click to create, edit, duplicate, or delete launch configurations.

Once defined, launch configurations can be referenced from per-file properties under the **Debug/Execute** and **Testing** tabs, and in the creation of named entry points.

Launch configurations contain the following properties, as organized under the Environment and Python tabs in the launch configuration properties dialog:

Run Arguments -- Enter any run arguments here. Wing does not interpret backslashes (\) on the command line and passes them unchanged to the sub-process. The only exceptions to this rule are **\'** and **\"** (backslash followed by single or double quote), which allow inclusion of quotes inside quoted multi-word arguments.

Initial Directory -- Specifies the initial working directory. By default this uses the initial directory specified by the project-wide environment. When the **Use default** radio button is checked, the initial working directory will instead be the directory where the launched file is located. When **Custom** is selected, the specified directory is used instead (use **$(WING:PROJECT_DIR)** for the project's directory).

Environment -- This is used to specify values that should be added, modified, or removed from the environment. Each entry is in **var=value** form and must be specified one per line in the provided entry area. An entry in the form **var=** (without a value) will remove the given variable so it is undefined. The popup selector is used to choose the environment to modify: Either the startup environment seen when Wing IDE was first started, or the Project-defined environment. When *Use*

project values or *Use inherited environment* is chosen, any entered values are ignored and the selected environment is used without changes.

Build Command -- This command will be executed before launching a subprocess with this launch configurations. This is useful to make sure that C/C++ extension modules are built, for example in conjunction with an external **Makefile** or **distutils** script, before execution is started. The build is configured through and takes place in the OS Commands tool.

Python Executable -- When the **Custom** radio button is checked and the entered field is non-blank, this can be used to set the full path to the Python executable that should be used when debugging source code in this project. When **Use default** is selected, Wing uses the project configuration.

Python Path -- The **PYTHONPATH** is used by Python to locate modules that are imported at runtime with the **import** statement. By default this uses the project setting. When the **Use default** checkbox selected, the inherited **PYTHONPATH** environment variable is used. Otherwise, when **Custom** is selected, the specified **PYTHONPATH** is used.

Python Options -- This is used to select the command line options sent to the Python interpreter while debugging. The default of **-u** sets Python into unbuffered I/O mode, which ensures that the debug process output, including prompts shown for keyboard input, will appear in a timely fashion.

For all of these, environment variable references may be used, as described in Environment Variable Expansion.

Shared Launch Configurations

By default each launch configuration is stored in the project file. In the launch configuration manager dialog, the **Shared** checkbox can be selected to cause Wing to store that launch configuration in the User Settings Directory instead, in a file named **launch**. Those launch configurations are then accessible from all projects.

Working on Different Machines or OSes

When the **Shared** checkbox is selected for a launch configuration, or when shared projects are used, it is necessary to design launch configurations carefully so that they will work across projects, machines, or operating systems.

For example, specifying a full path in the Python Path may not work on a different OS. The key to making this work is to use environment variable references in the form **${VARNAME}** as described in Environment Variable Expansion. The referenced environment variables can be special environment variables defined by Wing, such as **WING:PROJECT_DIR**, or user-defined values that are set either system-wide, or in Project Properties. Note that values set in **Environment** in

Project Properties are by default not stored in the shared project file, so those may vary on each development machine if desired.

A common example in configuring Python Path is to replace a full path like **/Users/myname/src/project/src** with **${WING:PROJECT_DIR}/src** (this assumes you store the project in **/Users/myname/src/project**). In general, working of the project's location is a good approach to maintaining some independence from differences on different development machines and OSes.

To make file paths work across OSes, use forward slashes instead of back slashes. The character sequence **..** can be used to move up a directory on all OSes, as for example in **{WING:PROJECT_DIR}/../libs/src**.

Source Code Editor

Wing IDE's source code editor is designed to make it easier to adopt the IDE even if you are used to other editors.

> ***Note***
>
> **Key things to know about the editor**
>
> - The editor has personalities that emulate other commonly used editors such as Visual Studio, VI/Vim, Emacs, and Brief.
> - Context-appropriate auto-completion, goto-definition, and code index menus are available when working in Python code
> - The editor supports a wide variety of file types for syntax colorization.
> - Key mappings and many other behaviors are configurable.
> - The editor supports folding for some file types

4.1. Syntax Colorization

The editor will attempt to colorize documents according to their MIME type, which is determined by the file extension, or content. For example, any file ending in **.py** will be colorized as a Python source code document. Any file whose MIME type cannot be determined will display all text in black normal font by default.

All the available colorization document types are listed in the File Properties dialog's File Attributes tab. If you have a file that is not being recognized automatically, you can use the **File Type** menu found there to alter the way the file is being displayed. Your selections from this menu are stored in your project file, so changes made are permanent in the context of that project.

If you have many files with an unrecognized extension, use the **Extra File Types** preference to add your extension.

Syntax coloring can be configured as described in the section Syntax Coloring.

4.2. Right-click Editor Menu

Right-clicking on the surface of the editor will display a context menu with commonly used commands such as Copy, Paste, Goto Definition, and commenting and indentation operations.

When revision control is enabled in Project Properties under the Extensions tab, the menu is populated with additional items for the selected revision control system.

User-defined scripts may also add items here, as described in the Scripting chapter.

4.3. Navigating Source

The set of menus at the top of the editor can be used to navigate through your source code. Each menu indicates the scope of the current cursor selection in the file and may be used to navigate within the top-level scope, or within sub-scopes when they exist.

When editor tabs are hidden by clicking on the options drop down in the top right of the editor area, the left-most of these menus lists the currently open files by name.

You can use the **Goto Definition** menu item in the editor context menu to click on a construct in your source and zoom to its point of definition. Alternatively, place the cursor or selection on a symbol and use the **Goto Selected Symbol Defn** item in the **Source** menu, or its keyboard equivalent. Control-Click (and Command-Click on OS X) also jumps to the point of definition.

To visit all points of use of a symbol, right click on it and select **Find Points of Use** or use the item of the same name in the **Source** menu. The points of use are shown in the Uses tool. Clicking on items in the list visits that use. Alt-Clicking on a symbol also displays points of use.

When moving around source, the history buttons in the top left of the editor area can be used to move forward and backward through visited files and locations within a file in a manner similar to the forward and back buttons in a web browser.

Other commonly used ways to navigate to files that are open include the **Window** menu, which lists all open files; the **Open Files** tool which also supports defining named file sets; the **Recent** sub-menu in the **File** menu; **Open From Project** in the **File** menu, which finds project files quickly by typing a fragment of the file name; and **Open From Keyboard** in the **File** menu, which operates in a temporary input

area at the bottom of the IDE window and offers auto-completion of file names as you type.

Find Symbol in the **Source** menu provides a way to find a symbol defined in the current Python scope by typing a fragment of its name.

See also the Find Uses tool and Source Browser.

4.4. File status and read-only files

The editor tabs, or editor selection menu when the tabs are hidden, indicate the status of the file by appending ***** when the file has been edited or **(r/o)** when the file is read-only. This information is mirrored for the current file in the status area at the bottom left of each editor window. Edited status is also shown in the **Window** menu by appending ***** to the file names found there.

Files that are read-only on disk are initially opened within a read-only editor. Use the file's context menu (right-click) to toggle between read-only and writable state. This alters both the editability of the editor and the writability of the disk file so may fail if you do not have the necessary access permissions to make this change.

4.5. Transient, Sticky, and Locked Editors

Wing can open files in several modes that control how and when files are closed:

Transient Mode -- Files opened when searching, debugging, navigating to point of definition or point of use, and using the Project or Source Browser tools with the **Follow Selection** checkbox enabled are opened in transient mode and will be automatically closed when hidden.

The maximum number of non-visible transient files to keep open at any given time can be set with the **Editor / Advanced / Maximum Non-Sticky Editors** preference.

Sticky Mode -- Files opened from the File menu, from the keyboard file selector, or by double clicking on items in the Project or Source Browser tools will be opened in sticky mode, and are kept open until they are explicitly closed.

Locked Mode -- When multiple splits are visible, a third mode is available where the file is locked into the editor. In this case, the editor split is not reused to display any newly opened or visited files unless no unlocked splits are present.

A file can be switched between these modes by clicking on the stick pin icon in the upper right of the editor area.

Right-click on the stick pin icon to navigate to files that were recently visited in the associated editor or editor split. Blue items in the menu were visited in transient state and black items were sticky. Note that this differs from the Recent area in the File menu, which lists only sticky file visits and includes visits for all editors and editor splits.

Transient files that are edited are also automatically converted to sticky mode.

4.6. Auto-completion

Wing Personal and Professional display an auto-completer in the editor and shells. When the completer appears, type until the correct symbol is highlighted in the list, or use the up/down arrow keys, and then press the Tab key or double click on an item. Wing will fill in the remaining characters for the source symbol, correcting any spelling errors you might have made in the name.

To alter which keys cause auto-completion to occur, use the **Auto-completion Keys** preference. Ctrl-click on the list to select multiple keys. For printable keys such as '.', '(', '[', and ':' the key will be added to the editor and any relevant auto-editing operations will be applied. For '.' the completer will be shown again for the attributes of the completed symbol.

To cancel out of the auto-completion popup, press the **Esc** key or **Ctrl-G**. The auto-completer will also disappear when you exit the source symbol (for example, by pushing **space** or any other character that isn't a completion key and can't be contained in a source symbol), if you click elsewhere on the surface of the source code, or if you issue other keyboard-bound commands that are not accepted by the auto-completer (for example, **save** through keyboard equivalent).

The completer can be configured to display immediately, only after a specified number of characters, or after a time delay. Completion may be case sensitive or insensitive and the completer may be auto-hidden after a specified timeout. These and other configuration options are in the **Auto-completion preferences group**.

Auto-Completer Icons

The auto-completer contains two columns of icons that indicate the type and origin of the symbol. The first column may contain one of the following icons:

	A Python builtin
	A snippet defined in the Snippets tool
	An argument for the current function or method scope
	The symbol was found by introspecting the live runtime state

The second column of icons may contain one of the following icons:

	A Python keyword
	A module name
	A class name
	A Python package (a directory with __init__.py in it)
	A method name

	A function name
	An object instance (other than the basic types below)
	A dictionary
	A tuple
	A list
	A string
	An integer
	A float
	An exception
	A Python stack frame

Additionally, icons in the second column may be annotated as in the following examples (the annotation may be applied to any of the above):

	An upward pointing arrow indicates that the symbol was inherited from a superclass
	A leftward pointing arrow indicates that the symbol was imported with "from x import" style import statement

Turbo Completion Mode for Python (Experimental)

When the **Python Turbo Mode** preference is enabled, Wing will use a different completion mode for Python files and in the shells. This treats any non-word key as being a completion key, in a context appropriate way. **Ctrl**, **Alt**, and **Command** act as cancel keys, in addition to **Esc**.

This mode can be considerably faster to use when the completer contains the desired text. Once the correct completion is selected in the completer, the next source code character can immediately be typed. The completion will be placed, the next key will be entered into the editor, any relevant auto-editing operations will be applied, and the completer shown again if appropriate.

In contexts where a new symbol is being defined, Wing disables Turbo mode depending on the character being pressed. For example, pressing **=** after a name at the start of a line, entering an argument name in a def, and entering a symbol after **for** all define a new symbol in most cases. In these contexts, **Tab** must be pressed to cause completion to occur.

The draw-back of operating in this mode is that Wing may fail to recognize some contexts where a new symbol is being defined, or may enter undesired completions when code is being typed before a referenced symbol has been

defined. To make canceling from the completer more convenient in this case, **Ctrl**, Alt``, and **Command** are also treated as cancel keys, in addition to **Esc**.

For the same reason, snippets do not participate in Turbo mode. To enter snippets found in the auto-completer, press **Tab** or **Enter**.

This mode is experimental. Please email feedback and suggestions to support@wingware.com.

How Auto-completion Works

The information shown in Wing's auto-completer comes from several sources: (1) Static analysis of Python code, (2) introspection of extension module contents, (3) inspection of keywords and builtins in the active Python version, (4) introspection of the runtime application state when the debugger is active or when working in the shells, (5) enumeration of relevant code snippets, and in some cases (6) user-provided interface description files. See Source Code Analysis for more information on how static analysis works and how you can help Wing determine the types of values.

Because static analysis can be defeated by Python's dynamic nature, it is sometimes more effective to work from live runtime state. This can be done by placing a breakpoint in the source code, running to it, and then working in the editor or (in Wing IDE Pro) in the Debug Probe.

In non-Python files, the auto-completer is limited to words found within similar contexts in the file, keywords defined for syntax highlighting that file type, and any snippets relevant to the editing context.

4.7. Source Assistant

The **Source Assistant** tool (in Wing IDE Personal and higher) can be used to display additional information about source symbols in the editor, auto-completer, and tools such as the **Project**, **Search in Files**, **Python Shell**, **Debug Probe**, and **Source Browser**.

The display will include links to the point of definition of the selected symbol, the symbol's type (when available) and a link to the type's point of definition. Depending on context and symbol type, the Source Assistant will also display relevant docstrings, call signature, return type, super-classes, overridden methods.

When invoking a function or method, the Source Assistant will display information both for the callable being invoked and the current argument or item in the auto-completer.

4.7.1. Docstring Type and Validity

By default the Source Assistant displays a type and validity indicator for docstrings, showing whether the docstring was successfully parsed or reformatted. The following messages may be displayed:

â■■ PEP287 -- The docstring parses successfully using PEP 287 reStructuredText Docstring Format and is being rendered accordingly. This only occurs when the **Use PEP 287 for docstrings** option is enabled.

â■■ PEP287 -- The docstring does not parse successfully as reStructuredText and is showing inline parse errors. This only occurs when the **Show PEP 287 parse errors** option is enabled.

Rewrapped -- The docstring is being shown as plain text but Wing has heuristically rewrapped paragraphs. This only occurs when the **Rewrap plain text docstrings** option is enabled.

Plain Text -- The docstring is being shown as plain text, exactly as it appears in the source code.

See Source Assistant Options for a list of the available display options.

4.7.2. Python Documentation Links

For symbols in the Python standard library, Wing will attempt to compute a documentation URL whenever possible. These point to http://docs.Python.org/ but can be redirected to another server with the Source Analysis > Advanced > **Python Docs URL Prefix preference**. To access locally stored documentation, a local http server must be used because **#** bookmark references do not work with **file:** URLs.

4.7.3. Working with Runtime Type Information

When working in the editor, auto-completer, project view, or source browser, the Source Assistant is fueled by Wing's Python source code analysis engine. Because of Python's dynamic nature, Wing cannot always determine the types of all values, but presents as much information as it can glean from the source code.

When a debug process is active, or when working in the **Python Shell**, Wing also extracts relevant information from the live runtime state. Since this yields complete and correct type information even for code that Wing's static analysis engine cannot understand, it is often useful to run to a breakpoint before designing new code that is intended to work in that context.

For more hints on helping Wing understand your source code, see Source Code Analysis and Helping Wing Analyze Code.

4.7.4. Source Assistant Options

There are several options available to control docstring rendering. These accessed by right clicking on the Source Assistant:

Use PEP 287 docstrings -- By default Wing tries to render docstrings by treating them as PEP 287 reStructuredText Docstring Format. This option can be used to disable PEP 287 rendering so they are always shown as plain text instead.

Show PEP 287 parse errors -- By default Wing falls back to displaying docstrings that cannot be parsed as reStructuredText as plain text. Enable this option to instead render them with reStructuredText parse errors.

Show docstring type and validity -- This enables or disables the floating docstring type and validity indicator in the top right of the docstring area.

Rewrap plain text docstrings -- By default Wing employs a heuristic to rewrap paragraphs in docstrings, in order to make better use of available display space. This option can be disabled to show the docstring exactly as it appears in the source code.

Always show docstrings -- By default Wing shows the docstring only of the last symbol being displayed in the Source Assistant, in order to save on display space. Enable this option to always show the docstring for all symbols.

The Source Assistant context menu can also be used to copy text or HTML to the clipboard, change the display font size, and access this documentation.

4.8. Auto-editing

Wing IDE Professional provides some optional auto-editing features, where the IDE tries to reduce typing by auto-entering expected text. The following operations are available:

- **Auto-Close Characters** -- Wing enters matching quotes, parentheses, brackets, braces, and comment closing characters. When this is enabled Wing skips over existing closing characters if they are typed anyway. Wing also auto-enters opening parentheses, brackets, and braces when an unmatched closing character is typed in Python code. This operation is disabled selectively when working within strings, comments, and in other contexts where the auto-edit is more likely to interfere than assist with editing. For example, quotes are not auto-closed within strings, most auto-closing is disabled within single-quoted strings, auto-closing is disabled if there is a matching unclosed character, auto-closing parentheses is disabled before a symbol, and some operations are omitted while auto-entering invocation arguments.
- **Auto-Enter Invocation Args** -- Wing enters the default arguments for a function or method invocation. The tab key or ',' can be used to move among the arguments. Argument entry ends when moving past the last argument, or pressing ')' at the last argument. Unaltered default arguments are automatically removed when argument entry ends.
- **Apply Quotes to Selection** -- Wing will surround a non-empty selection with quotes when the quote character is typed. In Python code, this will also convert the type of quote used in a string (either single quote or double quote) if the string is selected, or the caret is in the triple quote area, or one or more of the enclosing quotes is selected.

- **Apply Comment Key to Selection** -- For single-character comment keys, Wing will comment out or uncomment out the currently selected lines, using the configured **Block Comment Style**.
- **Apply (), [], and {} to Selection** -- When an open parenthesis, bracket, or brace is typed over a non-empty selection, Wing surrounds the selection with the matching characters.
- **Apply Colon to Selection** -- When one or more lines are selected, Wing creates a new block using those lines and places the caret for immediate entry of the block type (**if**, **try**, **for**, **with**, etc). When **try** is subsequently entered, Wing auto-enters the matching **except** block. In this case, **except** is selected so it can be changed into **finally**. Pressing the **Tab** key moves into the **except** or **finally** block.
- **Auto-Enter Spaces** -- In Python code, Wing auto-enters spaces when typing operators or punctuation. Some associated characters may also be entered, such as ',' after a dict item when ':' is pressed. When this operation is enabled, Wing also refuses to enter redundant spaces or commas in contexts where spacing is being enforced. In non-Python files this operation only enters spaces after a comma. Note that for some operations such as typing "==" spacing will be adjusted differently after the first and second keys are pressed. When this is enabled, the following sub-operations are available:

  - **Auto-Space After Keywords** -- In Python code, Wing also auto-enters spaces after keyword names. No space is added when the keyword name matches a snippet in the auto-completer, so that snippets can still be used.
  - **Enforce PEP8 Style Spacing** -- Wing will enforce PEP8 style spacing.
  - **Spaces in Argument Lists** -- Wing will auto-enter spaces in argument lists. This option overrides PEP8 style enforcement in that context, if it is enable.
- **Manage Blocks on Repeated Colon Presses** -- In Python code, Wing auto-indents the current line, enters the EOL character(s), and auto-indents the new line after a new block start is typed and ":" is pressed. In order to allow for adjustment of indentation before continuing, no EOL will be inserted after 'else', 'elif', 'except', and 'finally' if the indentation position for that statement is ambigious due to the presence of multiple potentially matching starting blocks. In that case, pressing ':' repeatedly will toggle the indentation between the possible positions. When this option is enabled and a new line was entered, pressing ":" a second time will remove the new line and indent the following line of code under the new block. Pressing ":" a third time will indent the next contiguous block of lines, up to any blank line or line that belongs to an enclosing block.

- **Continue Comment or String on New Line** -- Wing auto-enters comment or string delimiters when Enter is pressed within the text of an existing comment or a string in the form ("") or (').
- **Correct Out-of-Order Typing** -- Wing corrects common typos in a way that can reduce typing. For example, **x(.)** is replaced with **x().**, **x(:)** is replaced with **x():**, and Wing will add **.** when it is missing in **x().d**.

Each of these operations can be enabled or disabled independently in the **Auto-Editing** preferences group.

Where relevant (such as in spacing) Wing's auto-editing modes adhere to the PEP8 Style Guide for Python Code.

4.9. Bookmarks

Wing IDE Professional and higher support named bookmarks that can be set and accessed from the **Source** menu and the key bindings shown there, and with **Toggle Bookmark** in the editor context menu. Defined bookmarks are listed in the **Bookmarks** tool and are shown with a background color change or underline on the editor. The style and color of bookmark indicators can be changed with the **Bookmark Style** and **Bookmark Color** preferences.

Bookmark names are global to the project and refer to a particular position within a selected file:

- **For Python files,** bookmarks are defined relative to the enclosing scope (method, class, or function), so changes before the line where the bookmark is located will not cause the bookmark's relative position in source code to be changed, even if those changes are made outside of Wing. Edits made outside of Wing that affect the code between the anchoring scope and the bookmarked line will cause a bookmark's position to slip.
- **For all other types of files,** bookmarks are defined simply by file name and line number. If the file is edited outside of Wing, the bookmark's position may appear to slip.

When navigating to a bookmark from the Source menu or key binding, Wing will present a dialog or entry area at bottom of the screen (depending on editor personality) into which the bookmark name can be typed. A list of possible completions will be displayed. Pressing tab will select the currently highlighted completion.

A list of defined bookmarks is available in the Bookmarks tool, which is available from the Tools menu. Right click on an entry for a context menu of operations for the selected bookmark or bookmarks. Multi-selection is possible by holding down the shift and/or control keys. Double clicking or middle mouse clicking will navigate to the selected bookmark.

When the **Bookmarks** tool has focus, keyboard navigation is possible with the arrow keys and by typing letters to move quickly to a particular bookmark. Enter can then be pressed to navigate to the selected bookmark.

When bookmarks are defined in the current editor file, then Wing will add a bookmark icon to the top right of the editor. Clicking on this will pop up a menu of the bookmarks in the current file.

Traversing bookmarks (within the current file or within all files) is also possible with the **Traverse Bookmarks** sub-menu in the **Source** menu and the key bindings shown there.

In VI mode, the standard **m** and **\`** plus key bindings are supported, in addition to the operations in the **Source** menu, which allow for the definition of bookmarks with names longer than one character.

Emacs, Brief, and other key bindings also support bookmarks. However, bookmark functionality for VI, Emacs, and Brief key bindings is omitted in Wing IDE Personal.

4.10. File Sets

File Sets are used create named sets of files that can then be opened as a group or searched from the Search in Files tool.

File sets can be created in several ways:

- Open the desired files and use the **Name Set of Open Files...** item in the **Files > File Sets** menu.
- Select the desired files in the Project, Open Files, or in other tools and use the **Name Set of Selected Files...** item in the **Files > File Sets** menu.
- Select the desired files in the Project or Open Files tool, right click and select the **Name Selected File Set...** menu item.
- Search in the Search in Files tool and when the search is complete use the **Name Result File Set** item in the **Options** menu to name the set of files in which a search match was found.

Once defined, file sets can be opened from the **Files > File Sets** menu and they are included by name in the Search in Files tool's **Look in** menu.

To view or edit the defined file sets, use the **Manage File Sets...** item in the **File > File Sets** menu. Right click to access the available operations in this dialog. To rename a file set, click on its name and edit the name in place.

Binding File Sets to Keys

File sets can be bound to a key sequence, so that the pressing that sequence will open the file set in the editor. This is done in the **Manage File Sets** dialog, by selecting the file set, right clicking, and selecting **Set Key Binding...**.

Shared File Sets

File sets can either be stored in the project file (the default) or in a shared file that is used by all projects. To make a file set into a shared file set, open the **Manage File Sets** dialog and check the **Shared** checkbox.

4.11. Code Snippets

Wing Professional provides support for defining and using code snippets for commonly reused bits of code and other text. Snippets might be used for standard file skeletons, comment formats, dividers, class definitions, function definitions, HTML tables, and much more. Variants of snippets may be defined for different contexts, for example to include or omit **self** in a **def** depending on whether or not it is a method in a class.

Wing's snippet functionality is implemented in the **Snippets** tool panel and by providing the snippets by name in the editor's auto-completer. Key bindings can be assigned to snippets so that the snippets tool does not have to be visible in order to use a snippet.

Although Wing comes with a few example snippets, in most cases users will want to define their own, to match their coding conventions and preferences.

User Interface

The **Snippets** tool panel provides the means for adding, editing, removing, and executing snippets, and also assigning key bindings for pasting selected snippets into the current editor. Most of the functionality is provided by the options menu in the top right and by right clicking on the snippet list. Note that some of the operations (those followed by **...** in the menus) will prompt for input at the bottom of Wing IDE's window.

The option menu in the top right of the **Snippets** tool (also accessible by right-clicking on the tab area) provides items for adding, removing, and renaming file types into which to organize snippets. The name of the file type is the file extension that Wing should use by default when creating a new file based on a snippet. It is also used to look up the mime type of the file, so that the snippet can be made available within any file of that type, regardless of its actual name. The ***** file type, which is always present, allows defining snippets that can be applied to all file types.

To add, edit, renamed, copy, and remove snippets, use the items in the context menu that appears when you right-click on the surface of the snippet list in the **Snippets** tool. This menu also provides items for inserting the snippet into the current file or a new file.

Contexts

It is possible to specify the context within the file for which a snippet is appropriate. This allows, for example, the definition of a snippet **def** that varies to include or omit **self** depending on whether or not it is within a class. When available, this is done with items in the snippet list context menu. The snippet defined for context **all** will be used when no specific context match is made. The default set of snippets that ship with Wing illustrate this feature with the **def** and **class** snippet variants.

The set of valid contexts depends on file type. For Python files the valid context names are **module**, **class**, **method**, **function**, **comment**, **string**. For HTML and XML, files are divided into **content**, **code** (within **<** and **>**), **comment**, and **string**. Other files only distinguish **code**, **comment**, and **string**. Additionally, the context **all** is used for all file types to indicate any context.

To set the context for a snippet, click on the context name in the snippet lists's **Context** column, or use the items in the right-click context menu on the snippet list.

Key Bindings

The snippet list context menu menu also allows assigning key bindings to snippets, so they can be executed more easily. The key binding entry area is shown at the bottom of the IDE window, and **Enter** is pressed to accept the displayed binding. Note that bindings can be multi-key sequences such as **Ctrl-Shift-H Ctrl-A**. Pressing the keys in sufficiently rapid succession creates a sequence. Waiting a moment will start a new sequence when further keys are pressed. Clicking away from the entry area will abort the operation without assigning any key binding.

Note that key bindings are assigned to the snippet by name and not to a particular snippet file. If multiple like-named snippets exists for different file types or contexts, the appropriate snippet is chosen when the key binding is used.

Execution and Data Entry

When snippets are executed, Wing chooses the snippet by name and places the correct variant according to the file type and the context within the current editor. The caret position on the editor is used to determine the context, so altering the position of the caret within leading indentation may alter which snippet variant Wing selects.

When a snippet is used, Wing will place default arguments into the snippet, convert indentation and line endings to match the target file, paste it into the active editor, and place the editor into inline data entry mode to collect additional arguments for the snippet.

In data entry mode, Wing will move between the data entry fields in the snippet when **Tab** or **BackTab** are pressed. The position within the snippet's fields will be displayed in the status area at the bottom of the editor window.

In this mode, the **Indent** and **Outdent** commands in the **Indentation** sub-group of Wing's **Source** menu (and their key equivalents) can be used to increase or decrease the indentation of the whole snippet within the editor. However, the same snippet variant that was used initially will be used regardless of subsequent changes in indentation.

To exit data entry mode, press **Esc** (or **Ctrl-G** in emacs mode) or move the caret outside of the pasted snippet. To undo the snippet insertion, use **Undo** in the **Edit** menu or its key binding.

Auto-completion

Snippets are also listed in the editor's auto-completer and may be activated from there. To disable this feature, turn off the auto-completer preference **Include Snippets in Completer**.

Snippet Syntax

Snippets are text files that contain markers where user-provided values should be inserted. These markers are similar to Python's **%(varname)s** string substitution syntax but instead of containing only a variable name, the body of the marker contains richer argument collection information in the following format, with vertical bars dividing each value:

```
%(varname|type|default)s
```

Type and default are optional but the vertical bars must be present if omitting type but including a default. To write a snippet that includes Python style string formats, escape each **%** by writing **%%** instead.

Each part is defined as follows:

varname -- The name of the variable. Since arguments are collected inline, this name is used internally only. Future extensions may display this name to the user, by replacing underscores with spaces and capitalizing words (for example "user_name" would be rendered "User Name"). If a variable name is used multiple times in a snippet, the same value will be inserted multiple times.

An @ prepended to the variable name indicates that the value should be wrapped if it exceeds the configured **text wrap line column**.

type -- The type of data to collect. Currently this is one of:

string(length) -- a string with given maximum length (uses default 80 chars if length is omitted)

date -- current date in locale's preferred format or in the **time.strftime()** format given in the environment variable **__DATE_FORMAT__**

datetime -- current date+time in locale's preferred format or in the **time.strftime()** format given in the environment variable **__DATETIME_FORMAT__**

If this field is omitted or empty, string is assumed.

default -- The default value to use. This may be the actual value, or may contain environment variable references in the form **$(envname)** or **${envname}** to attempt to read all or part of the value from the named environment variable or one of the special variables enumerated in Environment Variable Expansion.

Environment variables can be specified either in the **Debug** tab of Wing's **Project Properties** or in the environment that exists before Wing is launched. Values in the Project Properties override any values set before starting Wing.

When this field is omitted, or when no default environment value can be found, the field will be left blank .

Indentation and Line Endings

Snippets should always use one tab for each level of indentation. Tabs will be replaced with the appropriate indentation type and size when the snippet is used in a new or existing file (either according to content of the target file or using the configured **indent style** and **indent size** for new files). Wing will force tab indentation in all newly created snippet files.

Similarly, line endings in snippets will be replaced with the appropriate type to match the file to which the snippet is applied. However, there is no requirement for snippet files to contain any particular kind of line ending.

If the snippet starts with **|x|** then **x** is a specification of how the indents in the snippet should be converted. It can be one of:

- *An integer*: Re-indent as a block, like Wing's indent-region command, so the first line is at the given number of indent levels.
- *The character 'm'*: Re-indent as a block, like Wing's indent-to-match command, so the first line is at the expected indent level according to context in the source.
- *The character 'm' followed by '+' or '-' and an integer*: Re-indent as for **'m'** and then shift left or right by the given number of indents.

Any **|x|** at the start of a snippet file will be removed before the snippet is inserted into an editor.

Cursor Placement

Snippets can contain **|!|** to indicate the final resting position of the cursor after all other fields have been filled. When this is present, inline data entry mode is terminated automatically when this position is reached (after all other fields have been entered). The mark will be removed before snippets are inserted into an editor.

Snippet Directory Layout

Snippets are stored in the **snippets** directory in the user settings directory. The first time the **Snippets** tool is used, this directory is populated by making a copy of the default set of snippets that ship with Wing (these can be found in **snippets** within your Wing IDE installation).

File Types

This directory is organized by the file type to which they apply. Snippets stored at the top level of this directory can be used with any file in the editor and are shown in the ***** tab in the Snippets tool. Those stored in sub-directories are used only for files of a particular type. The name of the sub-directories is the file extension for that file type (for example **py** for Python). This is converted to a mime type so that the snippets are available for all files of that type, regardless of their naming. The name of the file type directory also provides the file extension to use for new untitled files when a snippet is pasted into a new file.

Contexts

When snippets are defined for a particular context within a file, they are stored in a sub-directory named **context.ctx** where **context** is replaced with the context name (see above). When a snippet is defined as the default, or without a particular context, it is stored in the top level of the file type directory.

Configuration

Wing also stores a configuration file in the user's snippets directory. This file is named **.config** and is used for internal book keeping. It should not be altered or removed, as this may cause the loss of your snippet files.

Commands

The following commands are available for invoking snippets:

snippet -- This will insert a snippet (selected by name) at the cursor in the current editor. If there is a non-empty selection on the editor, it will replace the selection. The editor will be placed into data entry mode for the collection of the snippet arguments.

snippet-file -- This will create a new file of the type specified by the snippet file's extension and insert the selected snippet into it before entering data entry mode in the editor for the collection of the snippet arguments.

In most cases, you will use the **Assign Key Binding** item in the **Snippets** tool's context menu to invoke these commands for a particular snippet.

Scripting Snippets

Wing's extension scripting API exposes the editor's data entry mode and snippet processing capabilities. This can be used to write scripts that generate snippets and paste them into the editor for user data entry. This approach may be preferable when the snippet markup language described above is not sufficient.

For details, see the **PasteSnippet** and **StartDataEntry** methods in **wingapi.py** and refer to Scripting and Extending Wing IDE.

4.12. Indentation

Since indentation is syntactically significant in Python, Wing provides a range of features for inspecting and managing indentation in source code.

4.12.1. How Indent Style is Determined

When an existing file is opened, it is scanned to determine what type of indentation is used in that file. If the file contains some indentation, this may override the tab size, indent size, and indent style values given in preferences and the file will be indented in a way that matches its existing content rather than with your configured defaults. If mixed forms of indentation are found, the most common form is used.

For non-Python files, you can change indentation style on the fly using the **Indent Style** property in the **File Properties** dialog (accessed by right-clicking on the editor). This allows creating files that intentionally mix indentation forms in different parts of the file. To ask Wing to return to the form of indentation it determines as most prominent in the file, select **Match Existing Indents**.

For Python files, the **Indent Style** cannot be altered without converting the whole file's indent style using the Indentation Manager, which can be accessed from the button next to the **Indent Style** property and from the Tools menu.

4.12.2. Indentation Preferences

The following preferences affect how the indentation features behave:

1. The **Use Indent Analysis** preference is used to control whether analysis of current file content is used to determine the type of indentation placed during edits. It can be enabled for all files, only for Python files, or disabled. Note that disabling this preference for Python files can result in a potentially broken mix

of indentation in the files. In general, indent styles should not be mixed within a single Python file.

2. The **Default Tab Size** preference defines the position of tab stops and is used to determine the rendering of files with tabs only, or non-Python files with mixed tab and space indentation. In Python files with mixed indents, this value is ignored and the file is always shown in the way that the Python interpreter would see it.
3. The **Default Indent Size** preference defines the default size of each level of indent, in spaces. This is used in new empty files or when indent analysis has been disabled. Wing may override this value in files that contain only tabs in indentation, in order to make it a multiple of the configured tab size.
4. The **Default Indent Style** preference defines the default indentation style, one of **spaces-only**, **tabs-only**, or **mixed**. This is used in new empty files or when indent analysis has been disabled. Mixed indentation replaces each tab-size spaces with one tab character.

These preferences define how indentation is handled by the editor:

5. The **Auto-Indent** preference controls whether or not each new line is automatically indented.
6. The **Show Indent Guides** preference controls whether or not to show indentation guides as light vertical lines. This value can be overridden on a file-by-file basis from Editor tab in File Properties.
7. The **Show Python Indent Warnings** preference can be used to enable or disable warnings for Python files that may contain confusing or damaged indentation.
8. The **Show Override Warnings** preference controls whether or not Wing shows a warnings when the user enters indentation that does not match the form already within a file. This is currently only possible in non-Python files, by altering the **Indent Style** attribute in File Properties.

4.12.3. Indentation Policy

The project manager also provides the ability to define the preferred indentation style (overriding the preference-defined style) and to specify a policy for enforcing line endings, on a per-project basis. This is accomplished with **Preferred Line Ending** and **Line Ending Policy** under the Options tab in Project Properties.

4.12.4. Auto-Indent

The IDE ships with auto-indent turned on. This causes leading white space to be added to each newly created line, as return or enter are pressed. Enough white space is inserted to match the indentation level of the previous line, possibly adding or removing a level of indentation if this is indicated by context in the source (such as **if**, **while**, or **return**).

Note that if preference **Auto-indent** is turned off, auto-indent does not occur until the tab key is pressed.

In Python code, Wing also auto-indents after typing a colon after **else**, **elif**, **except**, and **finally**. Indentation will go to the closest matching **if** or **try** statement. I f there are multiple possible matching statements, the colon key can be pressed repeatedly to toggle through the possible positions for the line. Similarly, when **Smart Tab** is selected as the Tab Key Action, then pressing the Tab key repeatedly will toggle the line through the possible indent positions. This can also be accomplished with the **Indent to Match** toolbar and menu items (regardless of selected tab key action).

When pasting multiple lines into Python code and the caret is in the indent region or on a blank line, Wing will auto-indent pasted text as follows: (1) If the caret is in column zero, the text is indented to match the context, (2) If the caret is within the indent region but not in column zero, the text is indented to that position. If the auto-indent is incorrect, a single **Undo** will return the pasted text to its original indentation level, or the text can be selected and adjusted with the indentation toolbar or menu items or key equivalents.

4.12.5. The Tab Key

By default, the action of the tab key depends on the selected **Keyboard Personality**, file type, and position within the file as described under **Default for Personality** below.

To insert a real tab character regardless of the indentation mode or the position of the cursor on a line, type Ctrl-Tab or Ctrl-T.

The behavior of the tab key can be altered using the **Tab Key Action** preference, which provides the following options:

Default for Personality

This selects from the other tab key actions below according to the chosen keyboard personality, current file type, and in some cases the position of the caret within the file. In all non-Python files, the default is Move to Next Tab Stop. In Python files, the defaults are as follows by keyboard personality:

- **Normal**: Smart Tab
- **VI/VIM**: Move to Next Tab Stop

- **Emacs**: Indent to Match
- **Brief**: Smart Tab
- **Visual Studio**: Move to Next Tab Stop
- **OS X**: Smart Tab

Indent to Match

This indents the current line or selected lines to position them at the computed indent level for their context in the file.

Move to Next Tab Stop

This enters indentation characters matching the current file's style of indentation so that the caret reaches the next tab stop.

Indent Region

This enters indentation characters matching the current file's style of indentation to increase the indentation of the current line or selected lines by one level.

Insert Tab Character

This inserts a Tab character (chr(9)) into the file.

Smart Tab

This option is available for Python files only. It implements the following behavior for the tab key:

1. When the caret is within a line or there is a non-empty selection, this performs Indent to Match. When the line or lines are already at the matching position, indentation is toggled between likely positions as follows:

    a. If a comment precedes the current line or selection, then indentation will match the position of the prior non-comment code line (if any).
    b. If multiple nested blocks match an 'else', 'elif', 'except', or 'finally', then indentation will match the position of the enclosing blocks (traversing each in outward order).

    b. In other cases, indentation is reduced by one level.
2. When the caret is at the end of a non-empty line and there is no selection, one indent level is inserted. The **Smart Tab End of Line Indents** preference can be used to alter the type of indentation used or to disable this aspect of the Smart Tab feature.

4.12.6. Checking Indentation

Wing IDE analyzes existing indentation whenever it opens a Python source file, and will indicate a potentially problematic mix of indentation styles, allowing you to attempt to repair the file. Files can be inspected more closely or repaired at any time using the Indentation Manager.

To turn off indentation warnings in Python files, use the **Show Python Indent Warnings** preference.

Wing also indicates suspiciously mismatched indentation in source code by underlining the indent area of the relevant lines in red or yellow. In this case, an error or warning message is displayed when the mouse hovers over the flagged area of code.

4.12.7. Changing Block Indentation

Wing provides **Indent** and **Outdent** commands in the **Indentation** portion of the Source menu, which increase or decrease the level of indentation for selected blocks of text. All lines that are included in the current text selection are moved, even if the entire line isn't selected.

Indentation placed by these commands will contain either only spaces, only tabs, or a mixture of tabs and spaces, as determined by the method described in Indentation.

> ***Note***
>
> **Indenting to Match**
>
> The command **Indent Lines to Match** (also in the **Indentation** sub-menu) will indent or outdent the current line or selected lines to the level as a unit so that the first line is positioned as it would have been positioned by Wing's auto-indentation facility. This is very useful when moving around blocks of code.

4.12.8. Indentation Manager

The Indentation manager, accessible from the **Tools** menu, can be used to inspect and change indentation style in source files. It has two parts: (1) The indentation report, and (2) the indentation converter.

A report on the nature of existing indentation found in your source file is given above the horizontal divider. This includes the number of spaces-only, tabs-only, and mixed tabs-and-space indents found, information about whether indentation in the file may be problematic to the Python interpreter, and the tab and indent size

computed for that file. The manager also provides information about where the computed tab and indent size value come from (for example, an empty file results in use of the defaults configured in preferences).

Conversion options for your file are given below the horizontal divider. The three tabs are used to select the type of conversion desired, and each tab contains information about the availability and action of that conversion, and a button to start the conversion. In some of the conversion options, the indent size field shown in the indentation report is made editable, to allow specification of the desired resulting indent size.

Once conversion is complete, the indentation manager updates to display the new status of the file, and action of any subsequent conversions.

Conversions can be undone be moving to the converted source file and selecting **Undo** from the **Edit** menu.

4.13. Folding

The editor supports optional folding for Python, C, C++, Java, Javascript, HTML, Eiffel, Lisp, Ruby, and a number of other file formats. This allows you to visually collapse logical hierarchical sections of your code while you are working in other parts of the file.

You can turn folding on and off as a whole with the **Enable Folding** preference.

The **Fold Line Mode** preference can be used to determine whether or not a horizontal line is drawn at fold points, whether it is drawn above or below the fold point, and whether it is shown when the fold point is collapsed or expanded. **Fold Indicator Style** is used to select the look of the fold marks shown at fold points.

Once folding is turned on, an additional margin appears to the left of source files that can be folded. Left mouse click on one of the fold marks in this margin to collapse or expand that fold point. Right mouse clicking anywhere on the fold margin displays a context menu with the various folding operations.

You can also hold down the following key modifiers while left-clicking to modify the folding behavior:

- **Shift** -- Clicking on any fold point while holding down the shift key will expand that point and all its children recursively so that the maximum level of expansion is increased by one.
- **Ctrl** -- Clicking on any fold point while holding down the ctrl key will collapse that point and all its children recursively so that the maximum level of expansion is decreased by one.
- **Ctrl+Shift** -- On a currently expanded fold point, this will collapse all child fold points recursively to maximum depth, as well as just the outer one. When the

fold point is subsequently re-expanded with a regular click, its children will appear collapsed. Ctrl-shift-click on a collapsed fold point will force re-expansion of all children recursively to maximum depth.

Fold commands are also available in the **Folding** section of the **Source** menu, which indicates the key equivalents assigned to the operations:

- **Toggle Current Fold** -- Like clicking on the fold margin, this operates on the first fold point found in the current selection or on the current line.
- **Collapse Current More** -- Like ctrl-clicking, this collapses the current fold point one more level than it is now.
- **Expand Current More** -- Like shift-clicking, this expands the current fold point one more level than it is now.
- **Collapse Current Completely** -- Like shift-ctrl-clicking on an expanded node, this collapses all children recursively to maximum depth.
- **Expand Current Completely** -- Like shift-ctrl-clicking on a collapsed node, this ensures that all children are expanded recursively to maximum depth.
- **Collapse All** -- Unconditionally collapse the entire file recursively.
- **Expand All** -- Unconditionally expand the entire file recursively.
- **Fold Python Methods** -- Fold up all methods in all classes in the file.
- **Fold Python Classes** -- Fold up all classes in the file.
- **Fold Python Classes and Defs** -- Fold up all classes and any top-level function definitions in the file.

4.14. Brace Matching

Wing will highlight matching braces in green when the cursor is adjacent to a brace. Mismatched braces are highlighted in red.

You can cause Wing to select the entire contents of the innermost brace pair from the current cursor position with the Match Braces item in the Source menu.

Parenthesis, square brackets, and curly braces are matched in all files. Angle brackets (**<** and **>**) are matched also in HTML and XML files.

4.15. Support for files in .zip or .egg files

Source and other text files stored in .zip or .egg files may be loaded into the editor as readonly files. Wing is unable to write changes to a file within a .zip or .egg file or otherwise write to or create a .zip or .egg file.

When stepping through code, using goto definition, or using other methods to goto a line in a file, a file within a .zip or .egg file will be opened automatically. To open a file through the open file dialog, specify the name of the .zip or .egg file and add a **/** followed by the name of the file to open.

4.16. Keyboard Macros

The Edit menu contains items for starting and completing definition of a keyboard or command sequence macro, and for executing the most recently defined macro. Once macro recording is started, any keystroke or editor command is recorded as part of that macro, until macro recording is stopped again. Most commands may be included in macros, as well as all character insertions and deletions.

Macros can be quite powerful by combining keyboard-driven search (**Mini-search** in the **Edit** menu), cursor movements, and edits.

4.17. Notes on Copy/Paste

There are a number of ways to cut, copy, and paste text in the editor:

- Use the Edit menu items. This stores the copy/cut text in the system-wide clipboard and can be pasted into or copied from other applications.
- Use key equivalents as defined in the Edit menu.
- Right-click on the editor surface and use the items in the popup menu that appears.
- Select a range of text and drag it using the drag and drop feature. This will move the text from its old location to the new location, either within or between editors.
- On Linux, select text anywhere on the display and then click with the middle mouse button to insert it at the point of click.
- On Windows and Mac OS X, click with the middle mouse button to insert the current emacs private clipboard (if in emacs mode and the buffer is non-empty) or the contents of the system-wide clipboard (in all other cases). This behavior may be disabled via the **Middle Mouse Paste** preference
- In emacs mode, ctrl-k (**kill-line**) will cut one line at a time into the private emacs clipboard. This is kept separate from the system-wide clipboard and is pasted using ctrl-y (**yank-line**). On Windows and Mac OS X, ctrl-y will paste the contents of the system-wide clipboard only if the emacs clipboard is empty.
- In VI mode, named text registers are supported.

It is important to note which actions use the system-wide clipboard, which use the emacs private clipboard or VI registers, and which use the X windows selection (X Windows only). Otherwise, these commands are interchangeable in their effects.

Smart Copy

Wing can be configured to copy or cut the whole current line when there is no selection on the editor. This is done with **On Empty Selection** in the **Editor > Clipboard** preference group. The default is to use the whole line on copy but not cut.

4.18. Auto-reloading Changed Files

Wing's editor detects when files have been changed outside of the IDE and can reload files automatically, or after prompting for permission. This is useful when working with an external editor, or when using code generation tools that rewrite files.

Wing's default behavior is to automatically reload externally changed files that have not yet been changed within Wing's source editor, and to prompt to reload files that have also been changed in the IDE.

You can change these behaviors by setting the value of the **Reload when Unchanged** and **Reload when Changed** preferences

On Windows, Wing uses a signal from the OS to detect changes so notification or reload is usually instant. On Linux and Unix, Wing polls the disk by default every 3 seconds; this frequency can be changed with the **External Check Freq** preference.

4.19. Auto-save

The source code editor auto-saves files to disk every few seconds. The auto-save files are placed in a subdirectory of your User Settings Directory.

If Wing ever crashes or is killed from the outside, you can use these files to manually recover any unsaved changes. Copy the auto-save files to overwrite the older unsaved files, doing a comparison first to verify that the auto-save file is what you want.

Search/Replace

Wing provides a number of tools for search and replace in your source code. Which you use depends on the complexity of your search or replace task and what style of searching you are most familiar with.

5.1. Toolbar Quick Search

One way to do simple searches is to enter text in the search area of the toolbar. This scrolls as you type to the next match found after the current cursor position. Pressing **Enter** will search for each subsequent match, wrapping the search when the end of the file is reached.

Text matching during toolbar quick search is case-insensitive unless you enter a capital letter as part of your search string.

If focus is off the toolbar search area and it already contains a search string, clicking on it will immediately start searching in the current source editor for the next match. If you wish to search for another string instead, delete the text and type the desired search string. As you delete, the match position in the editor will

proceed backward until it reaches your original search start position, so that after typing your new search string you will be presented with the first match after the original source editor cursor position.

5.2. Keyboard-driven Mini-Search/Replace

The Edit menu contains a Mini-Search sub-menu that enumerates the available keyboard-driven search options. These are normally initiated with the keyboard command sequences shown in the menu and can be controlled entirely by using the keyboard. All interaction with the mini-search manager occurs using data entry areas displayed on demand at the bottom of the IDE window.

The implementation of the mini-search manager is very close to the most commonly used search and replace features found in Emacs, but it is available whether or not the Emacs editor personality is being used.

The following search and replace features are available in this facility:

- **Forward** and **Backward** -- These display a search string entry area at the bottom of the IDE window and interactively search forward or backward in the current source editor, starting from the current cursor position. The search takes place as you type and can be aborted with **Esc** or **Ctrl-G**, which returns the editor to its original cursor location and scroll position.

  Searching is case-insensitive unless you enter a capital letter as part of your search string. To search repeatedly, press **Ctrl-U** (or **Ctrl-S in emacs keyboard mode) to search forward and ``Ctrl-Shift-U** (or **Ctrl-R** in emacs mode) to search in reverse. The search direction can be changed any number of times and searching will wrap whenever the top or bottom of the file is reached. You can also enter **Ctrl-U** (or **Ctrl-S** in emacs mode) or **Ctrl-Shift-U** (or **Ctrl-R** in emacs mode) again initially while the search string is still blank in order to call up the most recently used search string and begin searching forward or backward with it.

  Once the mini-search entry area is visible, **Ctrl-W** will add the current word in the editor to the search string. Pressing **Ctrl-W** more than once while the mini-search entry is visible adds additional words from the editor to the search string.
- **Selection Forward** and **Selection Backward** -- These work like the above but start with the selection in the current source editor.
- **Regex Forward** and **Regex Backward** -- These work like the above but treat the search string as a regular expression.
- **Query/Replace** and **Regex Query/Replace** -- This prompts for search and replace strings in an entry area at the bottom of the IDE window and prompts for replace on each individual match found after the cursor location in the

current source editor. Press **y** to replace or **n** to skip a match and move on to the next one. The interaction can be canceled at any time with **Esc** or **Ctrl-G**. Matching is case insensitive unless a capital letter is entered as part of the search string. Searching is always forward and stops when the end of the file is reached, without wrapping to any un-searched parts between the top of the file and the position from which the search was started.

- **Replace String** and **Replace Regex** -- This works like the above command but immediately replaces all matches without prompting.

5.3. Search Tool

The dockable **Search** tool can be used for more advanced search and replace tasks within the current editor. It provides the ability to customize case sensitivity and whole/part word matching, search in selection, and perform wildcard or regex search and replace.

The **Replace** field may be hidden and can be shown from the **Options** menu in the bottom right of the tool.

To the right of the **Search** and **Replace** fields, Wing makes available a popup that contains a history of previously used strings, options for inserting special characters, and an option for expanding the size of the entry area.

The following search options can be selected from the tool:

- **Case Sensitive** -- Check this option to show only exact matches of upper and lower case letters in the search string.
- **Whole Words** -- Check this option to require that matches are surrounded by white space (spaces, tabs, or line ends) or punctuation other than _ (underscores).
- **In Selection** -- Search for matches only within the current selection on the editor.

The following additional options are available from the Options popup menu:

- **Show Replace** -- Whether or not the Replace field is visible in the tool.
- **Text Search** -- Select this to do a regular text search without wildcard or regex.
- **Wildcard Search** -- Select this to allow use of special characters for wildcarding in the search string (see Wildcard Search Syntax for details).
- **Regex Search** -- Select this to use regular expression style searching. This is a more powerful variant than wildcard search that allows for more complex specification of search matches and replacement values. For information on the syntax allowed for the search and replace strings, see Python's Regular Expression Syntax documentation. In this mode, the replace string can reference regex match groups with **\1**, **\2**, etc, as in the Python **re.sub()** call.

- **Wrap Search** -- Uncheck this to avoid wrapping around when the search reaches the top or bottom of a file.
- **Incremental** -- Check this to immediately start or restarted searching as you type or alter search options. When unchecked, use the forward/backward search buttons to initiate searching.
- **Find After Replace** -- Select this to automatically find the next search match after each Replace operation.

5.4. Search in Files Tool

The dockable **Search in Files** tool is used to search and replace within sets of files, or for searching Wing's documentation. It performs searches in batch and displays a result list for all found matches. This list can then be traversed to view the matches in the source editor, and is automatically updated as edits alter the search results. Searching may span the current editor, a single selected file, all open files, all project files, all of Wing's documentation, or sets of files on disk.

Files in a set may be filtered by file type, for example searching only through Python files in the project.

In addition the options also available in the search tool, the following choices are available in the **Options** popup menu:

- **Replace Operates On Disk** -- Check this to replace text in un-opened files directly on disk. Caution: see Replace in Multiple Files for details on this option.
- **Recursive Directory Search** -- Check this to search recursively within all sub-directories of the selected search directory.
- **Omit Binary Files** -- Check this to omit any file that appears to contain binary data.
- **Auto-restart Searches** -- Check this to restart searching immediately if it is interupted because a search parameter or the set of files being searched is changed.
- **Open First Match** -- Check this to automatically open the first batch search match, even before the result list is clicked upon.
- **Show Line Numbers** -- Check this to include line numbers in the batch result area.
- **Result File Name** -- This is used to select the format of the result file name shown in the batch result area.

5.4.1. Replace in Multiple Files

For searches that operate on open files, replace always occurs in the open file editor and can be undone or saved to disk subsequently, as with any other edit operation.

When replacing text in batch mode, some of the files being searched may not currently be open in an editor. In this case, Wing will by default open all altered files and make changes in newly created editors that remain open until the user saves and closes them explicitly. This is the safest way to undertake multi-file global replace operations because it clearly shows which files have been altered and makes it possible to undo changes.

An alternative approach is available by selecting the **Replace Operates on Disk** option from the **Options** popup. This will cause Wing to change files directly on disk in cases when there is no currently open editor.

Because global replace operations can be tricky to do correctly, we *strongly* recommend using a revision control system or frequent backups and manually comparing file revisions before accepting files that have been altered.

5.5. Find Points of Use

Wing IDE is able to find the locations where a symbol is used in the current project's Python files. To start a search, select or place the cursor in a symbol and then use **Find Points of Use** in the **Source** menu or editor's context menu (right-click) or **Alt-click** on a symbol. Lines with matching symbols will be displayed in the **Uses** tool and clicking on a match will display it in an editor.

Since Python is a dynamic language, it is sometimes impossible to determine for certain whether a match is the same symbol. Matches are assigned a likelyhood of being correct, as follows:

- **Likely**: The original symbol and found symbol resolve to the same definition so that using **Goto Definition** on each will end up in the same place.
- **Possible**: Either the original symbol or the found symbol don't resolve to any definition.
- **Unlikely**: The original symbol resolves to a different definition than the found symbol.

Possible matches are listed with a question mark **(?)** preceding the filename and unlikely matches are listed with double question mark **(??)** preceding the filename. Only likely and possible matches are displayed by default. The display of possible and unlikely matches may be toggled via the **Options** menu on a per-search basis.

When searching for uses of a class **__init__** or **__new__** methods, the results will include matches where the class or a derived class is used by their original name to create new instances.

If Wing is failing to see matches as resolving to the same point of definition, it may help to add to the Python Path in Project Properties so that the source analysis engine can resolve module imports.

Completed searches are stored in the **Uses** tool and can be referenced by clicking on the drop down menu at the top of the tool and deleted by clicking on the close icon. Note that searches do not automatically refresh as code is modified, but may be updated manually with **Refresh** in the **Options** menu.

5.6. Wildcard Search Syntax

For wild card searches in the Search tools, the following syntax is used:

***** can be used to match any sequence of characters except for line endings. For example, the search string **my*value** would match anything within a single line of text starting with **my** and ending with **value**. Note that ***** is "greedy" in that **myinstancevalue = myothervalue** would match as a whole rather than as two matches. To avoid this, use Regex Search instead with **.*?** instead of *****.

? can be used to match any single character except for line endings. For example, **my???value** would match any string starting with **my** followed by three characters, and ending with **value**.

[and **]** can be used to indicate sets of match characters. For example **[abcd]** matches any one of **a**, **b**, **c**, or **d**. Also, **[a-zA-Z]** matches any letter in the range from **a** to **z** (inclusive), either lower case or uppercase. Note that case specifications in character ranges will be ignored unless the **Case Sensitive** option is turned on.

Refactoring

Wing IDE includes support for refactoring, which is the process of modifying code to improve its structure and organization without changing its behavior. For example, refactoring can be used to rename a symbol wherever it is referenced or to move a block of code into a function, replacing it with an invocation of the new function. Wing IDE's refactoring support provides very high-level editing operations that are informed by its understanding of Python source code.

6.1. Rename Symbol

The rename symbol operation renames a variable, function, class, or module and updates the locations where it is used. To start a rename operation, click on the symbol in the editor and then select **Rename Symbol** from the **Refactor** menu or from the **Refactor** sub-menu of the editor context menu (right-click). Wing will begin searching for all of the locations where the symbol is used and list them in the **Refactoring** tool. To complete the operation, enter the new symbol name and press the **Rename Checked** button.

Note that each found match for the symbol is displayed with a check box that can be deselected to omit that match from the rename operation. Please refer to Find

Points of Use for more information on how Wing finds symbols for refactoring operations.

After it completes, the rename operation can be undone with the **Revert** button in the **Refactoring** tool.

6.2. Move Symbol

The move symbol operation moves a variable, function, or class, and updates locations where it is used to reference the symbol at its new location. To start a move operation, click on the symbol to be moved and then select **Move Symbol** from the **Refactor** menu or from the **Refactor** sub-menu of the editor context menu (right-click). Wing will search for all of the locations where the symbol is used and list them in the **Refactoring** tool. To complete the operation, enter the destination filename and / or scope name and press the **Move and Update Checked** button.

Note that each found match for the symbol is displayed with a check box that can be deselected to omit that match from the rename operation. Please refer to Find Points of Use for more information on how Wing finds symbols for refactoring operations.

After it completes, the rename operation can be undone with the **Revert** button in the **Refactoring** tool.

6.3. Extract Function / Method

The extract function / method operation creates a new function or method from the currently selected lines. It replaces the lines with a call to the new function or method, passing in needed arguments and returning any values needed in the calling block of code.

To start an extract operation, select the lines to be extract in the editor and then select **Extract Function/Method** from the **Refactor** menu or from the **Refactor** sub-menu of the editor context menu (right-click). Wing will then display the **Refactoring** tool. To complete the operation, enter the name for the new function or method, select the scope in which to define it, and press the **Extract** button.

After it completes, the extract operation can be undone with the **Revert** button in the **Refactoring** tool.

Note that the extract operation currently cannot extract lines that contain return statements before the final line.

6.4. Introduce Variable

The introduce variable operation adds a variable that is initialized to the value of an existing expression and then replaces the expression with the new variable. To start an introduce variable operation, select an existing expression and then select **Introduce Variable** from the **Refactor** menu or from the **Refactor** sub-menu of the editor context menu (right-click). Wing will find all places the expression is used in the current scope and list them in the **Refactoring** tool. To complete the operation, enter the name for the new variable and press the **Introduce Variable** button. The name may include a dot, so a name starting with **self.** may be used to introduce an instance attribute in a method.

Note that each found match for the expression is displayed with a check box that can be deselected to omit that match from the rename operation.

After it completes, the introduce variable operation can be undone with the **Revert** button in the **Refactoring** tool.

Diff/Merge Tool

Wing IDE Professional provides single and multi-file difference and merge capabilities.

To initiate a session, use the Diff/Merge toolbar item (click to display a menu of options) or use the **Difference and Merge** menu item in the **Source** menu. You will be prompted for any file or directory names in the status area at the bottom of the IDE window. Additional sessions can be started concurrently but only one session is current at a given time. The same menus can be used to switch among multiple concurrent sessions, when there are two or more.

Once a session is started, the selected files will be displayed side by side, one annotated with **A:** and the other annotated with **B:**. Use the newly revealed toolbar items to move to the next or previous difference pair, to merge differences from one file into the other, or to terminate the session. Navigation and merging is also possible with the key bindings listed in the diff/merge menu.

In addition, a summary listing all changes is available from the diff/merge icon displayed at the top right of editors in the active session. This includes line number, change summary, and Python scope name when applicable. Selecting a change from this menu will jump to it.

The following types of difference/merge sessions are available:

- **Compare Files** -- Compare two selected files.
- **Compare Directories** -- Compare two selected directories. The **Diff/Merge tool**, which will be shown while the multi-file session is active, will display a list of files and estimated degree of difference in each file pair. Clicking on the list

will display the first difference in the selected file pair. The selection on the list will also update as you move through the difference list.

- **Compare Visible Files** -- Compare the two visible files. This is only available when two editor splits are shown and two different files are open in them.
- **Compare Buffer with Disk** -- Compare the current file and its contents on disk. This is only available when the current file has unsaved edits.
- **Compare Recent** -- This provides a sub-menu for quick access to recently performed comparisons.
- **Compare to Repository** -- When a file is in checked into one of the version control systems that Wing IDE supports, this item can be used to compare the working copy of the file with the corresponding revision in version control.

Diff/Merge Options

The Difference and Merge menu also contains two items that control the action of the diff/merge sessions:

- **Lock Scrolling** -- When this is checked, Wing keeps the scrolling position of the two files in the diff/merge session synchronized.
- **Ignore Whitespace** -- When this is checked, Wing will ignore changes that consist solely of white space (space, tab, line feed, or carriage return characters).

It is also possible to select between side by side or top/bottom orientation of the two files shown during a difference and merge session using the **Orientation** preference.

The color used in the highlights for differences can be configured with the **Diff/Merge Color** preference.

Source Code Browser

The **Source Browser**, which is available only in Wing IDE Professional, acts as an index to your source code, supporting inspection of collections of Python code from either a module-oriented or class-oriented viewpoint.

> ***Note***
>
> **Background Source Analysis**
>
> Wing IDE's source code analyzer will run in the background from the time that you open a project until all files have been analyzed. You may notice this overhead immediately after opening your project, depending on the size of your source base. Until analysis is complete, the class-oriented view within

the browser window will only include those classes that have been analyzed. This list is updated as more code is analyzed.

8.1. Display Choices

The source code browser offers three ways in which to browse your source code: All code by module, all code by class, or only the current file. These are selected from the menu at the top left of the browser.

8.1.1. Browse Project Modules

When browsing project modules, the source browser shows in alphabetical order all Python modules and packages that you have placed into your project and all modules and packages reachable by traversing the directory structure that contains your project files (including all sub-directories). The following types of items are present in this display mode, each of which is displayed with its own icon:

- Packages, which are directories that contain a number of files and a special file **__init__.py**. This file optionally contains a special variable **__all__** that lists the file-level modules Python should automatically import when the package as a whole is imported. See the Python documentation for additional information on creating packages.
- Directories found in your project that do not contain the necessary **__init__.py** file are shown as directories rather than packages.
- Python files found at any level are shown as modules.

Within each top-level package, directory, or module, the browser will display all sub-modules, sub-directories, modules, and any Python constructs. These are all labeled by generic type, including the following types:

- **class** -- an object class found in Python source
- **method** -- a class method
- **attribute** -- a class or instance attribute
- **function** -- a function defined at the top-level of a Python module
- **variable** -- a variable defined at the top-level of a Python module

The icons for these are shown in the **Options** menu in the top right of the source browser. Note that the base icons are modified in color and with arrows depending on whether they are imported or inherited, and whether they are public, semi-private, or private. This is described in more detail later.

8.1.2. Browsing Project Classes

When browsing by class, the browser shows a list of all classes found in the project. Within each class, in addition to a list of derived classes, the methods and attributes for the class are shown.

Navigation to super classes is possible by right-clicking on classes in the display.

8.1.3. Viewing Current Module

The browser can also be asked to restrict the display to only those symbols defined in the current module. This view shows all types of symbols at the top level and allows expansion to visit symbols defined in nested scopes. In this mode, the browser can be used as an index into the current editor file.

8.2. Display Filters

A number of options are available for filtering the constructs that are presented by the source code browser. These filters are available from the **Options** popup menu at the top right of the browser. They are organized into two major groups: (1) construct scope and source, and (2) construct type.

8.2.1. Filtering Scope and Source

The following distinctions of scope and source are made among the symbols that are shown in the source browser. Constructs in each category can be shown or hidden as a group using the filters in the **Options** menu:

- **Public** -- Constructs accessible to any user of a module or instance. These are names that have no leading underscores, such as **Print()** or **kMaxListLength**.
- **Semi-Private** -- Constructs intended for use only within related modules or from related or derived classes. These are names that have one leading underscore, such as **_NotifyError()** or **_gMaxCount**. Python doesn't enforce usage of these constructs, but they are helpful in writing clean, well-structured code and are recommended in the Python language style guide.
- **Private** -- Constructs intended to be private to a module or class. These are names that have two leading underscores, such as **__ConstructNameList()** or **__id_seed**. Python enforces local-only access to these constructs in class methods. See the Python documentation for details.
- **Inherited** -- Constructs inherited from a super-class.
- **Imported** -- Constructs imported into a module with an import statement.

8.2.2. Filtering Construct Type

Constructs in the source code browser window can also be shown or hidden on the basis of their basic type within the language:

- **Classes** -- Classes defined in Python source.
- **Methods** -- Methods defined within classes.
- **Attributes** -- Attributes (aka 'instance variables') of a class. Note that these can be either class-wide or per-instance, depending on whether they are defined within the class scope or only within methods of the class.
- **Functions** -- Non-object functions defined in Python source (usually at the top-level of a module or withing another function or method).
- **Variables** -- Variables defined anywhere in a module, class, function, or method. This does not include function or method parameters, which are not shown in the source browser.

8.3. Sorting the Browser Display

In all the display views, the ordering of constructs within a module or class can be controlled from the **Options** popup menu in the browser.

- **Alphabetically** -- Displays all entries in the tree in alphabetic order, regardless of type.
- **By Type** -- Sorts first by construct type, and then alphabetically.
- **In File Order** -- Sorts the contents of each scope in the same order that the symbols are defined in the source file.

8.4. Navigating the Views

To navigate source code from the browser, double click on the tree display. This will open source files to the appropriate location.

Source files opened from the browser will automatically close when browsing elsewhere, except if they are edited or if the stick pin icon in the upper right of the source area is clicked to indicate that the source file should remain open. For details on this, see Transient, Sticky, and Locked Editors.

The option **Follow Selection** may be enabled in the **Options** menu to cause the browser to open files even on a single click or as the currently selected item on the browser is changed from the keyboard.

Right-clicking on classes will present a popup menu that includes any super classes, allowing quick traversal up the class hierarchy.

8.5. Browser Keyboard Navigation

Once it has the focus, the browser tree view is navigable with the keyboard, using the up/down arrow keys, page up and page down, home/end, and by using the right arrow key on a parent to expand it, or the left arrow key to collapse a parent.

Whenever a tree row is selected, pressing enter or return will open the source view for the selected symbol in a separate window, indicating the point of definition for that symbol.

Interactive Python Shell

Wing provides an integrated Python Shell for execution of commands and experimental evaluation of expressions. The version of Python used in the Python Shell, and the environment it runs with, is configured in your project using Project Properties.

This shell runs a separate Python process that is independent of the IDE and functions without regard to the state of any running debug process. In Wing Professional, the **Debug Probe** can be used to interact in a similar way with your debug process. For details see Interactive Debug Probe.

Convenient ways to run parts of your source code in the shell include:

Copy/Paste part of a file -- Wing will automatically adjust leading indentation so the code can be executed in the shell.

Drag and Drop part of a file -- This works like Copy/Paste.

Evaluate File in Python Shell -- This command in the **Source** menu will evaluate the top level of the current file in the shell.

Evaluate Selection in Python Shell -- The command in the **Source** menu and editor's context menu (right-click) will evaluate the current selection in the shell.

Options menu This menu in the Python Shell tool contains items for evaluating the current file or selection

In the Python Shell, the **Up** and **Down** arrow keys will traverse the history of the code you have entered and the return key will either execute the code if it is complete or prompt for another line if it is not. **Ctrl-Up** and **Ctrl-Down** will move the cursor up and down and **Ctrl-Return** will insert a new line character at the cursor position.

To restart the Python Shell, select **Restart Shell** from the **Options** menu in the top right of the tool. This will terminate the external Python process and restart it, clearing and resetting the state of the shell.

To save the contents of the shell, use **Save a Copy** in the Options menu or right-click context menu. The right-click context menu also provides items for copying and pasting text in the shell.

To preload some code into the Python Shell when it is started, you can set the **PYTHONSTARTUP** environment variable, as supported by the Python Shell outside of Wing IDE.

9.1. Python Shell Auto-completion

Wing's Python Shell includes auto-completion, which can be a powerful tool for quickly finding and investigating functionality at runtime, for the purposes of code learning, or in the process of crafting new code. The Python Shell's completer is fueled by introspection of the runtime environment.

The Source Assistant will display details for the currently selected item in the auto-completer within the Python Shell. This provides quick access to the documentation and call signature of functions and methods that are being invoked.

Goto-definition will also work in the Python Shell, using a combination of live runtime state and static analysis to attempt to find the definition of the symbol or its type.

9.2. Python Shell Options

The **Options** menu in the Python Shell contains some settings that control how the Python Shell works:

- **Wrap Lines** causes the shell to wrap long output lines in the display
- **Filter history by entered prefix** controls whether the history will be filtered by the string between the prompt an the cursor. If history is filtered and **a** is entered at the prompt, the up arrow will find the most recent history item starting with **a**
- **Evaluate Whole Lines** causes Wing to round up the selection to the nearest line when evaluating selections, making it easier to select the desired range
- **Auto-restart when Evaluate File** causes Wing to automatically restart the shell before evaluating a file, so that each evaluation is made within a clean new environment.

OS Commands Tool

Wing IDE Professional includes an **OS Commands** tool that can be used to execute and interact with external commands provided by the OS or by other software, and to execute files outside of the debugger.

This is used for the **Execute** items in the **Debug** menu and Project context menu and to run any build command configured in Project Properties or Launch Configurations. It can also be used for other purposes such as integrating external commands into Wing, starting code that is debugged using **wingdbstub**, and so forth.

Adding and Editing Commands

Whenever a file is executed outside of the debugger, or when a build command is configured, these are added automatically to the OS Commands tool.

Additional items can be added with the **Options** menu's **New** commands, and any existing items can be edited or removed with the Edit and Remove items here. For details, see OS Command Properties.

Executing Commands

The **Options** menu also includes items for starting, terminating, or restarting a command, clearing the execution console, and selecting whether consoles should auto-clear each time the process is started or restarted.

For Python files, it is also possible to specify that the Python interpreter should be left active and at a prompt after the file is executed. This is done with the **Python Prompt after Execution** item in the **Options** menu.

The area below the popup menu at the top of the OS Commands tool is the console where commands are executed, where output is shown and where input can be entered for sending to the sub-process. Use the popup menu to switch between multiple running processes, or add multiple instances of the OS Commands tool to view them concurrently. The console provides a context menu (right click) for controlling the process, copy/pasting, and clearing or saving a copy of the output to a file.

Toolbox

The OS Commands Toolbox is hidden by default but can be shown with the **Show Toolbox** item in the **Options** menu. This contains the same items in the popup menu at the top of the OS Commands tool, but can be convenient for editing or removing multiple items, or quickly executing a series of commands. Right click on the list for available actions, or middle click or double click on the list to execute items.

Using Bash

To set up a bash shell running inside Wing IDE, add an OS Command with executable set to **bash -norc** and enable the **Use pseudo-TTY** and **Line mode** options. This is a fairly limited integration in that the tab key, color, and cursor movement are not supported.

10.1. OS Command Properties

Items added to the OS Commands tool can be configured to run within a particular environment using the dialog shown when the item is added from the OS Commands tool or by selecting an item and using the **Edit** item in the **Options** menu.

There are three types of OS Commands: **(1) Command Lines**, which are executed in the environment configured in the OS Command itself **(2) Python files**, which are executed in the environment configured in their File Properties, and **(3) Named**

Entry Points, which are executed in the environment configured by the selected Named Entry Point.

Shared Properties

All OS Command types share the following configurable properties:

Title -- This is the user-assigned title to use for the command. If not set, the command line or file name is shown instead.

I/O Encoding -- This is the encoding to use for text sent to and received from the sub-process.

Key Binding -- This field can be used to assign a key binding to the command. Press the keys desired while focus is in the field. Multi-key sequences may be used if pressed within a few seconds of each other. To replace an incorrect value, wait briefly before retrying your binding. To reset the value to blank (no key binding), select all text and press Backspace or Delete.

Raise OS Commands when executed -- This option causes the OS Commands tool to be shown whenever this command is executed. When disabled, the tool will not be brought to front.

Auto-save files before execution -- Enable this to automatically save any unsaved changes in open files before the command is executed.

Use pseudo TTY -- This option is only available on Linux and OS X. When set, Wing runs the subprocess in a pseudo tty and tries to (minimally) emulate how the command would work in a shell. Many of the ANSI escape sequences are not supported, but the basics should work. For some commands, adding options can help it to work better in the OS Commands tool. For example, **bash -norc** works better than **bash** if you have bash using colors, and **ipython -colors NoColor** works better than **ipython** alone.

Line mode -- This option is only available on Linux and OS X (on Windows, all I/O will be done line by line). When it is unchecked, Wing will enter raw mode and send every keystroke to the subprocess, rather than collecting input line by line. Often, but not always, when a pseudo TTY is being used then line mode should be disabled. Some experimentation may be required to determine the best settings.

Additional Properties for Command Lines

In command lines, use **$(ENV)** or **${ENV}** to insert values from the environment or from the special variables enumerated in Environment Variable Expansion. These values will be empty if undefined.

Note that the commands are executed on their own and not in a shell, so any commands that are built into the shell cannot be used here. For example, on Windows **dir** and some others are built-in commands so cannot be used directly; however, the form **cmd /c dir** will work in this case. On Linux, invoking **bash** directly may be necessary in similar cases.

The **Environment** tab provided for command lines allows specifying the *Initial Directory*, *Python Path*, and *Environment*, which act the same as the corresponding values configurable in Project Properties.

Test Execute

While editing command properties, the Test Execute button can be used to try executing with the current settings. A temporary entry is added to the OS Commands tool, and removed again after the command properties dialog is closed.

Unit Testing

The Wing IDE **Testing** tool provides a convenient way to run and debug unit tests written using the standard library's unittest module, doctests, or the nose package from http://somethingaboutorange.com/mrl/projects/nose/ For nose support, the nose package must be installed via easy_install or other mechanism; Wing does not come with nose or install it for you. Wing's nose support was tested with nose 0.10.1.

Overview

To add tests, use the **Testing** menu items. Tests can be added individually with **Add Single File** and **Add Current File** or can be added by applying a filter to the set of all files in the project, using **Add Files from Project**. For details on adding from the project, see Project Test Files.

The testing framework used by files is set through the **Default Test Framework** field on the **Testing** page of Project properties or the **Test Framework** field on the **Testing** page of File properties for individual test files.

To run tests, press the **Run Tests** button, or use one of the items in the **Testing** menu. For details, see Running Tests.

While tests are running, a jogging man icon is shown next to the test(s) in the Testing tool's list.

After the tests have finished running, the status indicator for the test will turn into a green check or red warning sign, depending on whether the test failed or succeeded. Status indicators for each file will also be set to red or green depending on whether any test failed or not. Individual test nodes may be expanded to show any output generated by the test or any exception that occurred. Exceptions may be expanded to display tracebacks.

Navigating

Double-clicking on any node or using the Goto Source option on the right-click popup menu in the testing tool's tree will display source code in the editor, if the source is available

Note that the **File Filter** field in the Testing tool can be used to subset the list of tests displayed in the tool. Restore it to blank or use the **Clear** item in its popup menu to see the entire lists of tests. This is a convenient way to find and focus on only those tests being worked on.

11.1. Project Test Files

A subset of a project's files can automatically be included in the list of test files in the Testing tool. The set of files is specified by the **Test file patterns** field on the Testing tab of the Project Properties dialog (which can also be accessed using the **Add Files from Project** menu item.

Any file matching the glob style wildcard pattern specified here is considered a test file. For details, see Wildcard Search Syntax. If the field is left empty then no project files will automatically be added.

Automatically added files may not be removed from the project tool's list except by altering the set of wild cards in the Test file patterns project attribute.

11.2. Running Tests

Tests can be run and debugged from Wing in a variety of ways. The options are:

- Run all tests in the testing tool. This is done with the **Run All Tests** item in the **Testing** menu or by selecting no tests (or all tests) in the list and pressing the **Run Tests** button.
- Run only the tests in current file open in the source editor. This is done with the **Run Tests in Current File** item in the **Testing** menu.
- Run a subset of test(s) by location of the cursor or selection in the source editor. This is done with the **Run Tests at Cursor** item in the **Testing** menu.
- Run tests that failed the last time tests were run. This is done with the **Run Failed Tests** item in the **Testing** menu.
- Run all tests that were run the last time tests were run. This is done with the **Run Tests Again** item in the **Testing** menu.

Test files and/or individual tests may also be selected in the Testing tool and run with the **Run Tests** button or using the items in the context menu (right click) on the Testing tool.

For each of these run options, there is an equivalent debug option that will run the tests in the debugger. These are in the **Debug** group of the **Testing** menu.

To stop running tests, use the **Abort Running Tests** item in the **Testing** menu or the **Abort Tests** item on the Testing tool.

To clear the previous test results from the Testing tool, use the **Clear Results** item in the right-click context menu.

11.3. Running unittest Tests From the Command Line

Wing's unittest test runner can be run from the command line and store results in an XML file that can be loaded into Wing via the **Load Test Results** item in the **Testing** menu. The test runner script is **src/testing/runners/run_unittest_xml.py** within the Wing installation directory. It should be run with the Python interpreter that should be used for the selected tests as follows:

```
/path/to/python /path/to/src/testing/runners/run_unittests_xml.py [options] -q testMo
```

Where **[options]** is replaced with any of the command line options listed below and the test specification is the test specification used when running with the standard library's **unittest** module. The test specification above consists of **testModule** is the module name (without **.py**), **className** is the test class name, and **testName** is the name of the test to run. To run all tests in a class, omit the **testName**. To run all tests in a module, omit also the **className**.

Available command line options are:

- **--directory=<dirname>**: Run in the given directory. Otherwise runs in the current directory inherited from the command line.
- **--output-file=<filename>**: Write results to the selected file. Results are written to stdout if this option is not given.
- **--append-to-file**: Append results to the file selected with the **--output-file=** option.
- **--one-module-per-process**: Run each module in a separate process space to avoid unintended interactions between the tests. Tests are still run sequentially and not concurrently.
- **--pattern=<glob filename pattern>**: Run tests in each filename matching the given glob pattern. This option may be repeated multiple times with different glob patterns. It also turns on the **--one-process-per-module** option.

Note: Only the unittest test runner supports running from the command line. The doctest, nose, and Django test runners cannot be used this way.

Debugger

Wing's debugger provides a powerful toolset for rapidly locating and fixing bugs in single-threaded or multi-threaded Python code. It supports breakpoints, stepping through code, inspecting and changing stack or module data, watch points, expression evaluation, and command shell style interaction with the paused debug process.

The debugger is built around a TCP/IP client/server design that supports launching your application not just from Wing itself but also externally, as with CGI scripts or

code running in an embedded scripting facility within a larger application. Remote (host to host) debugging is also provided.

Because the debugger core is written in optimized C, debug overhead is relatively low; however, you should expect your programs to run about 50% slower within the debugger.

12.1. Quick Start

Wing IDE can be used to debug all sorts of Python code, including scripts and stand-alone applications written with wxPython, Tkinter, PyQt, PyGTK, and pygame. Wing can also debug web CGIs including those running under mod_python, code running under Zope, Plone, Turbogears, Django, Paste/Pylons, Twisted, and code running in an embedded Python interpreter.

This section describes how to debug stand-alone scripts and applications that can be launched from within Wing IDE. If you wish to debug web CGIs within the web server, web servlets, or embedded Python scripts, please refer to Debugging Externally Launched Code and, for remote host-to-host debugging, see Remote Debugging.

Before debugging, you will need to install Python on your system if you have not already done so. Python is available from www.python.org.

To debug Python code with Wing, open up the Python file and select **Start / Continue** from the Debug menu. This will run to the first breakpoint, unhandled exception, or until the debug program completes. Select **Step Into** instead to run to the first line of code.

Use the Debug I/O tool to view your program's output, or to enter values for input to the program. If your program depends on characteristics of the Windows Console or a particular Linux/Unix shell, see External I/O Consoles for more information.

In some cases, you may also need to enter a **PYTHONPATH** and other environment values using the **Project Properties** dialog available from the Project menu. This can also be used to specify which Python executable should be used to run with your debug process. Use this if Wing IDE cannot find Python on your system or if you have more than one version of Python installed.

To set breakpoints, just click on the leftmost part of the margin next to the source code. In Wing IDE Professional, conditional and ignore-counted breakpoints are also available from the **Breakpoint Options** group in the **Debug** menu, or by right-clicking on the breakpoints margin.

12.2. Specifying Main Entry Point

Normally, Wing will start debugging in whatever file you have active in the frontmost editor. Depending on the nature of your project, you may wish to specify a file or a named entry point as the default debug entry point. This is done with **Set Current As Main Debug File** in the **Debug** menu, by right clicking on a file in the Project tool and selecting **Set As Main Debug File**, or by setting **Main Entry Point** in Project Properties.

When a main debug entry point is specified, it is used whenever you start the debugger, except when using **Debug Current File** in the **Debug** menu, or when right-clicking on an entry in the project manager and choosing the **Debug Selected** context menu item.

Note that the path to the main debug file is highlighted in red in the project window.

The main entry point defined for a project is also used by the source code analysis engine to determine the python interpreter version and Python path to use for analysis. Thus, changing this value will cause all source files in your project to be reanalyzed from scratch. See section Source Code Analysis for details.

12.2.1. Named Entry Points

Named entry points can be used to define additional debug/execute entry points into Python code. These are accessed with the **Named Entry Points...** item in the **Debug** menu, and can be debugged or executed from the **Debug Named Entry Point** and **Execute Named Entry Point** sub-menus.

The **named entry point manager** is used to create, edit, duplicate, and delete named entry points. The manager's list is initially blank. Right click on the list to create, edit, duplicate, or delete a named entry point. To rename an entry point, click on its name and type the new name.

Each named entry point may be assigned a key binding to debug it and another key binding to execute it. This is also done by right clicking in the named entry point manager.

Each named entry point defines the following fields:

Python File -- The file to launch.

Environment -- The environment to use when launching the file. This can either be the project-defined environment from Project Properties with a specified command line, or it can be a selected launch configuration.

Show this dialog before each run -- Select this to show the named entry point properties dialog before debugging or executing it. This is off by default.

12.3. Debug Properties

In some cases, you may need to set project and per-file properties from the Project manager before you can debug your code. This is done to specify Python interpreter, **PYTHONPATH**, environment variables, command line arguments, start directory, and other values associated with the debug process. For details, see Project-Wide Properties and Per-file Properties.

12.4. Setting Breakpoints

Breakpoints can be set on source code by opening the source file and clicking on the breakpoint margin to the left of a line of source code. Right-clicking on the breakpoint margin will display a context menu with additional breakpoint operations and options. In Wing IDE Professional, the **Breakpoints** tool in the **Tools** menu can be used to view, modify, or remove defined breakpoints. Alternatively, the **Debug** menu or the toolbar's breakpoint icons can be used to set or clear breakpoints at the current line of source (where the insertion cursor or selection is located).

Breakpoint Types

In Wing IDE Professional, the following types of breakpoints are available:

- **Regular** -- A regular breakpoint will always cause the debugger to stop on a given line of code, whenever that code is reached.
- **Conditional** -- A conditional breakpoint contains an expression that is evaluated each time the breakpoint is reached. The debugger will stop only if the conditional evaluates to **True** (any non-zero, non-empty, non-**None** value, as defined by Python). You may edit the condition of any existing breakpoint with the **Edit Breakpoint Condition...** item in the **Breakpoint Options** group of the **Debug** menu, by right clicking on the breakpoint, or in the **Breakpoints** tool.
- **Temporary** -- A temporary breakpoint will be removed automatically after the first time it is encountered. No record of the breakpoint is retained for future debug runs.

Breakpoint Attributes

Once breakpoints have been defined, you can operate on them in a number of ways to alter their behavior. These operations are available as menu items in the **Debug** menu, in the breakpoint margin's context menu, and from the **Breakpoints** tool:

- **Ignore Count** -- It is possible to set an ignore count for a breakpoint. In this case, the breakpoint will be ignored the given number of times, and the debugger will only stop at the breakpoint if it is encountered more than the set

number of times. The ignore count is reset to its original value with each new debug run. Use the **Breakpoint** tool to monitor the remaining number of times a breakpoint will be ignored.

- **Disable/Enable** -- Breakpoints can be temporarily disabled and subsequently re-enabled. Any disabled breakpoint will be ignored until re-enabled.

Breakpoints Tool

The **Breakpoints** tool, available in the **Tools** menu displays a list of all currently defined breakpoints. The following columns of data are provided:

- **Enabled** -- Checked if the breakpoint is enabled. The checkbox can be used to alter the breakpoint's state.
- **Location** -- The file and line number where the breakpoint is located
- **Condition** -- The conditional that must be true for the breakpoint to cause the debug process to stop (or blank if the breakpoint is not conditional). This value can be changed by clicking on it and editing it directly on the list.
- **Temporary** -- Checked if the breakpoint is a temporary (one-time) breakpoint. The checkbox can be used to alter the breakpoint's type.
- **Ignores** -- The number of times the breakpoint should be ignored before it causes the debugger to stop. This value can be changed by clicking on it and editing it directly on the list.
- **Ignores Left** -- The number of ignores left for a breakpoint, if a debug process exists.
- **Hits** -- The number of times the breakpoint has been reached in the current debug run (if any).

To visit the file and line number where a breakpoint is located, double click on it in the list or select **Show Breakpoint** from the context menu obtained by right-clicking on the surface of the **Breakpoints** tool. Additional options are also available from this context menu.

Keyboard Modifiers for Breakpoint Margin

Clicking on the breakpoint margin will toggle to insert a regular breakpoint or remove an existing breakpoint. You can also shift-click to insert a conditional breakpoint, and control-click to insert a breakpoint and set an ignore count for it.

When a breakpoint is already found on the line, shift-click will disable or enable it, control-click will set its ignore count, and shift-control-click will set or edit the breakpoint conditional.

12.5. Starting Debug

There are several ways in which to start a debug session from within Wing:

- Choose **Start / Continue** from the **Debug** menu or push the **Debug** icon in the toolbar. This will run the main debug file if one has been defined (described in Setting a Main Debug File), or otherwise the file open in the frontmost editor window. Execution stops at the first breakpoint or exception, or upon program completion.
- Choose **Step Into** from the **Debug** menu or push the **Step Into** icon in the toolbar. This will run the main debug file if one has been defined, or otherwise the file open in the frontmost editor window. Execution stops at the first line of code.
- Choose **Debug Current File** from the **Debug** menu or **Debug Selected** from the right-click popup menu on the Project tool to run a specific file regardless of whether a main debug file has been specified for your project. This will stop on the first breakpoint or exception, or upon program completion.
- Choose **Run to Cursor** from the **Debug** menu or toolbar. This will run the main debug file if one has been defined or otherwise the file open in the frontmost editor window. Execution continues until it reaches the line selected in the current source text window, until a breakpoint or exception is encountered, or until program completion.
- Use **Debug Recent** in the **Debug** menu to select a recently debugged file. This will stop on the first breakpoint or exception, or upon program completion.
- Use one of the key bindings given in the **Debug** menu.

Additional options exist for initiating a debug session from outside of Wing and for attaching to an already-running process. These are described in sections Debugging Externally Launched Code and Attaching, respectively.

Once a debug process has been started, the status indicator in the lower left of the window should change from white or grey to another color, as described in Debugger Status.

Note

Non-Standard Python Interpreters

If you are attempting to run your debug process against a non-standard version of Python, for example one that has been compiled with altered values for **Py_TRACE_REFS** or **WITH_CYCLE_GC**, or that has been altered in other ways, you may need to recompile the debugger core module. This is only possible with Wing IDE Professional, as it requires access to the source code. Please contact us for details.

12.6. Debugger Status

The debugger status indicator in the lower left of editor Windows is used to display the state of the debugger. Mousing over the bug icon shows expanded debugger status information in a tool tip. The color of the bug icon summarizes the status of the debug process, as follows:

- **White** -- There is no debug process, but Wing is listening for a connection from an externally launched process.
- **Gray** -- There is no debug process and Wing is not allowing any external process to attach.
- **Green** -- The debug process is running.
- **Yellow** -- The debug process is paused or stopped at a breakpoint.
- **Red** -- The debug process is stopped at an exception.

The current debugger status is also appended to the Debugger status group in the IDE's **Messages** tool.

12.7. Flow Control

Once the debugger is running, the following commands are available for controlling further execution of the debug program from Wing. These are accessible from the tool bar and the **Debug** menu:

- At any time, a freely running debug program can be paused with the **Pause** item in the **Debug** menu or with the pause tool bar button. This will stop at the current point of execution of the debug program.
- At any time during a debug session, the **Stop Debugging** menu item or toolbar item can be used to force termination of the debug program. This option is disabled by default if the current process was launched outside of Wing. It may be enabled for all local processes by using the **Kill Externally Launched** preference.

When stopped on a given line of code, execution can be controlled as follows from the **Debug** menu:

Step Over Instruction will step over a single instruction in Python. This may not leave the current line if it contains something like a list comprehension or single-line for loop.

Step Over Statement will step over the current statement, even if if spans more than one line or contains a looping construct like a list comprehension.

Step Over Block will step over or finish the current block of code, such as a for loop, conditional, function, or method.

Step Into will attempt to step into the next executed function on the current line of code. If there is no function or method to step into, this command acts like Step Over Instruction.

Step Out will complete execution of the current function or method and stop on the first instruction encountered after returning from the current function or method.

Continue will continue execution until the next breakpoint, exception, or program termination

Run To Cursor will run to the location of the cursor in the frontmost editor, or to the next breakpoint, exception, or program termination.

You can you also step through code using the toolbar icons. The step icon in the toolbar implements Step Over Statement.

Move Program Counter Here in the editor context menu (right-click) can be used to move the current position within the innermost stack frame in the debug process to any other valid position within the same scope. Stepping or execution will then continue with the selected line.

Attach and **Detach** (only in Wing IDE Professional) may be used to change the debugger between different debug processes. This is for advanced users and is detailed in Attaching and Detaching.

12.8. Viewing the Stack

Whenever the debug program is paused at a breakpoint or during manual stepping, the current stack is displayed in the **Call Stack** tool. This shows all program stack frames encountered between invocation of the program and the current run position. Outermost stack frames are higher up on the list.

When the debugger steps or stops at a breakpoint or exception, it selects the innermost stack frame by default. In order to visit other stack frames further up or down the stack, select them in the **Call Stack** tool. You may also change stack frames using the **Up Stack** and **Down Stack** items in the **Debug** menu, the up/down tool bar icons, the stack selector popup menus the other debugging tools.

When you change stack frames, all the tools in Wing that reference the current stack frame will be updated, and the current line of code at that stack frame is presented in an editor window.

In Wing IDE Professional, the current stack frame is also used to control evaluation context in the **Debug Probe** and **Watch** tools.

To change the type of stack display, right-click on the **Call Stack** tool and select from the options for the display and positioning of the code line excerpted from the debug process.

When an exception has occurred, a backtrace is also captured by the **Exceptions** notification tool, where it can be accessed even after the debug process has exited.

12.9. Viewing Debug Data

The Wing IDE debugger provides several ways in which to look at your debug program's data:

1. By inspecting locals and globals using the **Stack Data** tool. This area displays values for the currently selected stack frame.
2. By browsing values in all loaded modules (as determined by **sys.modules**), using the **Modules** tool.
3. By watching specific values from either of the above views (right click on values to add them to the **Watch** tool)
4. By typing expressions in the Watch tool.

> ***Note***
>
> **Values Fetched on Demand**
>
> The variable data displayed by Wing is fetched from the debug server on the fly as you navigate. Because of this, you may experience a brief delay when a change in an expansion or stack frame results in a large data transfer.
>
> For the same reason, leaving large amounts of debug data visible on screen may slow down stepping through code.

12.9.1. Stack Data View

The **Stack Data** debugger tool contains a popup menu for selecting thread (in multi-threaded processes) and accessing the current debug stack, a tree view area for browsing variable data in locals and globals, and a textual view area for inspecting large data values that are truncated on the tree display.

Simple values, such as strings and numbers, and values with a short string representation, will be displayed in the value column of the tree view area.

Strings are always contained in **""** (double quotes). Any value outside of quotes is a number or internally defined constant such as **None** or **Ellipsis**.

Integers can be displayed as decimal, hexadecimal, or octal, as controlled by the **Integer Display Mode** preference.

Complex values, such as instances, lists, and dictionaries, will be presented with an angle-bracketed type and memory address (for example, **<dict 0x80ce388>**) and can be expanded by clicking on the expansion indicator in the **Variable** column. The memory address uniquely identifies the construct. If you see the same address in two places, you are looking at two object references to the same instance.

If a complex value is short enough to be displayed in its entirety, the angle-bracketed form is replaced with its value, for example **{'a': 'b'}** for a small dictionary. These short complex values can still be expanded in the normal way.

Upon expansion of complex data, the position or name of each sub-entry will be displayed in the **Variable** column, and the value of each entry (possibly also complex values) will be displayed in the **Value** column. Nested complex values can be expanded indefinitely, even if this results in the traversal of cycles of object references.

Once you expand an entry, the debugger will continue to present that entry expanded, even after you step further or restart the debug session. Expansion state is saved for the duration of your Wing IDE session.

When the debugger encounters a long string, it will be truncated in the **Value** column. In this case, the full value of the string can be viewed in the textual display area at the bottom of the Stack Data tool, which is accessed by right-clicking on a value and selecting **Show Detail**. The contents of the detail area is updated when other items in the Stack Data tool are selected.

> ***Note***
>
> **Opaque Data**
>
> Some data types, such as those defined only within C/C++ code, or those containing certain Python language internals, cannot be transferred over the network. These are denoted with **Value** entries in the form **<opaque 0x80ce784>** and cannot be expanded further. In Wing IDE Professional you may be able to use the Debug Probe to access them (for example try typing **dir(value)**).

12.9.1.1. Popup Menu Options

Right-clicking on the surface of the Stack Data view displays a popup menu with options for navigating data structures:

- **Show/Hide Detail** -- Used to quickly show and hide the split where Wing shows expanded copies of values that are truncated on the main debug data view (click on items to show their expanded form).
- **Expand More** -- When a complex data value is selected, this menu item will expand one additional level in the complex value. Since this expands a potentially large number of values, you may experience a delay before the operation completes.
- **Collapse More** -- When a complex data value is selected, this menu item will collapse its display by one additional level.
- **Watch by ...** -- These items can be used to watch a debug data value over time, as described in Watching Values.
- **Force Reload** -- This forces Wing IDE to reload the displayed value from the debug process. This is useful in cases where Wing is showing an evaluation error or when the debug program contains instances that implement **__repr__** or similar special methods in a way that causes the value to change when subjected to repeated evaluation.

12.9.1.2. Filtering Value Display

There are a number of ways in which the variable displays can be configured:

- Wing lets you prune the variable display area by omitting all values by type, and variables or dictionary keys by name. This is done by setting the two preferences, **Omit Types** and **Omit Names**.
- You can also tell Wing to avoid probing certain values by data type. This is useful to avoid attempting expansion of data values defined in buggy extension modules, which can lead to crashing of the debug process as the debugger invokes code that isn't normally executed. This preference is also respected during introspection of the runtime state for auto-completion and other features in the IDE. To add values to avoid, set preference **Do Not Expand**.
- Wing provides control over size thresholds above which values are considered too large to move from the debug process into the variable display area. Values found to be too large are annotated as **huge** in the variable display area and cannot be expanded further. The data size thresholds are controlled with preferences **Huge List Threshold** and **Huge String Threshold**.
- By default Wing will display small items on a single line in the variable display areas, even if they are complex types like lists and maps. The size threshold used for this is controlled with preference **Line Threshold**. If you want all values to be shown uniformly, this preference should be set to **0**.

12.9.2. Watching Values

Wing can watch debug data values using a variety of techniques for tracking the value over time. In most cases, watching a value is initiated by right-clicking a value within a Stack Data view and selecting one of the Watch menu items. The value is then added to the list in the **Watch** tool and tracked by one of the following methods:

- **By Symbolic Path** - The debugger looks at the symbolic path from **locals()** or **globals()** for the currently selected stack frame, and tries to re-evaluate that path whenever the value may have changed. For example, if you define a dictionary variable called **testdict** in a function and set a value **testdict[1] = 'test'**, the watched value for **testdict[1]** would show any value for that slot of **testdict**, even if you delete **testdict** and recreate it. In other words, value tracking is independent of the life of any object instances in the data path.
- **By Direct Object Reference** - The debugger uses the object reference to the selected value to track it. If you use this mode with **testdict** as a whole, it would track the contents of that dictionary as long as it exists. If you were to reassign the variable **testdict** to another value, your zoomed out display would still show the contents of the original dictionary instance (if it still exists), rather than the new value of the variable **testdict**. In other words, the symbolic path to the value is completely disregarded and only instance identity is used to track the value. Because it's meaningless to track immutable types this way, this option is disabled or enabled according to the values you select to zoom out into a separate window.
- **By Parent Reference and Slot** - The debugger uses the object reference to the parent of the selected data slot and uses a symbolic representation of the slot within the parent in order to determine where to look for any value updates. This means that reassignment of the variable that points to the parent does not alter what is displayed in the zoomed-out view; only reassignment of the selected slot changes what is displayed by the debugger.
- **By Module Slot** - This is only available for values within a module, such as **string**, **sys.path**, or **os.environ**. The debugger uses the module name to look up the module in **sys.modules** and references the value by symbolic path. Any change in the value, even across module reloads, is reflected in the Watch view.

For any of these, if the value cannot be evaluated because it does not exist, the debugger displays **<undefined>**. This happens when the last object reference to a reference-tracked value is discarded, or if a selected symbolic path is undefined or cannot be evaluated.

The Watch tool will remember watch points across debug sessions, except those that make use of an object reference, which do not survive the debug process.

12.9.3. Evaluating Expressions

The debugger **Watch** tool can also be used to view the value of keyboard-entered expressions. These may be entered by clicking on any cell in the Watch manager's display tree and editing or entering the desired expression in the Variable column. Press enter to complete the editing session.

Only expressions that evaluate to a value may be entered. Other statements, like variable assignments, import statements, and language constructs are rejected with an error. These may only be executed using the Debug Probe.

Expressions are evaluated in the context of the current debug stack frame, so this feature is available only when the debug program has been paused or has stopped at a breakpoint or exception. This also means that the value of the same typed expression may change as you move up and down the call stack in the main debugger window.

In cases where evaluating an expression results in changing the value of local or global variables, your debug program will continue in that changed context. Whenever a value is changed as a result of expression evaluation, the updated value will be propagated into any visible debugger variable display areas because Wing IDE refetches all displayed data values after the evaluation of each expression. However, since you may not notice these changes, caution is required to avoid undesired side-effects in the debug process.

Note that breakpoints are never reached as a result of expression evaluation, and any exceptions encountered are not reported. If you need to debug an expression, use the Debug Probe where exceptions will be reported.

12.9.4. Problems Handling Values

The Wing debugger tries to handle debug data as gently as possible to avoid entering into lengthy computations or triggering errors in the debug process while it is packaging debug data for transfer. Even so, not all debug data can be shown on the display. This section describes each of the reasons why this may happen:

Wing may time out handling a value -- Large data values may hang up the debug server process during packaging. Wing tries to avoid this by carefully probing an object's size before packing it up. In some cases, this does not work and Wing will wait for the data for the duration set by the **Network Timeout** preference and then will display the variable value as **<network timeout during evaluate>**.

Wing may encounter values too large to handle -- Wing will not package and transfer large sequences, arrays or strings that exceed the size limits set by **Huge List Threshold** and **Huge String Threshold** preferences. On the debugger display, oversized sequences and arrays are annotated as **huge** and **<truncated>** is prepended to large truncated strings.

To avoid this, increase the value of the threshold preferences, but be prepared for longer data transfer times. Note that setting these values too high will cause the debugger to time out if the **Network Timeout** value isn't also increased.

An alternative available in Wing IDE Professional for viewing large data values is to enter expressions into the Watch tool or Debug Probe to view sub-parts of the data rather than tranferring the whole top-level portion of the value.

Wing may encounter errors during data handling -- Because Wing makes assignments and comparisons during packaging of debug data, and because it converts debug data into string form, it may execute special methods such as **__cmp__** and **__str__** in your code. If this code has bugs in it, the debugger may reveal those bugs at times when you would otherwise not see them.

The rare worst case scenario is crashing of the debug process if flawed C or C++ extension module code is invoked. In this case, the debug session is ended.

More common, but still rare, are cases where Wing encounters an unexpected Python exception while handling a debug data value. When this happens, Wing displays the value as **<error handling value>**.

These errors are not reported as normal program errors in the Exceptions tool. However, extra output that may contain the exception being raised can be obtained by setting the **Debug Internals Log File** preference.

Stored Value Errors

Wing remembers errors it encounters on debug values and stores these in the project file. These values will not be refetched during subsequent debugging, even if Wing is quit and restarted.

To override this behavior for an individual value, use the **Force Reload** item in the right-click context menu on a data value.

To clear the list of all errors previously encountered so that all values are reloaded, use the **Clear Stored Value Errors** item in the **Debug** menu. This operates only on the list of errors known for the current debug file, if a debug session is active, or for the main debug file, if any, when no debug process is running.

12.10. Debug Process I/O

While running under the Wing debugger, any output from **print** or any writes to **stdout** or **stderr** will be seen in the **Debug I/O** tool. This is also where you enter keyboard input, if your debug program requests any with **input()** or **raw_input()** or by reading from **stdin**.

The code that services debug process I/O does two things: (1) any waits on **sys.stdin** are multiplexed with servicing of the debug network socket, so that the debug process remains responsive to Wing IDE while waiting for keyboard input, and (2) in some cases, I/O is redirected to another window.

For a debug process launched from within Wing, keyboard I/O always occurs either in the Debug I/O tool or in a new external console that is created before the debug process is started. This can be controlled as described in External I/O Consoles. Using an external console is recommended when printing very large amounts of output from a debug process.

Debug processes launched outside of Wing, using **wingdbstub**, always do their keyboard I/O through the environment from which they were launched (whether that's a console window, web server, or any other I/O environment).

When commands are typed in the Debug Probe, I/O is redirected temporarily to the Debug Probe only during the time that the command is being processed.

12.10.1. External I/O Consoles

In cases where the debug process requires specific characteristics provided by the Windows Console or specific Linux/Unix shell, or to better handle very large amounts of debug process output, you can redirect debug I/O to a new external window using the **Use External Console** preference.

The most effective way to keep the external console visible after the debug process exits is to place a breakpoint on the last line of your program. Alternatively, enable the **External Console Waits on Exit** preference. However, this can result in many external consoles being displayed at once if you do not press enter inside the consoles after each debug run.

On Linux/Unix it is possible to select which console applications will be tried for the external console by altering the **External Consoles** preference.

Windows always uses the standard DOS Console that comes with your version of Windows.

12.10.2. Disabling Debug Process I/O Multiplexing

Wing alters the I/O environment in order to make it possible to keep the debug process responsive while waiting for I/O. This code mimics the environment found outside of the debugger, so any code that uses only Python-level I/O does not need to worry about this change of environment.

There are however several cases that can affect users that bypass Python-level I/O by doing C/C++ level I/O from within an extension module:

- Any C/C++ extension module code that does standard I/O calls using the C-level **stdin** or **stdout** will bypass Wing's I/O environment (which affects only Python-level **stdin** and **stdout**). This means that waiting on **stdin** in C or C++ code will make the debug process unresponsive to Wing, causing time out and termination of the debug session if you attempt to Pause or alter breakpoints at that time. In this case, redirection of I/O to the debugger I/O tool and Debug Probe (in Wing Pro only) will also not work.

- On all platforms, calling C-level **stdin** from multiple threads in a multi-threaded program may result in altered character read order when running under the Wing debugger.
- When debugging on win32, calling C-level **stdin**, even in a single-threaded program, can result in a race condition with Wing's I/O multiplexer that leads to out-of-order character reads. This is an unavoidable result of limitations on multiplexing keyboard and socket I/O on this platform.

If you run into a problem with keyboard I/O in Wing's debugger, you should:

1. Turn off Wing's I/O multiplexer by setting the **Use sys.stdin Wrapper** preference to **False**.
2. Turn on the **Use External Console** preference (for details see External I/O Consoles)

Once that is done, I/O should work properly in the external console, but the debug process will remain unresponsive to Pause or breakpoint commands from Wing IDE whenever it is waiting for input, either at the C/C++ or Python level.

Also, in this case keyboard input invoked as a side effect of using the Debug Probe will happen through unmodified **stdin** instead of within the Debug Probe, even though command output will still appear there.

12.11. Interactive Debug Probe

The **Debug Probe** acts like the Python Shell for evaluating and executing arbitrary Python code in the context of a debug program. This acts on the current debug stack frame, and is available only when the debug program is paused.

You may use many of Wing's source editor commands and key bindings within the Debug Probe, and can use the up/down arrow keys to traverse a history of recently typed commands.

Like the Python Shell, the Debug Probe in Wing provides auto-completion and integrates with the Source Assistant so that documentation and call signatures are readily available for functions and methods that are invoked here. Goto-definition works here as well.

This makes the Debug Probe particularly useful, not just to find and understand bugs, but also in crafting and trying out new code to fix the bug.

Even when no bugs are present, the Debug Probe can be used to craft code quickly in the live context in which it is intended to work. To do this, set a breakpoint where you plan to place the code, debug until you reach that breakpoint, then work in the Debug Probe to design parts or all of your new code. The auto-completer and Source Assistant running in the live program context make navigation of unfamiliar or complex code quite easy, and can greatly speed up the design and implementation of new features for existing code.

Conditional breakpoints are a natural companion for the Debug Probe. Setting a conditional breakpoint makes it easier to isolate one iteration or invocation out of many, thus isolating either a problematic case for which a bug fix is needed, or a particular case for which a new feature is desired.

In the Debug Probe, the **Up** and **Down** arrow keys will traverse the history of the code you have entered and the return key will either execute the code if it is complete or prompt for another line if it is not. **Ctrl-Up** and **Ctrl-Down** will move the cursor up and down and **Ctrl-Return** will insert a new line character at the cursor position.

12.11.1. Managing Program State

If commands you type change any local, instance, or global data values, cause modules to be loaded or unloaded, set environment variables, or otherwise alter the run environment, your debug program will continue within that altered state. All visible variable display views are also updated after each line entered in the Debug Probe in order to reflect any changes caused by your commands. Since you may not notice these changes, caution is needed to avoid creating undesired side-effects in the running debug program.

Note that breakpoints are never reached as a result of entries typed into the Debug Probe, and any exceptions are reported only after the fact. This means that activity in the Debug Probe has no effect on the debug run position or stack, even though an exception location in source code may in some cases be displayed.

12.11.2. Debug Probe Options

The **Options** menu in the Debug Probe provides the following choices:

- **Clear** -- Clear previous text from the shell.
- **Filter history by entered prefix** -- controls whether the history will be filtered by the string between the prompt an the cursor. If history is filtered and **a** is entered at the prompt, the up arrow will find the most recent history item starting with **a**
- **Save a Copy** -- Save a copy of the shell to a disk file.
- **Wrap Lines** -- Toggle whether or not long lines are wrapped in the display.

Preference **Raise Source from Tools** can be used to determine whether source code windows will be raised when exceptions occur in the Debug Probe.

12.12. Debugging Multi-threaded Code

Wing's debugger can debug multi-threaded code, as well as single-threaded code. By default, Wing will debug all threads and will stop all threads if a single thread stops. If multiple threads are present in the debug process, the Stack Data tool (and in Wing Pro the Debug Probe and Watch tools) will add a thread selector popup to the stack selector.

Even though Wing tries to stop all threads, some may continue running if they do not enter any Python code. In that case, the thread selector will list the thread as running. It also indicates which thread was the first one to stop.

When moving among threads in a multi-threaded program, the Show Position icon shown in the toolbar during debugging (between the up/down frame icons) is a convenient way to return to the original thread and stopping position.

Whenever debugging threaded code, please note that the debugger's actions may alter the order and duration that threads are run. This is a result of the small added overhead, which may influence timing, and the fact that the debugger communicates with the IDE through a TCP/IP connection.

Selecting Threads to Debug

Currently, the only way to avoid stopping all threads in the debugger is to launch your debug process from outside Wing, import wingdbstub, and use the debugger API's **SetDebugThreads()** call to specify which threads to debug. All other threads will be entirely ignored. This is documented in Debugging Externally Launched Code and the API is described in Debugger API

An example of this can be seen in the file **DebugHttpServer.py** that ships with Wing's support for Zope and Plone. To see this, unpack the WingDBG archive found inside the **zope** directory in your Wing installation.

Note, however, that specifying a subset of threads to debug may cause problems in some cases. For example, if a non-debugged thread starts running and does not return control to any other threads, then Wing's debugger will cease to respond to the IDE and the connection to the debug process will eventually be closed. This is unavoidable as there is no way to preemptively force the debug-enabled threads to run again.

12.13. Managing Exceptions

By default, Wing's debugger stops at exceptions when they would be printed by the Python interpreter or when they are logged with **logging.exception**. Wing will also stop on all **AssertionError** exceptions, whether or not they are printed or logged, since these usually indicate a program error even if they are handled.

The **Debugger > Exceptions** preference group can be used to control how Wing approaches exception reporting. This includes the following preferences.

Exception Reporting Mode

The overall strategy for identifying and reporting exceptions is configured with the **Report Exceptions** preference. The following choices are available:

When Printed (default) -- The debugger will stop on exceptions at the time that they would have been printed out by the Python interpreter.

For code with catch-all exceptions written in Python, Wing may fail to report unexpected exceptions if the handlers do not print the exception. In this case, it is best to rewrite the catch-all handlers as described in Trouble-shooting Failure to Stop on Exceptions.

In this exception handling mode, any code in **finally** clauses, **except** clauses that reraise the exception, and **with** statement cleanup routines will be executed before the debugger stops because they execute before the traceback is printed.

Always Immediately -- The debugger will stop at every single exception immediately when it is raised. In most code this will be very often, since exceptions may be used internally to handle normal, acceptible runtime conditions. As a result, this option is usually only useful after already running close to code that requires further examination.

At Process Termination -- In this case, the debugger will make a best effort to stop and report exceptions that actually lead to process termination. This occurs just before or sometimes just after the process is terminated. The exception is also printed to **stderr**, as it would be when running outside of the debugger.

When working with an Externally Launched Debug Process , the **At Process Termination** mode may not be able to stop the debug process before it exits, and in some cases may even fail to show any post-mortem traceback at all (except as printed to **stderr** in the debug process).

Similarly, when working with wxPython, PyGTK, and similar environments that include a catch-all exception handler in C/C++ code, the **At Process Termination** mode will fail to report any unexpected exceptions occurring during the main loop because those exceptions do not actually lead to process termination.

Immediately if Appear Unhandled -- The debugger will attempt to detect unhandled exceptions as they are raised in your debug process, making it possible to view the program state that led to the exception and to step through subsequently reached **finally** clauses. This is done by looking up the stack for exception handlers written in Python, and reporting only exceptions for which there is no matching handler.

> ***Note***
>
> Because of changes in the Python implementation, this mode no longer works in Python versions 2.7+ and 3.0+.

The **Immediately if Appear Unhandled** mode works well with wxPython, PyGTK, and in most other code where unexpected exceptions either lead to program termination or are handled by catch-all exception handlers written in C/C++ extension module code.

In some cases, Wing's unhandled exception detector can report normal handled exceptions that are not seen outside of the debugger. This occurs when the exceptions are handled in C/C++ extension module code. Wing can be trained to ignore these by checking the **Ignore this exception location** check box in the debugger's **Exception** tool. Ignored exceptions are still reported if they actually lead to program termination, and your selection is remembered in your project file so only needs to be made once. Use **Clear Ignored Exceptions** from the **Debug** menu at any time to reset the ignore list to blank.

Reporting Logged Exceptions

The **Report Logged Exceptions in When Printed Mode** preference controls whether exceptions that are not printed but that are logged with a call to **logging.exception** will be reported by the default **When Printed** exception reporting mode. This preference is ignored in other exception reporting modes.

Exception Type Filters

The **Never Report** and **Always Report** preferences can be used to specify that certain exception types should never be reported at all, or always reported regardless of whether they are printed or logged. For example, by default Wing will never stop on **SystemExit** or **GeneratorExit** since these occur during normal program behavior, and Wing will always stop on **AssertionError** since this usually indicates a bug in code even if it is handled.

In some code, adding **NameError** or **AttributeError** to the **Always Report** list may help uncover bugs; however, this may not work if these are treated as normal expected exceptions by the authors of the code and there are too many such cases to ignore them with the **Ignore this exception location** checkbox in the **Exceptions** tool.

12.14. Running Without Debug

Files may also be executed outside of the debugger. This can be done with any Python code, makefiles, and any other file that is marked as executable on disk. This is done with the **Execute Current File** and **Execute Recent** items in the **Debug** menu, or with **Execute Selected** after right-clicking on the project view.

Files executed in this way are run in a separate process and any input or output occurs within the OS Commands tool.

This is useful for triggering builds, executing utilities used in development, or even to launch a program that is normally launched outside of Wing and debugged using **wingdbstub.py**.

Wing can also run arbitrary command lines. See the OS Commands Tool chapter for more information on executing files or command lines from Wing.

Advanced Debugging Topics

This chapter collects documentation of advanced debugging techniques, including debugging externally launched code, and using Wing's debugger together with a debugger for C/C++ code.

See also the collection of How-Tos for tips of working with specific third party libraries and frameworks for Python.

13.1. Debugging Externally Launched Code

This section describes how to start debugging from a process that is not launched by Wing. Examples of debug code that is launched externally include CGI scripts or web servlets running under a web server and embedded Python scripts running inside a larger application.

13.1.1. Importing the Debugger

The following step-by-step instructions can be used to start debugging in externally launched code that is running on the same machine as Wing IDE:

1. Copy **wingdbstub.py** from the Wing IDE installation directory into the same directory as your debug program.
2. In some cases, you will also need to copy the file **wingdebugpw** from your User Settings Directory into the same directory as **wingdbstub.py**. This is needed when running the debug process as a different user or in a way that prevents the debug process from reading the **wingdebugpw** file from within your User Settings Directory.
3. At the point where you want debugging to begin, insert the following source code: **import wingdbstub** Depending on your code base, you may need to be cautious about whether this statement is reached by multiple processes. If

this happens, the first process will connect to Wing and the second one will fail to connect and continue running without debug. If you are debugging code in an embedded Python instance, see the notes in Debugging Embedded Python Code.

4. Make sure the Wing IDE preference **Accept Debug Connections** is turned on, to allow connection from external processes.
5. Set any required breakpoints in your Python source code.
6. Initiate the debug program from outside Wing IDE, for example with a page load in your web browser, if the program is a web app. You should see the status indicator in the lower left of the main Wing IDE window to yellow, red, or green, as described in Debugger Status. Make sure that you are running the Python interpreter without the **-O** option. The debugger will not work when optimization is turned on.
7. The debugger should stop at the first breakpoint or exception found. If no breakpoint or exception is reached, the program will run to completion, or you can use the **Pause** command in the **Debug** menu.

> ***Note***
>
> **Enabling Process Termination**
>
> In some cases, you may wish to enable termination of debug processes that were launched from outside of Wing IDE. By default, Wing recognizes externally launched processes and disables process termination in these cases unless the **Kill Externally Launched** preference is enabled.

If you have problems making this work, try setting **kLogFile** variable in **wingdbstub.py** for log additional diagnostic information.

> ***Note***
>
> **Behavior on Failure to Attach to IDE**
>
> Whenever the debugger cannot contact Wing IDE (for example, if the IDE is not running or is listening on a different port), the debug program will be run without debugging. This is useful since debug-enabled CGIs and other programs should work normally when Wing is not present. However, you can force the debug process to exit in this case by setting the **kExitOnFailure** flag in **wingdbstub.py**. To attach to processes started without debug, see Attaching (only available in Wing IDE Professional).

13.1.2. Debug Server Configuration

In some cases you may also need to alter other preset configuration values at the start of **wingdbstub.py**. These values completely replace any values set in Wing's Project or File Properties, which are relevant only when the debug program is launched from within Wing. The following options are available:

- The debugger can be disabled entirely with **kWingDebugDisabled=1**. This is equivalent to setting the **WINGDB_DISABLED** environment variable before launching the debug program.
- Set **kWingHostPort** to specify the network location of Wing IDE, so the debugger can connect to it when it starts. This is equivalent to setting the **WINGDB_HOSTPORT** environment variable before launching the debug program. The default value is **localhost:50005**. See section Remote Debugging for details if you need to change this value.
- You can control whether or not the debugger's internal error messages are written to a log file by setting **kLogFile**. Use **<stdout>**, **<stderr>**, or a file name. If the given file doesn't exist, it is created if possible. Note that using **<stderr>** may cause problems on Windows if the debug process is not running in a console. This is equivalent to setting the **WINGDB_LOGFILE** environment variable before launching the debug program (use a value of **-** to turn off logging to file).
- Set **kEmbedded** to **1** when debugging embedded scripts. In this case, the debug connection will be maintained across script invocations instead of closing the debug connection when the script finishes. When this is set to **1**, you may need to call **wingdbstub.debugger.ProgramQuit()** before your program exits, or before it discards an instance of Python, in order to cleanly close the debug connection to the IDE. This is equivalent to setting the environment variable **WINGDB_EMBEDDED**.

- Set **kAttachPort** to define the default port at which the debug process will listen for requests to attach (available in Wing IDE Professional only). This is equivalent to setting the **WINGDB_ATTACHPORT** environment variable before launching the debug program. If this value is less than 0, the debug process will never listen for attach requests. If it is greater than or equal to 0, this value is used when the debug process is running without being in contact with Wing IDE, as might happen if it initially fails to connect to the above-defined host and port, or if the IDE detaches from the process for a period of time. For Wing IDE Professional, this is described in more detail in section Attaching and Detaching.
- Set **kPWFilePath** and **kPWFileName** to define the search path and file name used to find a **wingdebugpw** file for the debugger. The environment variables **WINGDB_PWFILEPATH** and **WINGDB_PWFILENAME** will override these settings. The file path should be a Python list of strings if set in **wingdbstub.py** or a list of directories separated by the path separator (**os.pathsep**) when sent by environment variable. The string **$<winguserprofile>** may be used to specify Wing's User Settings Directory for the user that the debug process is running as. The password file name is usually **wingdebugpw** but may be changed in cases where this naming is inconvenient.
- Optionally, set **WINGHOME**, which is the Wing IDE installation directory (or the name of Wing's **.app** folder on OS X). This is set up during installation, but may need to be altered if you are running Wing from source or copied the debugger binaries over from another machine.

Setting any of the above-described environment variable equivalents will override the value given in the **wingdbstub.py** file.

13.1.3. Debugger API

A simple API can be used to control debugging more closely, once you have imported **wingdbstub.py** the first time, as was describe. This is useful in cases where you want to be able to start and stop debugging on the fly several times during a debug run, for example to avoid debug overhead except within a small sub-section of your code. It can also be useful in embedded scripting environments, particularly in those that alter the thread state or discard and recreate the Python instance across invocations.

To use the API, you must first onfigure and import **wingdbstub.py** as described in section Importing the Debugger.

High-Level API

The **wingdbstub.Ensure(require_connection=1, require_debugger=1)** function may be used to ensure the debugger is running and connected to the IDE. If **require_connection** is true, **ValueError** will be raised if a connection to the IDE

cannot be made. If **require_debugger** is true, **ValueError** will be raised if the debugger binaries cannot be found or the debugger cannot be started.

Low-Level API

After importing **wingdbstub**, the following calls may be made on **wingdbstub.debugger** to control the debugger:

- **StopDebug()** - Stop debugging completely and disconnect from Wing IDE. The debug program continues executing in non-debug mode and must be restarted to resume debugging.
- **StartDebug(stophere=0, connect=1)** -- Start debugging, optionally connecting back to the IDE and/or stopping immediately afterwards.
- **Break()** -- This pauses the free-running debug program on the current line, as if at a breakpoint.
- **ProgramQuit()** - This must be called before the debug program is exited if **kEmbedded** was set to **1** in **wingdbstub.py** or if **autoquit=0** in the preceding **StartDebug()** API call (if any). This makes sure the debug connection to the IDE is closed cleanly.
- **SetDebugThreads(threads={}, default_policy=1)** - This can be used in multi-threaded code to tell Wing's debugger which threads to debug. Pass in a dictionary that maps from thread id (as obtained from thread.get_ident()) to one of the following values: 0 to ignore the thread (do not debug it), or 1 to debug the thread and immediately stop it if any thread stops. The default_policy sets the action to take when a thread is not found in the thread map.
- **SuspendDebug()** - This will leave the connection to the debug client intact but disables the debugger so that connection overhead is avoided during subsequent execution. This should be used only to exempt a particular section of code from debug overhead. In most cases **StopDebug** is preferable.
- **ResumeDebug()** - This will resume debugging using an existing connection to Wing.

Here is a simple usage example:

```
import wingdbstub
a = 1 # This line is debugged
wingdbstub.debugger.SuspendDebug()
x = 1 # This is executed without debugging
wingdbstub.debugger.ResumeDebug()
y = 2 # This line is debugged
```

SuspendDebug() and **ResumeDebug()** can be called as many times as desired, and nested calls will be handled so that debugging is only resumed when the number of **ResumeDebug()** calls matches the number of **SuspendDebug()** calls.

13.1.4. Debugging Embedded Python Code

When Python code is run by an interpreter embedded in a larger application, you may need to craft special code to make debugging work properly.

If the host application is simply creating a single Python instance and reusing it for all script invocations, in most cases setting **kEmbedded=1** in **wingdbstub.py** will suffice.

In certain cases where the host application is manually creating or altering the thread state for each invocation of a script, you may need to use code as follows to reset the debugger and connection for each script invocation:

```
import wingdbstub
wingdbstub.Ensure()
```

In other cases where the host application uses an entirely different Python instance for each invocation, you may need to arrange that the Debugger API function **ProgramQuit** is called before each instance is destroyed and may also want to leave **kEmbedded=0** in **wingdbstub.py**. In this case you may also need to unset the environment variable **WINGDB_ACTIVE** before importing wingdbstub, as this may be left in the environment by the host application and will prevent wingdbstub from initiating debug in the second or later Python instance.

13.2. Remote Debugging

Since remote debugging is fairly complicated to configure, we currently recommend using remote display of the IDE via X Windows (Linux/Unix) or Remote Desktop (Windows) when possible, instead of setting up the IDE on a separate host from the debug process.

When this is not an option, you can also ask the debugger to connect remotely over the network. In order to do this, take the following steps (see also Remote Debugging Example):

(1) First set up Wing IDE to successfully accept connections from another process within the same machine, as described in section Importing the Debugger. You can use any Python script for testing this until you have values that work.

(2) Optionally, alter the **Server Host** preference to the name or IP address of the network interface on which the IDE listens for debug connections. The default server is **None**, which indicates that the IDE should listen on all the valid network interfaces on the host.

(3) Optionally, alter the preference **Server Port** to the TCP/IP port on which the IDE should listen for debug connections. This value may need to be changed if multiple copies of Wing IDE are running on the same host.

(4) Set the **Allowed Hosts** preference to include the host on which the debug process will be run. For security purposes, Wing will reject connections if the host isn't included here.

(5) Configure any firewall on the system that Wing IDE is running on to accept a connection on the server port from the system that the debug process will run on.

(6) Next install Wing IDE on the machine on which you plan to run your debug program. Creating an entire Wing IDE installation is the easiest approach. Alternatives are to copy only the debug server code out of a Wing installation on the same type of OS or to compile the debugger core from source code. For details, see Installing the Debugger Core.

(7) Next, transfer copies of all your debug code so that the source files are available on the host where Wing IDE will be running and at least the ***.pyc** files are available on the debug host.

During debugging, the client and server copies of your source files must match or the debugger will either fail to stop at breakpoints or stop at the wrong place, and stepping through code may not work properly.

Since there is no mechanism in Wing IDE for transferring your code, you need to use NFS, Samba, FTP or some other file sharing or file transfer mechanism to keep the remote files up to date as you edit them in Wing.

If files appear in different disk locations on the two machines, you will also need to set up a file location map, as described in File Location Maps.

(8) On your debug host, copy **wingdbstub.py** into the same directory as your source files and import it in your Python source as described in Debugging Externally Launched Code.

(9) If you didn't copy **wingdbstub.py** out of a complete installation of Wing IDE on the debug host, you will need to set **kWingHome** to match the location where you have copied the debug server code on your debug host.

(10) In **wingdbstub.py** on your debug host, set **kWingHostPort**. The host in this value must be the IP address of the machine where Wing IDE is running. The port must match the port configured with the **Server Port** preference on the host where Wing IDE is running.

(11) Then restart Wing and try running your program on the debug host. You should see the Wing IDE debugger status icon change to indicate that a debug process has attached.

If you have problems making this work, try setting **kLogFile** variable in **wingdbstub.py** for log additional diagnostic information.

13.2.1. File Location Maps

In cases where the full path to your source is not the same on both machines, you also need to set up a mapping that tells Wing where it can find your source files on each machine.

This is done with the **Location Map** preference, which lists corresponding local and remote directory locations for each remote host's dotted quad IP address.

Each host IP address in the location map is paired with one or more **(remote_prefix, local_prefix)** tuples. The remote file prefix will be a full path on the debug server's file system. The local file prefix is usually the full path of a local directory, though it may also be a file: url.

The best way to understand this is to look at the Location Map Examples.

When running Wing IDE on Windows XP, UNC formatted file names such as **\\machine\path\to\file** may be used. On other Windows systems, you must map remote drives to a drive letter such as **F:**. In cases where setting up a persistent drive mapping is a problem, use a **cmd.exe** script with a **net use** command to map the drive on demand.

Note that making symbolic links on the client or server will not work as an alternative to using this mapping. This is a side-effect of functionality in the debugger that ensures that debugging works right when symbolic links are present: Internally, source file names are always resolved to their actual full path location.

13.2.1.1. File Location Map Examples

The best way to understand location mapping is to inspect a few examples.

Defaults Explained

The default value for the **Location Map** preference contains one entry for **127.0.0.1** where the mapping is set to **None** (in Python this is represented as **{'127.0.0.1':None}**). This is equivalent to the more verbose Python representation of **{'127.0.0.1':[('/','')]}**. It converts full paths on the debug server to the client-side URLs without altering any part of the full path.

Two Linux/Unix Hosts

Here is an example setting for **debug.location-map** that would be used if running Wing on **desktop1** and debugging some code on **server1** with IP address **192.168.1.1**:

```
debug.location-map={
  '127.0.0.1':None,
  '192.168.1.1':[('/home/apache/cgi', '/svr1/home/apache/cgi')]
}
```

In this example, the files located in **/home/apache/cgi** on **server1** are the same files seen in **/server1/home/apache/cgi** on **desktop1** because the entire file system on **server1** is being shared via NFS and mounted on **desktop1** under **/svr1**.

To enter this value in Preferences, you would add **192.168.1.1** as a new Remote IP Address and a single local/remote mapping pair containing **/home/apache/cgi** and **/svr1/home/apache/cgi**.

IDE on Linux/Unix with Debug Process on Windows

If you are debugging between Windows and Linux or Unix, some care is needed in specifying the conversion paths because of the different path name conventions on each platform. The following entry would be used when running Wing IDE on a Linux/Unix host and the debug process on a Windows host with ip address **192.168.1.1**:

```
debug.location-map={
  '127.0.0.1':None,
  '192.168.1.1':[(r'e:\src', '/home/myuser/src')],
}
```

In this example the Linux/Unix directory **/home/myuser** is being shared via Samba to the Windows machine and mapped to the **e:** drive.

In the Preferences GUI, you would add **192.168.1.1** as a new Remote IP Address and a single local/remote mapping pair containing **e:\src** and **/home/myuser/src**.

IDE on Windows with Debug Process on Linux/Unix

If running Wing IDE on a Windows host and the debug process on a Linux/Unix host with IP address **192.168.1.1**, the following would be used instead for the same file locations:

```
debug.location-map={
  '127.0.0.1':None,
  '192.168.1.1':[('/home/myuser/src', 'e:/src')],
}
```

Again, note the use of forward slashes in the URL even though the file is on a Windows machine.

In the Preferences GUI, you would add **192.168.1.1** as a new Remote IP Address and a single local/remote mapping pair containing **/home/myuser/src** and **e:/src**.

Two Windows Hosts

If running Wing IDE on Windows and the debug process on another Windows machine with IP address **192.168.1.1**, the following would be used:

```
debug.location-map={
  '127.0.0.1':None,
  '192.168.1.1':[(r'c:\src', 'e:/src')],
}
```

In this case, the host where Wing is running has mapped the entire remote (debug process) host's **c:** drive to **e:**.

In the Preferences GUI, you would add **192.168.1.1** as a new Remote IP Address and a single local/remote mapping pair containing **c:\src** and **e:/src**.

Two Windows Hosts using UNC Share

A UNC style path name can be used on Windows XP as follows:

```
debug.location-map={
  '127.0.0.1':None,
  '192.168.1.1':[(r'c:\src', '\\server\share\dir')],
}
```

In this case, c:src on the remote host, where the debug process is running, can be accessed as \serversharedir on the machine where Wing IDE is running.

In the Preferences GUI, you would add **192.168.1.1** as a new Remote IP Address and a single local/remote mapping pair containing **c:\src** and **\\server\share\dir**.

13.2.2. Remote Debugging Example

Here is a simple example that enables debugging a process running on a Linux/Unix host (**192.168.1.200**) using Wing IDE running on a Windows machine (**192.168.1.210**). This example is for wingdbstub users only. If you are using the WingDBG product to debug Zope code, please refer to the Zope Debugging How-To (also included in the WingDBG control panel's Help tab).

On the Windows machine, the following preferences must be specified:

- **Accept Debug Connections** should be checked
- **Server Host** should be set to **All Interfaces** (this is the default)
- **Server Port** should be set to 50005 (this is the default)
- **Allowed Hosts** should be altered by adding **192.168.1.200**

On the Linux/Unix machine, the following value is needed in **wingdbstub.py**:

```
kWingHostPort='192.168.1.210:50005'
```

Once this is done and Wing has been restarted, you should be able to run code that imports **wingdbstub** on the Linux/Unix machine and see the debug connection establish on the Windows machine.

Then you will need to set up file sharing between the two machines (for example, using Samba) and will need to establish a location map in your Wing IDE preferences on the Windows machine.

If your source code on the Linux/Unix machine is in **/home/myuser/mysource** and you map **/home/myuser** to **e:** on the Windows machine, then you would enter this location map via the Preferences GUI by adding **192.168.1.200** as a new Remote Host IP and entering a single mapping pair with **/home/myuser/mysource** and **e:/mysource**.

See Location Map Examples for additional examples.

13.2.3. Installing the Debugger Core

When Wing is used to debug a Python program remotely, the Wing debugger core must be installed on the remote machine. The easiest way to do that is just to install Wing IDE there. If that is not possible, there are two options: (1) Copy just the debugger files from a Wing IDE installation on the same type of machine, or (2) compile the debugger core from sources (available for Wing IDE Professional only).

Copying from Wing IDE Installation

When copying from an existing Wing IDE installation on another machine, you will need to copy all of the following files and directories under the Wing IDE installation directory (on OS X, the **Contents/Resources** directory within the **.app** folder):

```
wingdbstub.py
bin/wingdb.py
bin/#.#/src/debug/tserver
bin/#.#/src.zip/debug/tserver (only Python 2.5)
bin/#.#/opensource/schannel (Python versions other than 2.5)
bin/#.#/opensource.zip/schannel (only Python 2.5)
```

Replace **#.#** with each version Python you wish to debug under (for example, **2.5**). You can omit the directories for the versions that you are not using.

The directories within zip files (used only in Python 2.5 and later) can either be copied by moving the entire zip file or by creating a subset that contains only the necessary directories.

Be sure to copy these directories from a Wing installation on the same type of host, so that on Linux/Unix you include ***.so** extension modules, on Windows ***.pyd** extension modules, and so forth.

Compiling from Source

On machines for which there is no Wing IDE installer, the debugger core can be installed from source code. This is only available to Wing IDE Professional

customers, and requires signing a non-disclosure agreement. The compilation instructions are located in **build-files/README.DBG-SRC/txt** in the source distribution that you will be provided with.

13.3. Attaching and Detaching

Debug processes normally contact Wing IDE automatically during startup. However, Wing IDE can also attach to debug processes that are already running but not yet in contact with the IDE if the process will allow it. There are two cases where this is useful:

(1) When an externally launched process (one that uses **wingdbstub.py**, as described in section Debugging Externally Launched Code) cannot reach the IDE at the configured host and port during initial startup, for example because the IDE is not yet running or was not configured to accept debug connections.

(2) When a process attached to the IDE is disconnected using **Detach from Process** in the Debug menu or the detach icon in the toolbar.

In either case, the IDE can manage any number of detached processes, allowing you to attach to any one process at a time.

13.3.1. Access Control

Wing will not allow attach/detach functionality unless it has available to it a password that can be used to control access. This is important because an unsecured debug server provides the client (Wing IDE) full control of the host machine via the Debug Probe tool. Any Python command can be executed in this way, including programs that compromise the security of your machine and network.

Because Wing sets up an access control password during installation, attach and detach will work out of the box as long as your debug processes are launched from Wing IDE, by you from the command line, or in the context of some service or program that is running under your user name on a machine that has access to your User Settings Directory.

If you plan to debug remotely, you will also need to copy the file **wingdebugpw** from your User Settings Directory into the same directory as **wingdbstub.py**.

13.3.2. Detaching

The **Detach from Process** item in the **Debug** menu is used to detach from an active debug process.

Whenever a process is detached, it continues running as if outside of the debugger, without stopping at any breakpoints or exceptions. Even if a process is paused within the debugger at time of detaching from the IDE, the process will start running actively immediately after the IDE disconnects.

13.3.3. Attaching

The **Attach to Process** item in the **Debug** menu is available whenever no other debug process is attached to the IDE. This brings up a dialog box that includes a list of available processes to attach to. The list is built from hard-wired host/port pairs given with the **Common Attach Hosts** preference, combined with known processes that were previously attached to Wing IDE.

Wing updates the list of available processes as debug sessions are terminated from the IDE, as they are seen to exit from the outside while attached to Wing, or when the process cannot be contacted by Wing.

To attach to a process, select it from the list and push the **Attach** button. You may also type in a host/port value manually if your choice is not on the list (see Identifying Foreign Processes).

Once you are attached to a process, it continues running until it reaches a breakpoint, unhandled exception, or you **Pause** it.

13.3.4. Identifying Foreign Processes

When debugging externally launched code (as described in Debugging Externally Launched Code), you may use the **kAttachPort** constant in **wingdbstub.py** to set the port on which the debug process will listen for attach requests from Wing IDE. This is useful when spawning multiple processes concurrently, or in other cases where the debug process may not be able to attached to Wing IDE as it starts up.

It is important to set unique values for the **kAttachPort** value for each concurrent, externally-launched process. If the set port is in use, a random port number will be used instead and it may be difficult to determine this number if the process cannot initially contact Wing IDE to register itself.

Once this is done, the debug process can be reached from Wing IDE by typing its host/port into the **Attach** dialog text areas. If you find yourself typing a host/port value often, it is best to add that value to the **Common Attach Hosts** preference.

See section Debugging Externally Launched Code for more information.

13.3.5. Constraints

Wing supports attaching only to a single debug process at a time. Whenever you detach from a process, it begins free-running and will not stop at any breakpoints or non-fatal exceptions. This limits what can be done with detach/attach from a single copy of Wing. If you wish to actively debug two processes at once, simultaneously controlling stepping, breakpoint activation, and execution (as in a client/server network program), you must run two copies of Wing at once.

13.4. OS X Debugging Notes

System-Provided Python

The copy of Python in **/Library/Python** on OS X does not include source files for the standard libraries, so Wing's editor will not offer autocompletion values for those modules. To work around this, use Python from within **/Library/Frameworks/Python.frameworks** instead or copy of Python installed from the standard source distribution.

MacPorts Python

At least some versions of the MacPorts packaging of Python are known not to work with Wing's debugger because it contains an **_md5** module that won't load. To work around this, use a different distribution of Python instead.

Debugging 32-bit Python on a 64-bit System

On 64-bit OS X systems, you can set up a shell script with the following contents and set it as the Python Executable in Project Properties, in order to facilitate debugging Python in 32-bit mode:

```
#!/bin/bash
arch -i386 python "$@"
```

This should only be necessary if your code needs 32-bit libraries. Wing's debugger works in either 64-bit or 32-bit mode.

13.5. Debugger Limitations

There are certain situations that the debugger cannot handle, because of the way the Python programming language works. If you are having problems getting the debugger to stop at breakpoints or to display source as you step through your code, one or more of these may apply.

Always read the Trouble-shooting Failure to Debug section first. If that fails to uncover your problem, refer to the following detailed documention of debugger limitations (many of which are extremely rare and esoteric):

(1) Your source files must be stored on disk and accessible to the IDE. If you are trying to debug code fragments, try writing them to disk temporarily and setting the **__file__** variable in the module name space before invoking Python's **exec** or **eval**. This will allow Wing's debugger to map code objects to the source you've temporarily written to disk.

(2) Running without saving will lead to incorrect display of breakpoints and run position because the debug process runs against the on-disk version of the source

file. Wing will indicate in the Messages tool and Stack Data status indicator that some files are out of **sync** so this case should only occur if you ignore its warnings.

(3) You cannot run the debug program using the **-O** or **-OO** optimization options for the Python interpreter. This removes information about line numbers and source file names, making it impossible to stop at breakpoints or step through code.

(4) There are several cases where Wing may fail to stop at breakpoints or exceptions, or may fail to find source files corresponding with breakpoints or exception points. All of these are caused by storage of incorrect file names in ***.pyc** files:

- Moving ***.pyc** files on disk after they are generated invalidates the file name stored in the file if it is a partial relative path. This happens if your **PYTHONPATH** or **sys.path** contains partial relative path names.
- A similar problem may result from use of **compileall.py** and some other utilities that don't record a correct filename in the ***.pyc** file.
- If you run the same code twice using different paths to the same working directory, as is possible on Linux/Unix with symbolic links, the file names left in ***.pyc** may contain a mix of each of these paths. If the symbolic link that was used is subsequently removed, some of the file names become invalid.

The fix for all of these problems is to remove the ***.pyc** files and let Python regenerate them from the corresponding ***.py** files with the correct file name information.

Hint: You can open ***.pyc** files in most text editors to inspect the stored file names.

(5) For code that spends much of its time in C/C++ without calling Python at all, for example as in a GUI main loop, the debugger may not reliably stop at breakpoints added during a run session, and may not respond to Pause requests. See section Debugging Non-Python Mainloops for more information.

(6) You cannot use **pdb** or other debuggers in code that you are running within the Wing debugger. The two debuggers conflict because they attempt to use the same debugger hooks in the Python interpreter.

(7) If you override **__import__** in your code, you will break the debugger's ability to stop at breakpoints unless you call the original **__import__** as part of your code whenever a module is actually imported. If you cannot call the original **__import__** for some reason, it may be possible to instead use **wingdbstub** and then call **wingdbstub.debugger.NotifyImport(mod)** from your import handler (where **mod** is the module that was just imported).

(8) If you set **__file__** in a module's name space to a value other than its original, Wing will be unable to stop at breakpoints in the module and may fail to report exceptions to the IDE's user interface.

(9) If you use an extension module to call C/C++ level **stdio** calls instead of using the Python-level facilities, the debug process will remain unresponsive to Wing IDE while waiting for keyboard input, I/O redirection to the Debug Probe in Wing Pro will fail, and you may run into out-of-order character reads in some cases. Details can be found in Debug Process I/O.

(10) Using partial path names in module **__file__** attribute can in rare cases cause Wing to fail to stop on breakpoints and exceptions, to fail to display source files, or to confuse source files of the same name.

A partial path name may end up in **__file__** only when (a) invoking Python code with a partial path name, for example with **python myfile.py** instead of **python /path/to/myfile.py**, (b) sending partial path names into **exec**, (c) using partial path names in your **PYTHONPATH** or **sys.path**, or (d) using **compileall.py** or similar tool to compile modules with a partial path name.

Because Wing does everything possible to avoid this problem in practice, it actually only occurs in the following rare cases:

- When modules are loaded with partial path names and **os.chdir()** is called before debugging is started. This is only possible when using **wingdbstub** or otherwise starting debug after your debug process is started.
- When modules are loaded with partial path names and **os.chdir()** is called after **wingdbstub.debugger.SuspendDebug()** and before **wingdbstub.debugger.ResumeDebug()**.
- When modules are loaded with partial path names and removed from **sys.modules** before the debugger is started or while debugging is suspended.
- When code objects are created on the fly using **compile()**, the C API, or the new module, a relative filename or an incorrect filename are used for the filename argument, and **os.chdir()** is called before the code is executed.

(11) Wing tries to identify when source code in the IDE matches or does not match the code that is running in the debug process. There are certain very rare cases where this will fail, which may lead to failure to stop on breakpoints and other problems even when files are identified by the IDE as being synchronized:

Using **execfile()**, **eval()**, or **exec** with a globals dict that contains **__file__** will cause Wing to incorrectly assert that the specified file has been reloaded. In practice, this scenario usually occurs when **execfile()** is called from the top level of a module, in which case the module is in fact being loaded or reloaded (so no mis-identification of module load status occurs). However, in cases where a module load takes a long time or involves a long-running loop at the top level, the **execfile()**, **eval()**, or **exec** may occur **after** edits to the module have been made and saved. In this case, Wing will mis-identify the module as having been reloaded with the new edits.

This problem can also be triggered if a **globals** with **__file__** is explicitly passed to **execfile()**, **eval()**, or **exec**. However, it will only occur in this case when the code object file name is **?**, and **locals** and **globals** dictionaries are the same, as they are by default for these calls.

(12) In very rare cases, when using the **wingdbstub.py**, if you set **sys.exitfunc** after debugging has been started, the IDE will time out on a broken network connection after the debug program exits on an exception. This only happens in some exception handling modes with exceptions that look like they will be handled because a try/except block is present that might handle the exception, but where the exception is not in the end handled and the debug program exits without calling **StopDebug()**. Work-arounds include setting **sys.exitfunc** before importing **wingdbstub.py** or adding a top-level try/except clause that always calls **StopDebug()** before exiting the debug program.

(13) Naming a file **<string>** will prevent the debugger from debugging that file because it is confused with the default file name used in Python for code that is not located in a file.

(14) The debugger may fail to step or start after stopping at a breakpoint if the floating point mode is set to single precision (24 bit) on Intel x86 and potentially other processors. This is sometimes done by graphics libraries such as DirectX or by other code that optimizes floating point calculations.

(15) When using Stackless Python, overriding **stackless.tasklet.__call__** without calling the Wing debugger's **__call__** will break the debugger.

Integrated Version Control

Wing IDE ships with integrated support for the Subversion, Mercurial, Bazaar, Git, CVS, and Perforce version control systems. Version control operations can be accessed with a menu on the main menubar, context menus in the editors and **Project** view, and a tool available from the **Tools** menu.

By default Wing auto-detects which version control system is in use for your project, based on files and directories that have been added to the project, and assigns a single active system for project-wide operations such as status, update, or commit. However, when right-clicking in the editor or **Project** view, the appropriate version control system is used even if this is different from the one defined for the project as a whole.

The name of the menu in the menu bar and the tool in the **Tools** menu changes to match the version control system that Wing is using for the project as a whole. Which version control systems will be considered for projects can be controlled by enabling or disabling each one in the **Version Control** preferences group.

Wing relies on being able to run the command line executable, such as **svn**, **git**, or **p4**, for any version control system in use. It also relies on an external ssh agent or

other security agent to help authorize version control operations. Wing does not store passwords nor does it provide a way to enter them for each operation. See Version Control Configuration for help configuring ssh or the command line executables.

Note that version control operations are directory-based, just as they are on the command line, and most operations are applied recursively to sub-directories and their files. This is true even if those sub-directories or files are not visible in the **Project** view in Wing.

14.1. Version Control Tool Panel

The version control tool panel for the active version control system can be shown by selecting it from the Tools menu or as a side effect of selecting operations from any of the version control menus.

By default, the version control tool contains a **Project Status** view that shows the status operation applied to the entire project. It summarizes which files have been modified, and can also show unregistered files when the **Show Unregistered** option in the right-click context menu is enabled.

Note that the **Project Status** operates on files in the project, and thus requires that some files or directories managed by the active version control system have been added to the project.

Operations invoked for a version control system will also display a view within the version control tool. These views display the output from the external command run to implement the operation, any input parameters, and optionally the console output for the external command. The menu in the top left can be used to switch between operations or to return to the **Project Status** view. Clicking on the **X** icon closes the view for the current operation. Operations may also be cancelled and many may be run again using the buttons in the lower right.

The **Options** menu can be used to access the version control preferences, documentation, or a console that displays the version control invocations.

14.2. Common Version Control Operations

Some operations are similiar across different version control systems and are supported in Wing by common commands. There are some variations among these from one version control system to another (for example, the **add** operation in CVS is not recursive), but there are more similarities than differences and the operations should perform as they do on the command line.

Commit

The commit operation copies changes in the local file system to the version control repository that the files are associated with. The repository might be entirely local

in distributed systems such as git or bzr or it may be on a server in centralized systems such as Subversion or CVS.

The tool shown for a commit operation has a several tabs that contain the commit message, the diffs for this commit, the list of files eligible for the commit, and the results once the commit is run. The **Files** tab may be used to select files for the commit by un-checking files that should not be committed.

The common operation **Commit Project** can be used to run the commit operation against all the files in the project.

Diff

The diff operation displays a tool with the differences between files on the local file system and files in the repository. The diff appears in the tool itself and the right-click context menu may be used to copy the diff text, goto the source for a particular section of the diff, or run the diff command again.

Status

The status operation displays a tool with the status of files in the scope of the command. The files are displayed as a tree by default, but may also be displayed as a flat list by right-clicking and selecting **View as List**. To the left of the file name, there is an icon to indicate if the file has been modified (or added or removed), has a conflict, is locked, or is not registered. Unregistered files are omitted from the status view by default. They can be shown by right-clicking on the tool and selecting **Show Unregistered**.

The common operation **Project Status** can be used to run the status operation against all the files in the project. This requires that files or directories managed by the selected version control system have alreadya been added to the project.

Log

This operation displays a list of all the revisions, with commit comments, for the files that are in the scope of the command.

Add

The add operation registers a file or directory to be added in the next commit.

Remove

The remove operation requests that a file or directory be removed in the next commit.

Revert

This operation will dispose of any local changes and revert the local files to match the current revision in the repository.

14.3. Bazaar

Wing's Bazaar support requires the **bzr** command line executable to be installed separately from Wing. Please see http://bazaar-vcs.org/ for information about Bazaar. The **bzr** executable may either be in your path or set it with the **Bazaar executable** preference in the Version Control / Bazaar preferences group.

The Bazaar support defines the following commands, in addition to those documented in Common Version Control Operations. Please see the Bazaar documentation for information on what these commands do when executed by the command line executable.

Merge Entire Branch

Merge changes in remote branch with the local branch. This command runs **bzr merge <remote>** to merge the changes.

Push Entire Branch

Push changes in local branch to remote branch. This command runs **bzr push <remote>** to push the changes.

14.4. CVS

Wing's **CVS** support requires the **cvs** command line executable to be installed separately from Wing. Please see http://www.nongnu.org/cvs/ for information about CVS. The **cvs** executable may either be in your path or set it with the **CVS executable** preference in the Version Control / CVS preferences group.

The CVS support works best if usernames and passwords are handled by another program such as **ssh-agent**, **pageant**, or another ssh agent. For details on this see Setting up SSH.

If this is not possible and you must use the obsolete **pserver** authentication mechanism, you will need to issue the **cvs login** command once from the command line before starting Wing.

CVS defines the following commands, in addition to those documented in Common Version Control Operations:

Revert

This operation will dispose of any local changes and revert the local files to match the current revision in the repository.

14.5. Git

Wing's Git support requires the **git** command line executable to be installed separately from Wing. Please see http://git-scm.com/ for information about Git. The **git** executable may either be in your path or set it with the **Git executable** preference in the Version Control / Git preferences group.

The Git support defines the following commands, in addition to those documented in Common Version Control Operations. Please see the Git documentation for information on what these commands do when executed by the command line executable.

List Branches

List all branches in local repository

Switch Branch

Switch to a different named branch. This runs **git checkout <branch>**

Fetch Repository Changes

Fetch changes from a remote repository. This runs **git fetch <remote>**

Pull Branch Changes

Pull changes on a branch from a remote repository to the local repository. This runs **git pull <remote> <branch>**

Push Branch Changes

Push changes on a branch from the local repository to a remote repository. This runs **git push <remote> <branch>**

14.6. Mercurial

Wing's Mercurial support requires the **hg** command line executable to be installed separately from Wing. Please see http://mercurial.selenic.com for information about Mercurial. The **hg** executable may either be in your path or set it with the **Mercurial executable** preference in the Version Control / Mercurial preferences group.

The Mercurial support defines the following commands, in addition to those documented in Common Version Control Operations. Please see the Mercurial documentation for information on what these commands do when executed by the command line executable.

Pull

Pull changes from a remote repository to a local one and optionally update the working directory of the local repository.

Update

Update entire working directory with changes from the local repository.

Merge

Merge changes in local repository with the working directory.

Push

Push changes in local repository to remote repository.

14.7. Perforce

Wing's **Perforce** is disabled by default and must be enabled with the **Active** preference in the Version Control / Perforce preferences group. The support also requires the **p4** command line executable to be installed separately from Wing. Please see http://www.perforce.com for information about Perforce. The **p4** executable may either be in your path or set it with the **Perforce executable** preference.

Wing finds Perforce's working directory is found by executing **p4 client -o** in the environment defined in Project Properties when a project is opened or the environment is changed. The client specification must be defined outside of Wing.

If Wing's **Project Home Directory** project property is set to a value outside of the Perforce tree, it may be necessary to add **-d pathname** (with the appropriate pathname for your configuration) to **Extra Global Arguments** in Wing's Perforce preferences.

If you usually use the Perforce GUI, you may need to start up the GUI before the environment used by the **p4** executable is set up properly.

Perforce defines the following commands, in addition to those documented in Common Version Control Operations:

Edit

Prepare the files for editing and make any editor the file is opened in writable. Note that revert on an unmodified file that's opened for editing will release the file from edit status.

14.8. Subversion

Wing's **Subversion** support requires the **svn** command line executable to be installed separately from Wing. Please see http://subversion.tigris.org/ for information about Subversion. The **svn** executable may either be in your path or set it with the **SVN executable** preference in the Version Control / SVN preferences group.

The Subversion support works best if usernames and passwords are handled by another program such as **ssh-agent**, **pageant**, or another ssh agent. For details on this see Setting up SSH.

Using SSH is preferred because there is no safe way to interact with the **svn** executable to pass it a username and password. The **--username** and **--password** command line arguments can be used, but will expose the password to anyone on the system who can list process command lines. If there is no alternative, these

can be specified in the **Extra global arguments** preference in the Version Control / SVN preferences group.

Subversion defines the following commands, in addition to those documented in Common Version Control Operations:

Revert

This operation will dispose of any local changes and revert the local files to match the current revision in the repository.

Resolved

This is used to indicate that a conflict that arose during **update** has been resolved. Files that are in conflict cannot be checked in with **commit** until the **resolved** operation is completed.

Blame/Praise

This can be used to see the revision number and author for every line in a file.

Last Revision Diff

This shows the differences for the changes that were most recently checked in for a files.

14.9. Version Control Configuration

This section provides additional information for users that have not already started using a version control system outside of Wing.

14.9.1. Configuring SSH

Most modern version control systems use SSH as a secure and convenient way to access the version control repository.

To set up SSH on Windows:

1. Install putty -- the combined installer is easiest
2. Add the location where putty is installed to your **PATH** environment variable from the Advanced tab of the System control panel.
3. Run **puttygen** and generate an SSH2 RSA key pair. Use a passphrase you will remember. Save both private and public keys to disk. Copy the contents of the key box (starting with "ssh-rsa") to **rsa-public.key** on disk.
4. Copy the **rsa-public.key** file to your server and add it to the **.ssh/authorized_keys** file under your username. E.g., use **pscp rsa-public.key user@hostname:** and then log into hostname and **cat rsa-public.key >> .ssh/authorized_keys**.
5. Run putty and enter host name in **Host Name** and **Saved Sessions** boxes then press **Save**. Go to the **Connection** category and enter your user name

on the server into the **Auto-login** username box. Go back to **Session** category and press **Save** again.

6. Run **pageant**, which adds an icon to your Windows tray. Right click and select **Add Key**. Navigate to the private key saved from **puttygen** and enter your passphrase when prompted.
7. Restart **putty**, click on the saved session, press Load, and then Open. This should open a connection to the server without prompting for any further information.

To set up SSH on Linux/Unix:

If you do not already have **openssh** and **cvs** installed, install them from packages that came with your Linux or Unix distribution.

1. If **ssh-add -l** complains that it cannot find the SSH agent, run **ssh-agent bash** (or your favorite shell). This can be skipped on most modern Linux distributions because they run the X window manager inside ssh-agent.
2. If you don't already have an ssh key in **.ssh**, issue the command **ssh-keygen -t rsa** to create a key pair in **.ssh/id_rsa** (the private key) and **.ssh/id_rsa.pub** (the public key). Enter a passphrase you will remember.
3. Copy the file **.ssh/id_rsa.pub** to your server and add it to the **.ssh/authorized_keys** file under your username. E.g., use **scp rsa-public.key user@hostname:** and then log into hostname and **cat rsa-public.key >> .ssh/authorized_keys**.
4. Back on your client (where you plan to run Wing), type **ssh-add** and enter your passphrase to get the SSH key loaded into **ssh-agent**.
5. Type **ssh user@hostname** and you should be able to log into your server without being asked for a password.

To set up SSH on OS X:

You can set up SSH on OS X in the same way as on Linux/Unix (described above). OS X automatically manages ssh keys so you will be prompted for access to the key chain as needed by the version control system.

14.9.2. Configuring Subversion

Installing Subversion

On Windows: Download from http://subversion.tigris.org/ and add installation location to **PATH** environment variable from the Advanced tab of the System control panel

On Linux/Unix: Install Subversion using the packages that came with your Linux/Unix distribution or download from http://subversion.tigris.org/ and build from sources.

Subversion with SSH

First time configuration: Install and configure SSH as described earlier (this also loads authentication information into the cache for the current session)

To check out a repository: Type **svn checkout svn+ssh://hostname/path/to/repository**. If you're not sure what to check out try this first: **svn list svn+ssh://hostname/**

Future sessions require: On Windows, double click on your private key file and enter your pass phrase, or on Linux/Unix, run **ssh-add** and enter your pass phrase.

Subversion with http/https or file URLs

To check out a repository with http or https, type **svn checkout http://hostname/path/to/repository**. If you're not sure what to check out try this first: **svn list http://hostname/**

To check out a repository with file: URLs, type **svn checkout file:///path/to/repository** You will be prompted for your user name and password, which will be cached by Subversion for future sessions.

14.9.3. Configuring CVS

Installing CVS

On Windows: Download from http://www.nongnu.org/cvs and add installation location to **PATH** environment variable from the Advanced tab of the System control panel

On Linux/Unix: Install CVS from using the packages that came with your Linux/Unix distribution or download from http://www.nongnu.org/cvs and build from sources.

Using CVS with SSH

First time configuration: Install and configure SSH as described earlier (this also loads authentication information into the cache for the current session). Then: On Windows, add **CVS_RSH=plink** to your environment from the Advanced tab of the System control panel. On Linux/Unix, add **CVS_RSH=ssh** to your environment. For example, **CVS_RSH=ssh; export CVS_RSH** on the command line, or add this to your **.bashrc** file. Note that **Environment** in your Project Properties can also be used to set **CVS_RSH** or other environment variables, however only for CVS commands issued from the IDE.

To check out a repository: Type **cvs -d :ext:username@hostname:/path/to/repository co module_name**

Future sessions require: On Windows, double click on your private key file and enter your pass phrase, or on Linux/Unix, run **ssh-add** and enter your pass phrase

Using CVS with pserver

CVS's pserver authentication mechanism is obsolete but it is still used for anonymous CVS access in some places, such as on sourceforge.net. If you are working with a pserver repository that requires a password, then you will need to issue **cvs login** once from the command line before starting Wing.

Source Code Analysis

Wing's auto-completer, source assistant, source index menu, goto-definition capability, find uses, refactoring, and other features all rely on a central engine that reads and analyzes your source code in the background as you add files to your project or alter your code in the source code editor. This engine can also load and inspect extension modules used by your code, can make use of live runtime state when available in a debug process or in the integrated Python Shell, and can read user-provided interface description files.

15.1. How Analysis Works

In analysing your source, Wing will use the Python interpreter and **PYTHONPATH** that you have specified in your **Project Properties**. If you have indicated a main debug file for your project, the values from that file's properties are used; otherwise the project-wide values are used. Whenever any of these values changes, Wing will re-analyze some or all of your source code.

You can view the Python interpreter and **PYTHONPATH** that are being used by the source code analysis engine, by selecting the Show Analysis Stats item in the Source menu. The values shown in the resulting dialog window are read-only but may be changed by pushing the Settings button. See Project-wide Properties for details on changing these values.

Be aware that if you use multiple versions of the Python interpreter or different **PYTHONPATH** values for different source files in your project, Wing will analyse all files in the project using the one interpreter version and **PYTHONPATH** it finds through the main debug file or project-wide debug properties settings. This may lead to incorrect or incomplete analysis of some source, so it is best to use only one version of Python with each Wing IDE project file.

When Wing tries to find analysis information for a particular module or file, it takes the following steps:

- The path and same directory as the referencing module are searched for an importable module
- If the module is Python code, Wing statically analyses the code to extract information from it
- If the module is an extension module, Wing looks for a ***.pi** interface description file as described later in this section

- If the module cannot be found, Wing tries to import it in a separate process space in order to analyze its contents
- If a debug process is active, Wing tries to read relevant type information from the live runtime state associated with the source code

15.2. Static Analysis Limitations

The following are known limitations affecting features based on static source analysis:

- Argument number, name, and type is not determined for functions and methods in extension modules.
- Analysis sometimes fails to identify the type of a construct because Python code doesn't always provide clues to determine the data type.
- Types of elements in lists, tuples, and dictionaries are not identified.
- Analysis information may be out of date if you edit a file externally with another editor and don't reload it in Wing. See section Auto-reloading Changed Files for reload options.
- From time to time, as Python changes, some newer Python language constructs and possible type inferencing cases are not supported.

A good way to work around these limitations, when they arise, is to place a breakpoint in the code where you are working, run to it, and then auto-completion and other information presented by the IDE will be based on the actual runtime state rather than static analysis.

See Helping Wing Analyze Code for more information.

15.3. Helping Wing Analyze Code

There are a number of ways of assistant Wing's static source analyzer in determining the type of values in Python code.

Using Live Runtime State

When a debug process is active, or when working in the **Python Shell**, Wing extracts relevant type information from the live runtime state associated with your Python code. Since this yields complete and correct type information even for code that Wing's static analysis engine cannot understand, it is often useful to run to a breakpoint before designing new code that is intended to work in that context. In the editor, the cog icon in the auto-completer indicates that type information was found in the live runtime state. In Wing IDE Professional, the **Debug Probe** can be used to immediately try out new code in the runtime environment for which it is being designed.

Using isinstance() to Assist Analysis

One way to inform the code analysis facility of the type of a variable is to add an **isinstance** call in your code. For example **isinstance(obj, CMyClass)** or **assert isinstance(obj, CMyClass)** when runtime type checking is desired. The code analyzer will pick up on these and present more complete information for the asserted values.

In cases where doing this introduces a circular import, you can use a conditional to allow Wing's static analyser to process the code without causing problems when it is executed:

```
if 0:
  import othermodule
  assert isinstance(myvariable, othermodule.COtherClass)
```

In most code, a few **isinstance** calls go a long way to making code faster and easier to edit and navigate.

Using *.pi files to Assist Analysis

Wing's source analyser can only read Python code and does not contain support for understanding C/C++ extension module code other than by attempting to import the extension module and introspecting its contents (which yields only a limited amount of information and cannot determine argument number, name, or types). Also, since Python is a dynamic language, it is possible to craft code that Wing's static analysis engine cannot understand.

For both of these cases, it is possible to create a ***.pi** (Python interface) file that describes the contents of a module. This file is simply a Python skeleton with the appropriate structure and call signature to match the functions, attributes, classes, and methods defined in a module. Wing IDE will read this file and merge its contents with any information it can obtain through static analysis or by loading an extension module. In somes cases, as for Python bindings for GUI and other toolkits, these ***.pi** files can be auto-generated from interface description files.

For a module imported as **mymodule**, the interface file is called **mymodule.pi**. Wing will search for ***.pi** files first in the same directory as it finds the Python module (or the extension module source code if it has not yet been compiled and the source code's directory is on your configured Python Path). If not found, Wing will look in the directory path set with the **Interfaces Path** preference. Next, Wing will look in the **resources/builtin-pi-files** directory within your Wing IDE installation. Finally, Wing will look in **resources/packages-pi-files**, which is used to ship some ***.pi** files for commonly used third party packages.

When searching on the interfaces path or in the **resources** directories, the top level of the directory is checked first for a matching ***.pi** file. Then, Wing tries looking in a sub-directory **#.#** named according to the major and minor version of Python being used with your source base, and subsequently in each lower major/minor version back to **2.0**.

For example, if **c:\share\pi\pi-files** is on the interfaces path and Python 2.7 is being used, Wing will check first in **c:\share\pi\pi-files**, then in **c:\share\pi\pi-files\2.7**. then in **c:\share\pi\pi-files\2.6**, and so forth.

Example ***.pi** files used by Wing internally to produce autocompletion information for builtins can be seen in the directory **resources/builtin-pi-files** inside your Wing IDE installation. This also illustrates the above-described version number fallback mechanism.

In cases where Wing cannot find a ***.pi** at all, it will attempt to load the module by name (in a separate process space) so that it can introspect its contents. The results of this operation are stored in **pi-cache** within the User Settings Directory and used subsequently. This file is regenerated only if the ***.pyd** or ***.so** for the loaded module changes.

15.4. Analysis Disk Cache

The source code analyzer writes information about files it has recently examined into the Cache Directory that is listed in Wing's About box, which is accessed from the **Help** menu.

Cache size may be controlled with the **Max Cache Size** preference However, Wing does not perform well if the space available for the cache is smaller than the space needed for a single project's source analysis information. If you see excessive sluggishness, either increase the size of the cache or disable it entirely by setting its size to 0.

If the same cache will be used by more than one computer, make sure the clocks of the two computers are synchronized. The caching mechanism uses time stamps, and may become confused if this is not done.

The analysis cache may be removed in its entirety. Wing IDE will reanalyze your code and recreate the cache as necessary.

PyLint Integration

Wing Pro provides a simple integration with pylint, which is a third party tool that runs error and warning analyses on Python code.

To use the tool, you must install **pylint** separately first and verify that it works from the command line. Note that **pylint** has certain dependencies that may be missing from your system. See the pylint website for installation details.

Once this is done and **pylint** works on the command line, bring up the **PyLint** tool from the **Tools** menu. Right click on the tool and select **Configure**. This will open a configuration file in an editor in Wing. You can alter the following here:

- **command** -- The command that invokes pylint

- **args** -- Additional command line arguments to send to pylint (see the pylint documentation for details on those available)
- **timeout** -- The maximum amount of time to wait for pylint to complete before aborting analysis.
- **autosave** -- Set this to **1** to automatically save a file before starting pylint to analyze it or **2** to auto-save all open files before starting pylint to analyze any file. **0** disables any auto-saving.

Once you have edited the configuration file as desired, save and close it.

Per-project pylintrc files can also be specified. If a file **.pylintrc** exists in the same directory as a Wing project file, then this file name is passed to pylint using the **--rcfile** argument. See the pylint documentation for details on what this file can contain.

Next, bring up some Python source code in an editor in Wing and then right click on the **PyLint** tool and select **Update**. After some time (up to a minute for larger files), lists of errors, warnings, and informational messages will be placed into the tool. Click on the items to bring up the source code with the indicated line selected.

Note that you can disable messages on the command line to pylint, as configured using the **args** item in the configuration file. See the pylint documentation for details.

Processing multiple files

The context menu on the **PyLint** tool will include an item for running **pylint** on all the files in the current package, when the current file is in a package (a directory that contains a file **__init__.py**). In this case, the file name as well as the line number is shown in the **Line** column of the output.

Note that this option adds **--output-format=parseable** to the **pylint** command line so that the file names can be obtained. This may not work with all **pylint** versions.

Using VirtualEnv on Windows

On Windows, pylint installed into a virtualenv does not work because **pylint.bat** is invoking just **python** and that may find the wrong Python installation. To fix this, edit **pylint.bat** and change **python** to the full path of the virtualenv's Python. Another fix is to edit **pylint** instead and add the following lines at the top:

```
import os
dirname = os.path.dirname(__file__)
execfile(os.path.join(dirname, 'activate_this.py'))
```

Credits

Thanks to Markus Meyer for providing the original implementation of this capability for Wing IDE. The source code for this integration is available under open source license in **scripts/pylintpanel.py** within your Wing IDE installation.

Scripting and Extending Wing IDE

Wing IDE provides an API that can be used to extend and enhance the IDE's functionality with scripts written in Python.

Simple scripts can be written without any extra tools -- Wing will find and load scripts at startup and reload them when they are edited within Wing IDE and saved to disk. The API Wing allows scripts access to the editor, debugger, project, and a range of application-level functionality. Scripts may also access all documented preferences and can issue any number of documented commands which implement functionality not duplicated in the formal Python API.

Scripts can be executed like any other command provided by Wing IDE. Scripts can add themselves to the editor and project context menus, or to new menus in the menu bar, and they can also register code for periodic execution as an idle event. They can also be bound to a key combination, or can be invoked by name using the **Command by Name** item in the **Edit** menu.

Errors encountered while loading or executing scripts are displayed in the **Scripts** channel of the **Messages** tool.

Scripts can optionally be designated as plugins, which allows the script to enable or disable itself as a whole when appropriate (for example, according to project contents or current editor file type), and allows the user to selectively enable or disable the script in the Tools menu.

More advanced scripting, including the ability to add tool panels, is also available but generally requires running a copy of Wing IDE from source code, so that scripts can be debugged more efficiently.

17.1. Scripting Example

The scripting facility is documented in detail in the sections that follow, but in most cases it is easiest simply to work from the examples in the **scripts** directory in the Wing IDE installation, using the rest of this chapter as a reference.

User scripts are usually placed inside a directory named **scripts** within the User Settings Directory. They can also be placed in **scripts** inside the Wing IDE installation.

Try adding a very simple script now by pasting the following into a file called **test.py** within one of the **scripts** directories:

```
import wingapi
def test_script(test_str):
  app = wingapi.gApplication
  v = "Product info is: " + str(app.GetProductInfo())
  v += "\nAnd you typed: %s" % test_str
  wingapi.gApplication.ShowMessageDialog("Test Message", v)
```

Then select Reload All Scripts from the Edit menu. This is only needed the first time a new script file is added, in order to get Wing to discover it. Afterward, Wing automatically reloads scripts whenever they are saved to disk.

Next execute the script with the **Command by Name** item in the **Edit** menu and then type **test-script** followed by pressing the **Enter** key in the text entry that appears at the bottom of the IDE window. Wing will ask for the argument **test_str** using it's builtin argument collection facility. Type a string and then **Enter**. The script will pop up a modal message dialog.

Next make a trivial edit to the script (e.g., change "And you typed" to "Then you typed"). Save the script and execute the script again. You will see that Wing has automatically reloaded the script and the new text appears in the message dialog.

Finally, make an edit to the script that introduces an error into it. For example, change **import wingapi** to **import wingapi2**. Save the script and Wing will show a clickable traceback in the **Scripts** channel of the **Messages** tool. This makes it easy to quickly find and fixed errors in scripts during their development.

To make life easier, you may want to create a project for your scripting work, and then add **WINGHOME/bin** to your **Python Path** in Project Properties, where **WINGHOME** is replaced with the installation location of Wing IDE or on OS X the name of the **Contents/Resources** folder inside of Wing's **.app** folder. This will make it possible for Wing to show auto-completion and call tips for items inside the module **wingapi**.

That's all there is to basic scripting. The most relevant examples for most simple scripts can be found in **editor-extensions.py** in the **scripts** directory inside the Wing IDE installation. This shows how to access and alter text in the current editor, among other things.

For more advanced scripting, where more complete debugging support is needed, you will need to obtain a copy of the Wing IDE source code distribution and run Wing from source code so that the scripts (and all of Wing) can be debugged with another copy of Wing (usually your binary installation of Wing). This is done by signing and submitting a non-disclosure agreement.

17.2. Getting Started

Scripts are Python modules or packages containing one or more Python functions. When Wing starts up, it will search all directories in the configured **Script Search Path** for modules (***.py** files) and packages (directories with an **__init__.py** file and any number of other ***.py** files or sub-packages).

Wing will load scripts defined in each file and add them to the command set that is defined internally. The script directories are traversed in the order they are given in the preference and files are loaded in alphabetical order. When multiple scripts

with the same name are found, the script that is loaded last overrides any loaded earlier under that name.

Functions in scripts are exposed as commands in Wing unless their names start with an underscore. Commands may be bound to keys, added to menus or run via **Command by Name** on the **Edit** menu.

Naming Commands

Commands can be referred to either by their short name or their fully qualified name (FQN).

The short name of a command is the same as the function name but with underscores optionally replaced by dashes (**cmdname.replace('_', '-')**).

The FQN of a command always starts with **.user.**, followed by the module name, followed by the short name.

For example, if a function named **xpext_doit** is defined inside a module named **xpext.py**, then the short name of the command created will be **xpext-doit** and the FQN will be **.user.xpext.xpext-doit**.

Reloading Scripts

Once script files have been loaded, Wing watches the files and automatically reloads them when they are edited inside Wing IDE and saved to disk. As a result, there is usually no need to restart Wing when working on a script, except when a new script file is added. In that case, Wing will not load the new script until the **reload-scripts** command (**Reload All Scripts** in the **Edit** menu) is issued or the IDE is restarted.

Reloading will not work for any file that sets **_ignore_scripts** or for modules outside of the script path. For details on how reloading works, see Advanced Scripting.

Overriding Internal Commands

Wing will not allow a script to override a command that Wing defines internally (those documented in the Command Reference). If a script is named the same as a command in Wing, it can only be invoked using its fully qualified name. This is a safeguard against completely breaking the IDE by adding a script.

One implication of this behavior is that a script may be broken if a future version of Wing ever adds a command with the same name. This can generally be avoided by using appropriately descriptive and unique names and/or by referencing the command from key bindings and menus using only its fully qualified name.

17.3. Script Syntax

Scripts are syntactically valid Python with certain extra annotations and structure that are used by Wing IDE to determine which scripts to load and how to execute them.

Only functions defined at the top level of the Python script are treated as commands, and only those that start with a letter of the alphabet. This allows the use of _ prefixed names to define utilities that are not themselves commands, and allows use of Python classes defined at the top level of script files in the implementation of script functionality.

Script Attributes

In most cases additional information about each script **def** is provided via function attributes that define the type of arguments the script expects, whether or not the command is available at any given time, the display name and documentation for the command, and the contexts in which the script should be made available in the GUI.

The following are supported:

- **arginfo** -- This defines the argument types for any arguments passed to the script. It is a dictionary from the argument name to an **ArgInfo** specification (described in more detail below) or a callable object that returns this dictionary. Argument information is used by Wing to drive automatic collection of argument values from the user. When this is missing, all arguments are treated as strings.
- **available** -- This defines whether or not the script is available. If missing, the command is always available. If set to a constant, the truth value of that constant defines availability of the script. If set to a callable object, it is invoked with the same arguments as the script itself and the return value determines availability.
- **label** -- The label to use when referring to the command in menus and elsewhere. When omitted, the label is derived from the command name by replacing underscores with a space and capitalizing each word (**cmdname.replace('_', ' ').title()**)
- **doc** -- The documentation for the script. Usually, a docstring in the function definition is used instead.
- **contexts** -- The contexts in which the script will be added in the GUI, a described in more detail below.
- **plugin_override** -- Used in scripts that are designated as plugins to indicate that a command should be enabled even if the plugin is not. It should be set to **True**.

ArgInfo

Argument information is specified using the **CArgInfo** class in the Wing API (**wingapi.py** inside **bin** in the Wing IDE installation, although the class is imported from Wing IDE's internals) and the **datatype** and **formbuilder** modules in Wing's **wingutils** package. The source code for this class and support modules is only available in the source distribution, although most use cases are covered by the following.

CArgInfo's contructor takes the following arguments:

- **doc** -- The documentation string for the argument
- **type** -- The data type, using one of the classes descended from wingutils.datatype.CTypeDef (see below for the most commonly used ones)
- **formlet** -- The GUI formlet to use to collect the argument from the user when needed. This is one of the classes descended wingutils.formbuilder.CDataGui (see below for the most commonly used ones).
- **label** -- The label to use for the argument when collected from the user. This argument may be omitted, in which case Wing builds the label as for the **label** function attribute described above.

Commonly Used Types

The following classes in **wingutils.datatype.py** cover most cases needed for scripting:

- **CBoolean** -- A boolean value. Constructor takes no arguments.
- **CType** -- A value of type matching one of the parameters sent to the constructor. For example, **CType("")** for a string, **CType(1)** for an integer, and **CType(1.0, 1)** for a float, or an integer.
- **CValue** -- One of the values passed to the constructor. For example **CValue("one", "two", "three")** to allow a value to be either **"one"**, **"two"**, or **"three"**.
- **CRange** -- A value between the first and second argument passed to the constructor. For example, **CRange(1.0, 10.0)** for a value between **1.0** and **10.0**, inclusive.

Additional types are defined in **wingutils.datatype.py**, but these are not usually needed in describing scripting arguments.

Commonly Used Formlets

The following classes in **guiutils.formbuilder.py** cover most of the data collection formlets needed for scripting:

CSmallTextGui -- A short text string entry area with optional history, auto-completion, and other options. The constructor takes the following keyword arguments, all of which are optional:

```
max_chars         -- Maximum allowed text length (-1=any, default=80)
history           -- List of strings for history (most recent 1st) or
                     a callable that will return the history (default=None)
choices           -- List of strings with all choices, or a callable
                     that will take a fragment and return all possible
                     matches (default=None)
partial_complete  -- True to only complete as far as unique match when
                     the tab key is pressed.  Default=True.
stopchars         -- List of chars to always stop partial completion.
                     Default=''
allow_only        -- List of chars allowed for input (all others are
                     not processed).  Set to None to allow all.  Default=None
auto_select_choice -- True to automatically select all of the entry text
                     when browsing on the autocompleter (so it gets erased
                     when any typing happens).  Default=False.
default           -- The default value to use.  Default=''
select_on_focus   -- True to select range on focus click; false to retain
                     pre-focus selection.  Default=False
editable          -- True to allow editing this field.  Default=True.
```

CLargeTextGui -- A longer text string. The constructor takes no arguments.

CBooleanGui -- A single checkbox for collecting a boolean value. The constructor takes no arguments.

CFileSelectorGui -- A keyboard-driven file selector with auto-completion, optional history, and option to browse using a standard file open dialog. The constructor takes the following keyword arguments:

```
want_dir          -- True to browse for a directory name (instead of a
                     file name).  Default=False.
history           -- Optional list with history of recent choices, most
                     recent first.  Default=()
default           -- The default value to use.  Default=''
```

Additional formlet types are defined in **guiutils.formbuilder.py** but these are not usually needed in collecting scripting arguments.

CPopupChoiceGui -- A popup menu to select from a range of values. The constructor takes a list of items for the popup. Each item may be one of:

```
None                -- A divider
string              -- The value.  The label used in the menu is derived:
                       label = value.replace('_', ' ').title()
(value, label)      -- The value and label to use in menu.
(value, label, tip) -- The value, label, and a tooltip to show when the
                       user hovers over the menu item.
```

CNumberGui -- A small entry area for collecting a number. The constructor takes these arguments (all are required):

```
min_value            -- The minimum value (inclusive)
max_value            -- The maximum value (inclusive)
page_size            -- Increment when scroller is used to browse the range
num_decimals         -- Number of decimal places (0 to collect an integer)
```

Additional formlets for collecting data are defined in **guiutils.formbuilder.py**, but these are not usually needed for scripting.

Magic Default Argument Values

Wing treats certain defaults values specially when they are specified for a script's arguments. When these default values are given, Wing will replace them with instances of objects defined in the API. This is a convenient way for the script to access the application, debugger, current project, current editor, and other objects in the API. All the default values are defined in the **wingapi.py** file, as are the classes they reference.

- **kArgApplication** -- The **CAPIApplication** instance (this is a singleton). Also accessible as **wingapi.gApplication**.
- **kArgDebugger** -- The currently active **CAPIDebugger**. Also accessible as **wingapi.gApplication.GetDebugger()**.
- **kArgProject** -- The currently active **CAPIProject**. Also accessible as **wingapi.gApplication.GetProject()**.
- **kArgEditor** -- The currently active **CAPIEditor**. Also accessible as **wingapi.gApplication.GetActiveEditor()**.
- **kArgDocument** -- The **CAPIDocument** for the currently active editor. Also accessible as **wingapi.gApplication.GetActiveDocument()**.

GUI Contexts

Scripts can use the **contexts** function attribute to cause Wing to automatically place the script into certain menus or other parts of the GUI. The following contexts are currently supported (they are defined in **wingapi.py**):

- **kContextEditor** -- Adds an item to the end of the editor's context menu (accessed by right clicking on the editor)
- **kContextProject** -- Adds an item to the end of the project's context menu (accessed by right clicking on the project)
- **kContextNewMenu** -- Adds an item to a new menu in the menu bar. This is a class whose constructor takes the localized name of the menu to add. The menu is only added if one or more valid scripts with that menu context are successfully loaded.

- **kContextScriptsMenu** -- Adds an item to the scripts menu, which is shown in the menu bar if any scripts are added to it (this is currently the same as **kContextNewMenu("Scripts")** but may be moved in the future).

All scripts, under both short and fully qualified name, are always listed along with all internally defined commands in the auto-completion list presented by the **Command by Name** item in the **Edit** menu, and in the **Custom Key Bindings** preference.

Top-level Attributes

Default values for some of the Script Attributes defined above can be set at the top level of the script file, and some additional attributes are also supported:

- **_arginfo** -- The default argument information to use when no per-script **arginfo** attribute is present.
- **_available** -- The default availability of scripts when no **available** attribute is present.
- **_contexts** -- The default contexts in which to add scripts when no **contexts** attribute is present.
- **_ignore_scripts** -- When set to True, Wing will completely ignore this script file.
- **_i18n_module** -- The name of the **gettext** internationalized string database to use when translating docstrings in this script. See below for more information.
- **_plugin** -- This indicates that the script is a plugin that can be selectively enabled and disabled either according to IDE state or by the user in preferences. See below for more information.

Importing Other Modules

Scripts can import other modules from the standard library, **wingapi** (the API), and even from Wing's internals. However, because of the way in which Wing loads scripts, users should avoid importing one script file into another. If this is done, the module loaded at the **import** will not be the same as the one loaded into the scripting manager. This happens because the scripting manager uniquifies the module name by prepending **internal_script_** so two entries in **sys.modules** will result. In practice, this is not always a problem except if global data at the top level of the script module is used as a way to share data between the two script modules. Be sure to completely understand Python's module loading facility before importing one script into another.

Internationalization and Localization

String literals and docstrings defined in script files can be flagged for translation using the **gettext** system. To do this, the following code should be added before any string literals are used:

```
import gettext
_ = gettext.translation('scripts_example', fallback=1).gettext
_i18n_module = 'scripts_example'
```

The string **'scripts_example'** should be replaced with the name of the **.mo** translation file that will be added to the **resources/locale** localization directories inside the Wing installation.

Subsequently, all translatable strings are passed to the **_()** function as in this code example:

```
kMenuName = _("Test Base")
```

The separate **_i18n_module** attribute is needed to tell Wing how to translate docstrings (which cannot be passed to **_()**).

Currently, the only support provided by Wing for producing the ***.po** and ***.mo** files used in the **gettext** translation system is in the build system that comes with the Wing IDE sources. Please refer to **build-files/wingide.py** and **build-files/README.txt** for details on extracting strings, merging string updates, and compiling the ***.mo** files. On Linux, KDE's **kbabel** is a good tool for managing the translations.

Plugins

When a script contains the **_plugin** attribute at the top level, it is treated as a plugin that can enable/disable itself as a whole and/or be enabled/disabled by the user in preferences.

When **_plugin** is present, it contains (name, _activator_cb) where name is the display name of the plugin and activator_cb is a function minimally defined as follows for a plugin that is always enabled:

def _activator_cb(plugin_id):

wingapi.gApplication.EnablePlugin(plugin_id, True) return True

The **_activator_cb** can also selectively enable the script by any code that accesses the Wing scripting API. For example, it could set up an instance that connects to signals in the API and calls **wingapi.gApplication.EnablePlugin()** to enable or disable itself according to project contents, file type in active editor, etc.

When a plugin is inactive, none of its commands are available and any added menus or menu items its adds to the GUI are removed. Plugins may denote particular commands as always available even when the plugin is inactive by setting the **_plugin_override** function attribute to **True**.

If the user disables a plugin in the Tools menu, this prevents loading of the plugin, and thus overrides **_activator_cb** and any **_plugin_override** attributes for the plugin.

17.4. Scripting API

Wing's formal scripting API consists of several parts:

1. The contents of the **wingapi.py** file in **bin** inside the Wing IDE installation (this file is located in **src** when working from the source distribution). Please refer to the file itself for details of the API.
2. The portions of the **wingutils.datatype** and **guiutils.formbuilder** modules that are documented in the preceding section.
3. All of the documented commands which can be invoked using the **ExecuteCommand()** method on **wingapi.gApplication**. Note keyword arguments can be passed to commands that take them, for example **ExecuteCommand('replace-string', search_string="tset", replace_string="test")**
4. All of the documented preferences which can be obtained and altered using **GetPreference** and **SetPreference** on **wingapi.gApplication**.

Scripts can, of course, also import and use standard library modules from Python, although Wing ships with a pruned subset of the standard library that includes only those modules that are used by the IDE's internals.

Advanced scripts may also "reach through" the API into Wing internals, however this requires reading Wing's souce code and no guarantee is made that these will remain unchanged or will change only in a backward compatible manner.

17.5. Advanced Scripting

While simple scripts can generally be developed from example using only the Wing IDE binary distribution, more advanced scripts require Wing to be run from the source code distribution, usually as a debug process being controlled by another copy of Wing IDE.

This provides not only more complete access to the source code for scripts that reach through the API into Wing internals, but also more complete support for debugging the scripts as they are developed.

To obtain Wing's source code, you must have a valid license to Wing IDE Professional or higher and must fill out and submit a non-disclosure agreement. Once this is done, you will be provided with access to the source code and more information on working with Wing IDE's sources.

Example

For an example of an advanced script that adds a tool panel to the IDE's interface, see **templating.py** in the **scripts** directory inside the Wing IDE installation.

How Script Reloading Works

Advanced scripters working outside of the API defined in **wingapi.py** should note that Wing only clears code objects registered through the API. For example, a script-added timeout (using **CAPIApplication.InstallTimeout()** method) will be removed and re-added automatically during reload, but a tool panel added using Wing internals will need to be removed and re-added before it updates to run on altered script code. In some cases, when object references from a script file are installed into Wing's internals, it will be necessary to restart Wing IDE.

Script files that define a global **_no_reload_scripts** will never be reloaded or unloaded. Files that define **_ignore_scripts** or that exist outside of the script path are also never reloaded.

Here is how reloading works:

1. All currently loaded script files are watched so that saving the file from an editor will cause Wing to initiate reload after it has been saved.
2. When a file changes, all scripts in its directory will be reloaded.
3. Wing removes all old scripts from the command registry, unregisters any timeouts set with **CAPIApplication.InstallTimeout()**, and removes any connections to preferences, attributes, and signals in the API.
4. Next **imp.find_module** is used to locate the module by name.
5. Then the module is removed from **sys.modules** and reloaded using **imp.find_module** and a module name that prepends **internal_script_** to the module name (in order to avoid conflicting with other modules loaded by the IDE).
6. If module load fails (for example, due to a syntax error), any timeouts or other connections registered by the module during partial load are removed and the module is removed from **sys.modules**.
7. If the module contains **_ignore_scripts**, then any timeouts or other connections are removed and scripts in the file are ignored.
8. Otherwise, Wing adds all the scripts in the module to the command registry and loads any sub-modules if the module is a package with **__init__.py**.

Note that reloading is by design slightly different than Python's builtin **reload()** function: Any old top-level symbols are blown away rather than being retained. This places some limits on what can be done with global data: For example, storing a database connection will require re-establishing the connection each time the script is reloaded.

Trouble-shooting Guide

This chapter describes what to do if you are having trouble installing or using Wing IDE.

> ***Note***
>
> We welcome feedback and bug reports, both of which can be submitted directly from Wing IDE using the **Submit Feedback** and **Submit Bug Report** items in the Help menu, or by emailing us at support at wingware.com.

18.1. Trouble-shooting Failure to Start

If you are having trouble getting Wing to start at all, read through this section for information on diagnosing the problem.

To rule out problems with a project file or preferences, try renaming your User Settings Directory and restart Wing. If this works, you can copy over files from the renamed directory one at a time to isolate the problem -- or email support at wingware dot com for help.

On Windows, the user's temporary directory sometimes becomes full, which prevents Wing from starting. Check whether the directory contains more than 65,000 files.

On Linux, OS X, or other Posix systems, in some cases when the **~/.cache** directory or the cache directory set by the **$XDG_CACHE_DIR** is located on an NFS or other remote file server, Wing can't obtain a lock on a database file. To use slower, dotfile locking set the **Use sqlite dotfile locking** preference to enabled or run Wing with the **--use-sqlite-dotfile-locking** command line option. Note that all Wing processes, regardless of the system they're running on, that use the same cache directory need to either use or not use dotfile locking.

Under a Windows terminal server, Wing may not be able to set up the environment variables it uses internally and will not start up. In this case, you can get Wing to start with the following commands:

```
set PYTHONOPTIMIZE=1
set PYTHONHOME=D:\Program Files\WingIDE\bin\PyCore
wing.exe
```

Alter **PYTHONHOME** according to the location at which you've installed Wing IDE.

On Linux with System Qt, Wing may fail to start if there is an incompatibility with the version of Qt that you have on your system. This should only occur if you are using the **--system-qt** command line option or have previoulsy changed the **Use System Qt** preference from its default value. To solve this, start Wing with the **--private-qt** command line option and restore the **Use System Qt** preference to **Use Wing's Private Qt**.

Constant Guard from Comcast can prevent Wing IDE from starting without showing any dialog or message that it is doing so.

In other cases, refer to Obtaining Diagnostic Output.

18.2. Speeding up Wing

Wing should present a responsive, snappy user interface even on relatively slow hardware. In some cases, Wing may appear sluggish:

With New Projects, the first time you set up a project file, Wing analyzes all source files for the source code browser and auto-completion facilities. During this time, the browser's class-oriented views will display only the source constructs from files of which analysis information has already been obtained. The user interface may also appear to be sluggish and Wing will consume substantial amounts of CPU time.

To avoid this in subsequent sessions, Wing stores its source analysis information to disk in a cache within your User Settings Directory.

On a multi-core virtual machine where Wing runs slowly, you may be able to improve performance by setting the processor affinity for Wing. This is done with **schedtool -a 0x1 -e wing5.0** on Linux (the schedtool package needs to be installed if not already present) and with **START /AFFINITY 01 "Wing IDE" "C:\Program Files\Wing IDE 5.0\bin\wing.exe"** on Windows. Although Wing runs on only one core, this technique has been reported to improve performance.

On OS X Mavericks, certain graphics drivers have a bug that substantially slows down Wing IDE because the OS is incorrectly detecting Wing IDE as inactive. Turning off App Nap has no effect on this, although the bug may be associated with that feature. The work-around is to put the computer to sleep briefly while Wing IDE is already running. Wing should then remain responsive until it is quit.

18.3. Trouble-shooting Failure to Debug

If you have trouble debugging with Wing IDE, select which of the following most closely describes the problem you are seeing.

18.3.1. Failure to Start Debug

Wing may fail to start the debug process in certain cases. If this happens, it often helps to try debugging a small test such as the following:

```
print("test1")
print("test2")
```

Use the **Step Into** command from the Debug menu to cause Wing IDE to attempt to run only as far as the first line of your code. This rules out possible problems caused by specific code.

Then check through the following common problems. For information on obtaining additional information from the debug sub-system, refer to the Diagnostic Output section:

Requires TCP/IP -- Wing's debugger uses a TCP/IP protocol to communicate with the IDE. Make sure that TCP/IP is installed and configured on your machine. If you are running a custom-built copy of Python, verify that the **socket** module is available.

Selecting Python Version -- If Wing says it can't find Python or if you've got multiple versions of Python on your system, make sure you've got your **Project Properties** set up to contain a valid interpreter (see Source / Show Python Environment menu item to verify that the right interpreter is being found).

Setting PYTHONPATH -- Enter any necessary **PYTHONPATH** for your debug process in Project Properties if not already defined in the environment.

Environment Conflicts -- If you set **PYTHONHOME** or **PYTHONPATH** environment variables, these may cause the debug process to fail if they do not match the particular Python interpreter that Wing is launching. You can either change the interpreter used so it matches, or unset or alter these environment values from the outside or via Project Properties from the Project menu.

- **PYTHONHOME** is a problem in all cases when it doesn't match the Python interpreter reported in the Source menu's Show Python Environment dialog.
- **PYTHONPATH** is only a problem if it contains directories that are part of a Python installation. When this doesn't match the interpreter version, this leads to import errors because Python tries to import incompatible modules.

Corrupt Python Install -- All forms of the Python binary distribution (TAR, RPM, and Windows installer) are known to have problems when a newer version of Python is installed directly over an older one on disk.

In this case, most Python programs will appear to work fine outside of Wing IDE but will not work within the Wing IDE debugger. This occurs because the debug support code uses sockets and other functionality that is not necessarily exercised by your debug program outside of the Wing debugger.

If you try to run a debug session in Wing IDE and it fails, you may be having this problem. The following test script can be used to confirm that the problem exists in your Python installation:

```
import sys
print('sys.version =', sys.version)
```

```
print('sys.executable =', sys.executable)
print('sys.version_info =', sys.version_info)
import socket
print('socket =', socket)
print('socket._socket =', socket._socket)
import select
print('select =', select)
```

To solve this problem, try uninstalling Python, manually removing any remaining files, and installing again. Or install Python into a new location on disk.

Once this is done, be sure to confirm that Wing is configured to use the new Python installation from the **Project Properties** dialog in the Project menu and that the **Show Python Environment** item in the **Source** menu displays the correct intepreter.

PyGame Full Screen Mode -- Wing's debugger is unable to debug games written with pygame when they are running in full screen mode. Use window mode instead. This is a problem also for other Python debuggers.

18.3.2. Failure to Stop on Breakpoints or Show Source Code

There are several reasons why Wing may fail to stop on breakpoints or fail to show the Python source code when the breakpoint is reached:

Missing or Incorrect Source File Names -- The most common cause of failure to stop on breakpoints or to bring up source windows while stopping or stepping through code is a mismatch between the file name that is stored in the ***.pyc** file and the actual location of the ***.py** source file.

This can be caused by (1) not saving before you run in the debugger, (2) using partial path names on PYTHONPATH or when invoking a script from the command line (the partial path stored in the ***.pyc** file may become invalid if current directory changes), (3) moving around the ***.pyc** file after they are created, or (4) using **compileall.py** to create ***.pyc** files from source. The easiest way to solve this is to use only full paths on PYTHONPATH and remove any suspect ***.pyc** files.

Concurrent Processes -- Wing may fail to stop when debugging an application that gets invoked repeatedly in separate processes, for example a CGI script invoked multiple times from a browser as part of a page load. This is because the debugger can only debug one process at a time. If the debugger is already connected to one process, the second and later processes will not be debugged and thus may miss breakpoints.

Other Problems -- Less common causes of this problem are (1) running Python with the **-O** optimization option, (2) running Python with **psyco** or other optimizer, (3) overriding the Python **__import__** routine, (4) adding breakpoints after you've started debugging an application that spends much of its time in C/C++ or other

non-Python code, and (5) on Windows, using symbolic links to directories that contain your source code files.

For more information, see the Debugger Limitations section.

18.3.3. Failure to Stop on Exceptions

Failure to stop on exceptions is most commonly caused by the same factors that can cause failure to stop on breakpoints. The rest of this section covers additional possible causes of failure to stop on exceptions.

By default, Wing only stops on exceptions for which a traceback is printed when the code is run outside of the debugger. If your code runs within a catch-all try/except clause written in Python (as in some GUI main loops or in an environment like Zope), Wing may not report all exceptions encountered in your debug process.

In some cases, altering the **Exception Reporting** preference will work. In others, it may suffice to set a breakpoint in the top-level exception handler.

An alternative is to recode your app by adding the following code to catch-all exception handlers:

```
import os, sys
if 'WINGDB_ACTIVE' in os.environ:
  sys.excepthook(*sys.exc_info())
```

The above only works with the default exception handling configuration. If you are not using the **When Printed** exception handling mode (as set by the **Report Exceptions** preference) then the above will not cause the debugger to stop. In that case, the following variant can be used instead:

```
import os

# No handler when running in Wing's debugger
if 'WINGDB_ACTIVE' in os.environ:
  dosomething()

# Handle unexpected exceptions gracefully at other times
else:
  try:
    dosomething()
  except:
    # handler here
```

Note that environments such as wxPython, PyGTK, and others include catch-all handlers for unexpected exceptions raised in the main loop, but those handlers cause the exception traceback to be printed and thus will be reported correctly by Wing without any modification to the handler.

18.3.4. Extra Debugger Exceptions

This section is only relevant if you have set the **Exception Reporting** preference to **Immediately if Appears Unhandled**.

When Wing's debugger is running in this exception handling mode, it sometimes appears to reveal bugs that are not seen when running outside of the debugger. This is a result of how this mode decides which exceptions should be shown to the user -- it is inspecting exceptions as they are raised and making decisions about whether or not the exception is unexpected or part of normal operation.

You can train Wing to ignore unwanted exception reports with the checkbox in the **Exceptions** tool.

You can also change the way Wing reports debug process exceptions with the **Exception Reporting** preference.

For more information, see Managing Exceptions.

18.4. Trouble-shooting Other Known Problems

Here are some other known problems that can affect some of Wing IDE's functionality:

Windows File Names with Spaces

When using Windows File Types or Open With to cause Python files to be opened with Wing, some versions of Windows set up the wrong command line for opening the file. You can fix this using *regedt32.exe*, *regedit.exe*, or similar tool to edit the following registry location:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Classes\Applications\wing.exe\shell\open\command
```

The problem is that the association stored there is missing quotes around the *%1* argument. It should instead read as follows:

```
"C:\Program Files\Wing IDE\bin\wing.exe" "%1" %*
```

Copy/Paste Fails on Windows

Webroot Secure Anywhere v8.0.4.66 blocks Wing IDE and Python's access to the clipboard by default so Copy/Paste will not work. The solution is to remove Wing IDE and Python from the list of applications that Webroot is denying access to the clipboard.

Failure to Find Python

Wing scans for Python at startup and may sometimes report that it could not be found even if it is on your machine.

If this happens all the time, point **Python Executable** in **Project Properties** (accessed from the **Project** menu) to your Python interpreter (python, python2.7, python.exe, etc). Wing remembers this and the message should go away, even with new projects.

If this happens only intermittently, it may be caused by high load on your machine. Try restarting Wing after load goes down. In some cases anti-virus software causes this during periods of intensive scanning.

Failure to Detect HTTP Proxy and Connect to wingware.com

Wing will try to open an http connection to **wingware.com** when you activate a license, check for product updates, or submit feedback or a bug report. If you are running in an environment with an http proxy, Wing will try to auto-detect your proxy settings. If this fails you will need to configure your proxy manually using Wing's **HTTP Proxy Server** preference. To determine the correct settings to use, ask your network administrator or see how to determine proxy settings.

18.5. Obtaining Diagnostic Output

Wing IDE and your debug code run in separate processes, each of which can independently be configured to collect additional diagnostic log information.

Diagnosing General IDE Problems

A quick way to diagnose problems seen while working with Wing IDE is to submit a bug report from the **Help** menu. Please include a description of the problem and check the **Include error log** checkbox so we can diagnose and fix the problem.

To diagnose other problems, such as failure to start, try looking at the file **error-log** in your User Settings Directory.

Alternatively, run **console_wing.exe** (on Windows) or **wing5.0 --verbose** (on Linux/Unix and OS X) from the command line to display diagnostic output.

Email this output to support at wingware.com along with your system type and version, version of Wing IDE, version of Python, and any other potentially relevant details.

Diagnosing Debugger Problems

To diagnose debugger problems, set preference **Debug Internals Log File** to a value other than **No logging** and turn on preferences **Use External Console** and **External Console Waits on Exit**. When you try again, Wing will display a debug console with diagnostics.

Alternatively, copy **wingdbstub.py** out of your Wing IDE installation, set **WINGDB_LOGFILE** environment variable to **<stderr>** or the name of a log file on disk (or alter **kLogFile** inside **wingdbstub.py**), turn on the **Accept Debug Connections** preference, and try launching the following script from the command line:

```
import wingdbstub
print("test1")
print("test2")
```

This prints diagnostic output that may be easier to capture in some cases.

Email this output to support at wingware.com. Please include also the contents of the file **error-log** in your User Settings Directory , and also your system version, version of Wing IDE, version of Python, and any other potentially relevant details.

Preferences Reference

This chapter documents the entire set of available preferences for Wing IDE Professional. Note that this includes preferences that are ignored and unused in Wing IDE Personal and Wing IDE 101.

Most preferences can be set from the **Preferences GUI** but some users may wish to build preference files manually to control different instances of Wing IDE (see details in Preferences Customization).

User Interface

Display Language

The language to use for the user interface. Either the default for this system, or set to a specific supported language.

Internal Name: **main.display-language**

Data Specification: **[None, de, en, fr]**

Default Value: **None**

Display Style

Configures the overall display style used by Wing IDE.

Internal Name: **gui.qt-display-theme**

Data Specification: **[tuple length 4 of: [None, selected-style, selected-colors], N one, [CDE, M**

Default Value: **(None, None, u'Windows', None)**

Color Palette

The overall color palette used by Wing IDE. All color preferences default to using colors from the palette, but can be overridden individually. The background colors from the palette will only apply to the editor, unless the Display Style preference is set to Match Palette. Additional palettes can be defined and added to the 'palettes' sub-directory of the User Settings directory.

Internal Name: **gui.qt-color-palette**

Data Specification: **<type str>**

Default Value: **wing-classic**

- **Layout**

Windowing Policy

Policy to use for window creation: Combined Toolbox and Editor mode places toolboxes into editor windows, Separate Toolbox mode creates separate toolbox windows, and One Window per Editor mode also creates a new window for each file opened in an editor.

Internal Name: **gui.windowing-policy**

Data Specification: **[combined-window, one-window-per-editor, separate-toolbox-window]**

Default Value: **combined-window**

Show Editor Tabs

Controls whether or not Wing shows tabs for switching between editors. When false, a popup menu is used instead.

Internal Name: **gui.use-notebook-editors**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Enable Tooltips

Controls whether or not tooltips containing help are shown when the mouse hovers over areas of the user interface.

Internal Name: **gui.enable-tooltips**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

- **Toolbar**

Show Toolbar

Whether toolbar is shown in any window.

Internal Name: **gui.show-toolbar**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Toolbar Size

Sets size of the toolbar icons. By default, adjusts according to available space.

Internal Name: **gui.toolbar-icon-size**

Data Specification: **[medium, default, xlarge, text-height, large, small]**

Default Value: **auto**

Toolbar Style

Select style of toolbar icons to use. By default, adjusts according to available space.

Internal Name: **gui.toolbar-icon-style**

Data Specification: **[medium, default, auto, xlarge, text-height, large, small]**

Default Value: **auto**

Groups Shown

Controls which groups of tools will be shown in the toolbar.

Internal Name: **guimgr.toolbar-groups**

Data Specification: **[tuple of: [search, indent, clip, bookmark, debug, vcs, proj, file, diff, test,**

Default Value: **['file', 'clip', 'search', 'diff', 'indent', 'proj', 'debug']**

Custom Items

Extra items to add to the tool bar.

Internal Name: **guimgr.toolbar-custom-items**

Data Specification: **[tuple of: [tuple length 3 of: <icon spec>, <type str>, <type str>]]**

Default Value: **()**

Primary Icon Color

Primary color for icons

Internal Name: **gui.icon-color-primary**

Data Specification: **[None or [tuple length 3 of: [from 0 to 255], [from 0 to 255], [from 0 to 255]]]**

Default Value: **None**

Secondary Icon Color

Secondary color for icons

Internal Name: **gui.icon-color-secondary**

Data Specification: **[None or [tuple length 3 of: [from 0 to 255], [from 0 to 255], [from 0 to 255]]]**

Default Value: **None**

Tertiary Icon Color

Tertiary color for icons

Internal Name: **gui.icon-color-tertiary**

Data Specification: **[None or [tuple length 3 of: [from 0 to 255], [from 0 to 255], [from 0 to 255]]]**

Default Value: **None**

Quaternary Icon Color

Quaternary color for icons

Internal Name: **gui.icon-color-quaternary**

Data Specification: **[None or [tuple length 3 of: [from 0 to 255], [from 0 to 255], [from 0 to 255]]]**

Default Value: **None**

Quinary Icon Color

Quinary color for icons

Internal Name: **gui.icon-color-quinary**

Data Specification: **[None or [tuple length 3 of: [from 0 to 255], [from 0 to 255], [from 0 to 255]]]**

Default Value: **None**

Senary Icon Color

Senary color for icons

Internal Name: **gui.icon-color-senary**

Data Specification: **[None or [tuple length 3 of: [from 0 to 255], [from 0 to 255], [from 0 to 255]]]**

Default Value: **None**

- **Fonts**

Display Font/Size

The base font and size to use for the user interface's menus and labels

Internal Name: **gui.qt-display-font**

Data Specification: **[None or <type str>]**

Default Value: **None**

Editor Font/Size

The base font and size to use for the source code editor, Python Shell, Debug Probe, Source Assistant, and other tools that display source code.

Internal Name: **edit.qt-display-font**

Data Specification: **[None or <type str>]**

Default Value: **None**

- **Keyboard**

Personality

Selects the overall editor personality, optionally to emulate another commonly used editor.

Internal Name: **edit.personality**

Data Specification: **[osx, normal, vi, eclipse, brief, emacs, visualstudio]**

Default Value: **osx**

Tab Key Action

Defines the action of the Tab key, one of: "Default for Personality" to emulate the selected Keyboard Personality. "Indent To Match" to indent the current line or selected line(s) to match the context, "Move to Next Tab Stop" to enter indentation characters so the caret reaches the next tab stop, "Indent Region" to increase the indentation of the selected line(s) by one level, or "Insert Tab Character" to insert a Tab character (chr(9)). For Python files, "Smart Tab" is an option that varies the tab key action according to the location of the caret within the line.

Internal Name: **edit.tab-key-action**

Data Specification: **[dict; keys: <type str>, values: <type str>]**

Default Value: **{'*': '--default--', 'text/x-python': '--default--'}**

Smart Tab End of Line Indents

Select type of indentation that Smart Tab will place at the end of a line.

Internal Name: **edit.smart-tab-eol-indents**

Data Specification: **[None, 1, 2, 3, 4]**

Default Value: **4**

Alt Key

Selects the key to use as the Alt- modifier in key bindings. Note that the Option key is also used to enter characters, such as ® on US keyboards or] on German keyboards. When the Option key is used for the Alt key, Alt-key bindings take precedence and thus may block entering of characters with the Option key. If both functions are needed, use the left Option key for the Alt-key and enter characters with the right Option key. If the Command keys are used for the Alt key, any Alt-key bindings will override Command-key bindings for the same key.

Internal Name: **gui.qt-osx-key-for-alt**

Data Specification: **[both-option-keys, command-keys, left-option-key, none]**

Default Value: **left-option-key**

Use Alt for Accelerators

Specifies whether plain Alt keystrokes should be used only for accelerators. When enabled, Alt-key presses that could be for an accelerator will be used only for accelerators and never for key bindings. When disabled, Alt-key bindings take precedence over accelerators. This preference is ignored when Wing is running with native OS X display style, since in that case accelerators do not exist.

Internal Name: **gui.qt-os-alt-for-accelerators**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

Custom Key Bindings

Override key bindings in the keymap. To enter the key, place focus on the entry area and type the key combination desired. The command is one of those documented in the user manual's Command Reference, or the name of any user scripts that have been loaded into the IDE. Leave the command name blank to remove the default binding for a key (this is useful when adding multi-key bindings that conflict with a default).

Internal Name: **gui.keymap-override**

Data Specification: **[dict; keys: <type str>, values: <type str>]**

Default Value: **{}**

Typing Group Timeout

Sets the timeout in seconds to use for typing, after which keys pressed are considered a separate group of characters. This is used for typing-to-select on lists and in other GUI areas. Before the timeout subsequent keys are added to previous ones to refine the selection during keyboard navigation.

Internal Name: **gui.typing-group-timeout**

Data Specification: **<type float>, <type int>**

Default Value: **1**

VI Mode Ctrl-C/X/V

Controls the behavior of the Ctrl-X/C/V key bindings in vi mode. Either always use these for cut/copy/paste, use them for vi native actions such as initiate-numeric-repeat and start-select-block, or use the default by system (clipboard on win32 and other commands elsewhere).

Internal Name: **vi-mode.clipboard-bindings**

Data Specification: **[other, clipboard, system-default]**

Default Value: **system-default**

- **Perspectives**

Auto-save Perspectives

Selects whether to auto-save perspectives when switching to another perspective. Can always auto-save, never auto-save, prompt each time a perspective is left, or auto-save as configured on a per-perspective basis.

Internal Name: **main.perspective-auto-save**

Data Specification: **[tuple length 2 of: [always, never, prompt, choose], <type str>]**

Default Value: **always**

Shared Perspective File

Selects the file to use for storing and retrieving shared perspectives. By default (when value is None) the file 'perspectives' in the user settings directory is used.

Internal Name: **main.perspective-shared-file**

Data Specification: **[one of: <type NoneType>, <type str>]**

Default Value: **None**

- **Other**

Show Splash Screen

Controls whether or not the splash screen is shown at startup.

Internal Name: **main.show-splash-screen**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Quit Application When Last Window Closes

Quit application when last document window closes

Internal Name: **guimgr.quit-on-last-window-close-osx**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

Auto-Focus Tools

Controls whether to automatically move keyboard focus from the editor to tools when they are revealed.

Internal Name: **gui.auto-focus-tools**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Case Sensitive Sorting

Controls whether names are sorted case sensitively (with all caps preceding small letters) or case insensitively

Internal Name: **gui.sort-case-sensitive**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

Auto-Show Bug Report Dialog

Whether the error bug reporting dialog (also available from the Help menu) is shown automatically when an unexpected exception is encountered inside Wing IDE.

Internal Name: **gui.show-report-error-dialog**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

Auto-check for Product Updates

Automatically attempt to connect to wingware.com to check for updates once every day after Wing is started.

Internal Name: **main.auto-check-updates**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Show Support+Upgrades Reminders

Show a reminder when Support+Upgrades for the active license is expired or will expire soon.

Internal Name: **main.monitor-support-upgrades**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Always Use Full Path in Tooltips

Enable to always show the full path of a file name in the tooltips shown from the editor tabs and file selection menus. When disabled, the configured Source Title Style is used instead.

Internal Name: **gui.full-path-in-tooltips**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

- **Advanced**

Max Error Log Size

The number of bytes at which the error log file (USER_SETTINGS_DIR/error-log) is truncated. This file can be sent to technical support to help diagnose problems with the IDE.

Internal Name: **main.max-error-log-size**

Data Specification: **[from 10000 to 10000000]**

Default Value: **500000**

Shared File Sets Repository

Selects the file to use for storing and retrieving shared named files sets. By default (when value is None) the file 'filesets' in the user settings directory is used.

Internal Name: **main.fileset-shared-file**

Data Specification: **[one of: <type NoneType>, <type str>]**

Default Value: **None**

Key Map File

Defines location of the keymap override file. Use None for default according to configured editor personality. See the Wing IDE Manual for details on building your keymap override file -- in general this is used only in development or debugging keymaps; use the keymap-override preference instead for better tracking across Wing versions.

Internal Name: **gui.keymap**

Data Specification: **[None or <type str>]**

Default Value: **None**

Projects

Auto-reopen Last Project

Controls whether most recent project is reopened at startup, in the absence of any other project on the command line.

Internal Name: **main.auto-reopen-last-project**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Close Files with Project

Controls whether any files open in an editor are also closed when a project file is closed

Internal Name: **proj.close-also-windows**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Show New Project Dialog

Whether to show New Project dialog when creating projects. When this is disabled, a blank project is created and can be configured and saved from the Project menu.

Internal Name: **proj.show-new-project-dialog**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Open Projects as Text

Controls whether project files are opened as project or as text when opened from the File menu. This does not affect opening from the Project menu.

Internal Name: **gui.open-projects-as-text**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

Confirm Drag Copy/Move

Controls whether or not the IDE will confirm file copy/move operations initiated by dragging items around on the Project view.

Internal Name: **proj.confirm-file-drags**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

- **Context Menu**

Groups Shown

Controls which groups of menu items will be shown in the Project tool's context menu.

Internal Name: **proj.context-menu-groups**

Data Specification: **[tuple of: [clip, script, vcs, nav, proj, file, debug]]**

Default Value: **['clip', 'nav', 'debug', 'vcs', 'proj', 'file', 'script']**

Custom Items

Extra menu items to add to the Project tool context menu.

Internal Name: **proj.context-menu-custom-items**

Data Specification: **[tuple of: [tuple length 2 of: <type str>, <type str>]]**

Default Value: **()**

Files

Auto-Save Files Before Debug or Execute

Controls whether or not all edited files are saved without asking before a debug run, before starting unit tests, or before a file or build process is executed.

Internal Name: **gui.auto-save-before-action**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

Default Directory Policy

Defines how Wing determines the starting directory to use when prompting for a file name: Either based on location of the resource at current focus, location of the current project home directory, the last directory visited for file selection, the current directory at startup (or selected since), or always the specific fixed directory entered here.

Internal Name: **main.start-dir-policy**

Data Specification: **[tuple length 2 of: [current-project, current-directory, recen t-directory, cu**

Default Value: **('current-focus', '')**

Title Style

Format used for titles of source files: Use Base Name Only to display just the file name, Prepend Relative Path to use partial relative path from the project file location or configured Project Home Directory, Append Relative Path to instead append the relative path after the file namePrepend Full Path to use full path, or Append Full Path to instead append the fullpath after the file name.

Internal Name: **gui.source-title-style**

Data Specification: **[append-relative, basename, prepend-fullpath, append-fullp ath, prepend-**

Default Value: **append-relative**

Default Encoding

The default encoding to use for text files opened in the source editor and other tools, when an encoding for that file cannot be determined by reading the file. Other encodings may also be tried. This also sets the encoding to use for newly created files.

Internal Name: **edit.default-encoding**

Data Specification: **[None or [Central and Eastern European iso8859-2, Japane se iso-2022-jp an, Western European cp1140, Chinese (PRC) hz, Portuguese cp860, Japane se iso-2022-jp**

Default Value: **None**

New File EOL

Default end-of-line to use. Wing matches existing line endings in non-blank files and uses this preference only when a file contains no end-of-line characters.

Internal Name: **edit.new-file-eol-style**

Data Specification: **[lf, cr, crlf]**

Default Value: **lf**

New File Extension

Default file extension for newly created files

Internal Name: **edit.new-file-extension**

Data Specification: **<type str>**

Default Value: **.py**

Max Recent Items

Maximum number of items to display in the Recent menus.

Internal Name: **gui.max-recent-files**

Data Specification: **[from 3 to 200]**

Default Value: **20**

Maximum File Size (MB)

Maximum size of files that Wing will try to open, in MB.

Internal Name: **gui.max-file-size**

Data Specification: **[from 1 to 100000]**

Default Value: **100**

- **File Types**

Extra File Types

This is a map from file extension or wildcard to mime type. It adds additional file type mappings to those built into Wing IDE. File extensions can be specified alone without dot or wildcard, for example "xcf" or using wildcards containing "*" and/or "?", for example "Makefile*". The mime type to use for Python files is "text/x-python".

Internal Name: **main.extra-mime-types**

Data Specification: **[dict; keys: <type str>, values: [text/x-smalltalk, text/x-sql, t ext/x-pov, t x-erlang, text/x-javascript, text/x-scss, text/x-fortran, text/x-mysql, text/x-vhdl, text/x-escrip**

Default Value: **{}**

File Filters

Defines file filters to apply to file names for inclusion and exclusion from a larger set (such as scanned disk files or all project files).

Each filter is named and contains one list of inclusion patterns and one list of exclusion patterns. The patterns can be a wildcard on the file name, wildcard on a directory name, or a mime type name.

> Only a single pattern needs to be matched for inclusion or exclusion. Exclusion patterns take precedence over inclusion patterns, so any match on an exclusion pattern will always exclude a file from the selected set. Filters are used in constraining search, adding project files, and for other operations on collections of files.

Internal Name: **main.file-filters**

Data Specification: **[file filters]**

Default Value: **{u'All Source Files': (set([]), set([('wildcard-filename', '*.pyo'), (' wildcard-filen 'wildcard-filename', '*~'), ('wildcard-filename', '*.manifest'), ('wildcard-filename ', '*.pdb'), ('w ame', '*.pyc'), ('wildcard-filename', '*.tar.gz'), ('wildcard-directory', '.bzr'), ('wild card-directo g')])), u'C/C++ Files': (set([('mime-type', 'text/x-c-source'), ('mime-type', 'text/x- cpp-source') ilename', '*.bsc'), ('wildcard-filename', '.#*'), ('wildcard-filename', '*.lib'), ('wildc ard-filenam wildcard-filename', '*.zip'), ('wildcard-filename', '.coverage'), ('wildcard-filena me', '*.wpu'), (' dcard-directory', '.bzr'), ('wildcard-directory', 'CVS'), ('wildcard-filename', '#*#'), ('wildcard-fi**

- **Reloading**

External Check Freq

Time in seconds indicating the frequency with which the IDE should check the disk for files that have changed externally. Set to 0 to disable entirely.

Internal Name: **cache.external-check-freq**

Data Specification: **<type float>, <type int>**

Default Value: **5**

Reload when Unchanged

Selects action to perform on files found to be externally changed but unaltered within the IDE. Use Auto Reload to automatically reload these files, Immediately Request Reload to ask via a dialog box upon detection, Requst Reload on Edit to ask only if the unchanged file is edited within the IDE subsequently, or Never

Reload to ignore external changes (although you will still be warned if you try to save over an externally changed file)

Internal Name: **cache.unchanged-reload-policy**

Data Specification: **[never-reload, auto-reload, request-reload, edit-reload]**

Default Value: **auto-reload**

Reload when Changed

Selects action to perform on files found to be externally changed and that also have been altered in the IDE. One of Immediately Request Reload to ask via a dialog box upon detection, Request Reload on Edit to ask if the file is edited further, or Never Reload to ignore external changes (although you will always be warned if you try to save over an externally changed file)

Internal Name: **cache.changed-reload-policy**

Data Specification: **[never-reload, request-reload, edit-reload]**

Default Value: **request-reload**

Check Hash before Reloading

Don't reload files if size has not changed and a hash of the contents matches the hash when it was last read. This check is skipped if file is larger than 5 MB.

Internal Name: **cache.check-hash-before-reload**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

- **External Display**

File Display Commands

Posix only: The commands used to display or edit local disk files selected from the Help menu or project files selected for external display. This is a map from mime type to a list of display commands; each display command is tried in order of the list until one works. The mime type "*" can be used to set a generic viewer, such as a web browser. Use %s to place the file name on the command lines. If unspecified then Wing will use the configured URL viewer in the environment (specified by BROWSER environment variable or by searching the path for common browsers). On Windows, the default viewer for the file type is used instead so this preference is ignored. On OS X, files are opened with "open" by default so this preference is rarely needed.

Internal Name: **gui.file-display-cmds**

Data Specification: **[dict; keys: <type str>, values: [list of: <type str>]]**

Default Value: **{}**

Url Display Commands

Posix only: The commands used to display URLs. This is a map from protocol type to a list of display commands; each display command is tried in order of the list until one works. The protocol "*" can be used to set a generic viewer, such as a multi-protocol web browser. Use %s to place the URL on the command lines. If unspecified then Wing will use the configured URL viewer in the environment (specified by BROWSER environment variable or by searching the path for common browsers). On Windows, the default web browser is used instead so this preference is ignored. On OS X, URLs are opened with "open" by default so this preference is rarely needed.

Internal Name: **gui.url-display-cmds**

Data Specification: **[dict; keys: <type str>, values: [list of: <type str>]]**

Default Value: **{}**

Editor

Error Indicators

Controls whether Wing will show error and/or warning indicators on the editor as red and yellow underlines. When shown, hovering the mouse over the indicator shows the error or warning detail in a tooltip.

Internal Name: **edit.error-display**

Data Specification: **[show-errors, show-none, show-all]**

Default Value: **show-all**

Show Line Numbers

Shows or hides line numbers on the editor.

Internal Name: **edit.show-line-numbers**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

Show Whitespace

Set to true to show whitespace with visible characters by default

Internal Name: **edit.show-whitespace**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

Show EOL

Set to true to show end-of-line with visible characters by default

Internal Name: **edit.show-eol**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

Split Reuse Policy

Policy for reusing splits in editors when new files are opened: Either always open in current split, reuse already visible editor falling back on current split, reuse already visible editor falling back on adjacent split, or always open in an adjacent split. This only has an effect when more than one editor split is visible.

Internal Name: **gui.split-reuse-policy**

Data Specification: **[current, reuse-adjacent, reuse-current, adjacent]**

Default Value: **current**

Strip Trailing White Space

Controls whether to automatically strip trailing white space in the editor. May be enabled for any file or only files that are part of the current project.

Internal Name: **main.auto-rstrip-on-save**

Data Specification: **[tuple length 2 of: [disabled, on-save-project, on-save], <type str>]**

Default Value: **disabled**

Block Comment Style

Style of commenting to use when commenting out blocks of Python code.

Internal Name: **gui.block-comment-style**

Data Specification: **[indented-pep8, indented, block]**

Default Value: **indented**

Scroll Past End

Set this to allow scrolling the editor past the last line.

Internal Name: **edit.scroll-past-end**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

Ensure File Ends With EOL When Saving

Whether to add an eol at the end of the file when it is saved

Internal Name: **edit.ensure-ending-eol-on-save**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

Enable Font Size Zooming

Whether to allow font size zooming in the editor, using the mouse wheel, track pad, or zoom-in and zoom-out commands.

Internal Name: **edit.enable-font-zoom**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

- **Selection/Caret**

Selection Color

The color used to indicate the current text selection on editable text.

Internal Name: **gui.qt-text-selection-color**

Data Specification: **[None or [tuple length 3 of: [from 0 to 255], [from 0 to 255], [from 0 to 255]]]**

Default Value: **None**

Caret Color

Selects the color to use for the editor caret.

Internal Name: **edit.caret-color**

Data Specification: **[None or [tuple length 3 of: [from 0 to 255], [from 0 to 255], [from 0 to 255]]]**

Default Value: **None**

Caret Width

Width of the blinking insertion caret on the editor, in pixels. Currently limited to a value between 1 and 3.

Internal Name: **edit.caret-width**

Data Specification: **[from 1 to 3]**

Default Value: **1**

Caret Flash Rate (ms)

Sets the time in milliseconds between showing and hiding the caret when it is flashing; use 0 to disable flashing entirely

Internal Name: **edit.caret-flash-rate**

Data Specification: **[from 0 to 2000]**

Default Value: **500**

Caret Line Highlight

Selects whether to highlight the line the caret is currently on. When enabled, a highlight color and alpha (transparency) can be set.

Internal Name: **edit.caret-line-highlight**

Data Specification: **[None or [tuple length 2 of: [None or [tuple length 3 of: [fro m 0 to 255], [f**

Default Value: **None**

- **Indentation**

Use Indent Analysis

Select when to use indent analysis (examination of current file contents) in order to determine tab size and indent size. Either always in all files, only in Python files, or never.

Internal Name: **edit.use-indent-analysis**

Data Specification: **[always, never, python-only]**

Default Value: **always**

Default Tab Size

Set size of tabs, in spaces, used in new files. Note that in Python files that contain mixed space and tab indentation, tab size is always forced to 8 spaces. Use the Indentation Manager to alter indentation in existing files.

Internal Name: **edit.tab-size**

Data Specification: **[from 1 to 80]**

Default Value: **8**

Default Indent Size

Sets size of an indent, in spaces, used in new files. This is overridden in non-empty files, according to the actual contents of the file. In files with tab-only indentation, this value may be modified so it is a multiple of the configured tab size. Use the Indentation Manager to alter indentation in existing files.

Internal Name: **edit.indent-size**

Data Specification: **[from 1 to 80]**

Default Value: **4**

Default Indent Style

Set the style of indentation used in new files. This is overridden in non-empty files, according to the actual contents of the file. Use the Indentation Manager to alter indentation in existing files.

Internal Name: **edit.indent-style**

Data Specification: **[mixed, spaces-only, tabs-only]**

Default Value: **spaces-only**

Auto Indent

Controls when Wing automatically indents when return or enter is typed.

Internal Name: **edit.auto-indent**

Data Specification: **[0, 1, blank-only]**

Default Value: **1**

Show Indent Guides

Set to true to show indent guides by default

Internal Name: **edit.show-indent-guides**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

Show Python Indent Warning Dialog

Set to show a warning dialog when opening a Python file that contains potentially problematic indentation: Either inconsistent and possibly confusing indentation, a mix of indent styles in a single file, or mixed tab and space indentation (which is not recommended for Python).

Internal Name: **edit.show-python-indent-warnings**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Show Override Warning Dialog

Show indent mismatch warning dialog when user selects an indent style that is incompatible with existing file content. This only applies to non-Python files since Wing disallows overriding the indent style in all Python files.

Internal Name: **edit.show-non-py-indent-warning**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

- **Line Wrapping**

Wrap Long Lines

Enable to wrap long source lines on the editor display.

Internal Name: **edit.wrap-lines**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

Edge Markers

Control whether and how edge markers are shown in the editor.

Internal Name: **edit.qt-show-edge-markers**

Data Specification: **[tuple length 3 of: [0, 1, 2], [from 0 to 10000], [None or [tupl e length 3 of:**

Default Value: **(0, 80, None)**

Reformatting Wrap Column

Column at which text should be wrapped by commands that automatically rearrange text

Internal Name: **edit.text-wrap-column**

Data Specification: **<type int>**

Default Value: **77**

- **Clipboard**

On Empty Selection

Controls whether or not to copy or cut the whole current line when there is no selection on the editor.

Internal Name: **edit.smart-clipboard**

Data Specification: **[disabled, copy-cut, copy]**

Default Value: **copy**

Middle Mouse Paste

Paste text into the editor from the clipboard when the middle mouse button is pressed. Disabling this is mainly useful for wheel mice with a soft wheel that causes pasting of text before wheel scrolling starts.

Internal Name: **edit.middle-mouse-paste**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

Adjust Indent After Paste

Controls whether Wing automatically adjusts indents after multi-line text is pasted. When enabled, a single undo will remove any alterations in indentation.

Internal Name: **edit.adjust-indent-after-paste**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

- **Syntax Coloring**

Background Color

Background color to use on the source editor, Python Shell, Debug Probe, Source Assistant, and other tools that display source code. Foreground colors for text may be altered automatically to make them stand out on the selected background color.

Internal Name: **edit.background-color**

Data Specification: **[None or [tuple length 3 of: [from 0 to 255], [from 0 to 255], [from 0 to 255]]]**

Default Value: **None**

Syntax Formatting

Formatting options for syntax coloring in editors. Colors are relative to a white background and will be transformed if the background color is set to a color other than white.

Internal Name: **.edit.syntax-formatting**

Data Specification: **[dict; keys: <type str>, values: [dict; keys: [italic, back, fore , bold], value**

Default Value: **{}**

Highlight Builtins

Highlight Python builtins

Internal Name: **edit.highlight-builtins**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

- **Occurrences**

Highlight Occurrences

Selects when to automatically highlight other occurrences of the current selection on the editor

Internal Name: **edit.highlight-occurrences**

Data Specification: **[always, never, words]**

Default Value: **words**

Match Case

Disable to allow occurrences highlighting also where case does not match.

Internal Name: **edit.match-case-occurrences**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

Occurrences Indicator Style

The style of indicator to use for highlighting other occurrences of the current selection on the editor.

Internal Name: **edit.occurrence-indicator-style**

Data Specification: **[box, block]**

Default Value: **block**

Occurrences Color

The color used to indicate the current text selection on editable text.

Internal Name: **edit.occurrence-color**

Data Specification: **[None or [tuple length 3 of: [from 0 to 255], [from 0 to 255], [from 0 to 255]]]**

Default Value: **None**

- **Bookmarks**

Bookmark Color

Color to use on the source editor to indicate the location of user-defined bookmarks.

Internal Name: **edit.qt-bookmark-color**

Data Specification: **[None or [tuple length 3 of: [from 0 to 255], [from 0 to 255], [from 0 to 255]]]**

Default Value: **None**

Bookmark Style

Visual display style to use for bookmarks: Either an underline, a background color change, or no visible marker.

Internal Name: **edit.bookmark-style**

Data Specification: **[None, underline, background]**

Default Value: **background**

- **Folding**

Enable Folding

Whether to enable folding source code.

Internal Name: **edit.enable-folding**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Line Mode

Whether and how to show a line at a collapsed fold point. Controls the position of the line and whether it is shown for collapsed or expanded fold points.

Internal Name: **edit.fold-line-mode**

Data Specification: **[above-collapsed, above-expanded, none, below-collapsed, below-expanded]**

Default Value: **below-collapsed**

Indicator Style

Selects the type of indicators to draw at fold points.

Internal Name: **edit.fold-indicator-style**

Data Specification: **[from 0 to 3]**

Default Value: **1**

Fold Trailing White Space

Controls whether or not trailing white space after a block of code is folded up along with the block, for a more compact folded display.

Internal Name: **edit.fold-trailing-whitespace**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Foreground Color

Color to use for the foreground of the fold indicators.

Internal Name: **edit.fold-mark-foreground-color**

Data Specification: **[None or [tuple length 3 of: [from 0 to 255], [from 0 to 255], [from 0 to 255]]]**

Default Value: **None**

Background Color

Color to use for the background of the fold indicators.

Internal Name: **edit.fold-mark-background-color**

Data Specification: **[None or [tuple length 3 of: [from 0 to 255], [from 0 to 255], [from 0 to 255]]]**

Default Value: **None**

- **Auto-completion**

Auto-show Completer

Controls whether or not the completer is always shown automatically during typing, never auto-shown, or shown only after a certain number of characters are in the completion fragment. When auto-show is disabled, the auto-completer can still be shown on demand with the Show Completer item in the Source menu.

Internal Name: **edit.autocomplete-autoshow-option**

Data Specification: **[always, never]**

Default Value: **always**

Auto-completer Height

The maximum number of lines to show in the auto-completer at once.

Internal Name: **edit.autocompleter-height**

Data Specification: **<type int>**

Default Value: **10**

Auto-complete Delay (sec)

Delay in seconds from last key press to wait before the auto-completer is shown. If 0.0, the auto-completer is shown immediately.

Internal Name: **edit.autocomplete-delay**

Data Specification: **<type float>, <type int>**

Default Value: **0.0**

Auto-complete Timeout

Timeout in seconds from last key press after which the auto-completer is automatically hidden. If 0.0, the auto-completer does not time out.

Internal Name: **edit.autocomplete-timeout**

Data Specification: **<type float>, <type int>**

Default Value: **0**

Completion Keys

Controls which keys will enter selected completion value into the editor.

Internal Name: **edit.autocomplete-keys**

Data Specification: **[tuple of: [f1, f3, return, space, period, bracketleft, tab, f12, colon, f10, pa**

Default Value: **['tab']**

Completion Mode

Selects how completion is done in the editor: Either insert the completion at the cursor, replace any symbols that match the leading part of the completion (and insert in other cases), or replace any existing symbol with the new symbol.

Internal Name: **edit.autocomplete-mode**

Data Specification: **[replace-matching, insert, replace]**

Default Value: **insert**

Case Insensitive Matching

Controls whether matching in the completer is case sensitive or not. The correct case is always used when a completion is chosen.

Internal Name: **edit.autocomplete-case-insensitive**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

Include Snippets in Completer

Whether or not to include code snippets in the auto-completer.

Internal Name: **edit.snippets-in-autocompleter**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

Python Turbo Mode (Experimental)

When enabled, the Python auto-completer enters the completion automatically whenever a key other than a valid symbol name key is pressed. When disabled, only the configured completion keys enter the completion into the editor.

Internal Name: **edit.autocomplete-turbo-mode**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

Non-Python Completion

Controls whether or not use the completer in non-Python files, where it uses a simple word list generated from the existing contents of the file. If enabled, the number of characters required before the completer is shown may be specified here.This value overrides any character threshold set above.

Internal Name: **edit.autocomplete-non-python-option**

Data Specification: **[always, never]**

Default Value: **3**

Non-Python Word Size

Sets the minimum size of words to add to the completion list for non-Python files. This affects only words found in the file, and not words included because they are keywords for that file type.

Internal Name: **edit.autocomplete-non-python-word-size**

Data Specification: **<type int>**

Default Value: **4**

- **Auto-editing**

Auto-Editing Enabled

Enable or disable Wing's auto-editing capability. When enabled, a default set of individual auto-editing operations (such as auto-closing quotes and parenthesis and auto-entering invocation arguments) will be activated. The individual operations can then be enabled or disabled independently in preferences.

Internal Name: **edit.auto-edit-enabled**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Auto-Close Characters

Enable to auto-close quotes, parenthesis, braces, comments, and so forth.

Internal Name: **edit.auto-edit-close**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Auto-Enter Invocation Args

Enable auto-entry of invocation arguments for a function or method call.

Internal Name: **edit.auto-edit-invoke**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Apply Quotes to Selection

Enable placing quotes around a non-empty selection.

Internal Name: **edit.auto-edit-quotes**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Apply Comment Key to Selection

Enable commenting out a non-empty selection when a comment character is pressed.

Internal Name: **edit.auto-edit-comment**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Apply (), [], and {} to Selection

Enable surrounding non-empty selection when a parenthesis is pressed.

Internal Name: **edit.auto-edit-parens**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Apply Colon to Selection

Enable creating a new block with a selected range of lines when colon is pressed.

Internal Name: **edit.auto-edit-colon-creates-block**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Auto-Enter Spaces

Enable auto-entering spaces around operators and punctuation.

Internal Name: **edit.auto-edit-spaces**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

** Auto-Space After Keywords**

Enable auto-entering spaces after keywords.

Internal Name: **edit.auto-edit-spaces-kw**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

** Enforce PEP8 Style Spacing**

When auto-entering spaces is enabled, enforce PEP8 style spacing by preventing redundant spaces.

Internal Name: **edit.auto-edit-spaces-enforce**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

** Spaces in Argument Lists**

When auto-entering spaces is enabled, also auto-enter spaces in argument lists.

Internal Name: **edit.auto-edit-spaces-args**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

Manage Blocks on Repeated Colon Key Presses

Auto-enter newline and auto-indent after typing a colon that starts a new Python block and indent following line or block of lines when colon is pressed repeatedly. This also starts a new Python block using a selected range of lines as the body, if colon is pressed on a non-empty selection.

Internal Name: **edit.auto-edit-colon**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

Continue Comment or String on New Line

Automatically continue comments or strings in the form ("") or () after a newline is typed within the comment or string text

Internal Name: **edit.auto-edit-continue**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Correct Out-of-Order Typing

Automatically correct code when typing keys out of order. This handles cases such as x(.) -> x(). and x(:) -> x(): as well as auto-inserting . when missing

Internal Name: **edit.auto-edit-fixups**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

- **Diff/Merge**

Orientation

Orientation of difference/merge views: Side-by-side or top/bottom

Internal Name: **diff.orientation**

Data Specification: **[horizontal, vertical]**

Default Value: **horizontal**

Lock Scrolling

Controls whether scrolling of the diff/merge editors is locked to synchronize the editor scroll positions.

Internal Name: **diff.scroll-lock**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

Ignore White Space

Controls whether differences will ignore changes that alter white space only.

Internal Name: **diff.ignore-whitespace**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

Empty Session Warning

Controls whether to warn when changing white space filtering causes sessions to become empty of changes.

Internal Name: **diff.empty-session-warning**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

Diff Color

Color to use on the source editor for differences during a diff/merge session. The current mark is drawn in a lighter version of the same color. The within-difference change indicators are drawn transparently with the color set in the Text Selection Color preference.

Internal Name: **edit.qt-diff-color**

Data Specification: **[None or [tuple length 3 of: [from 0 to 255], [from 0 to 255], [from 0 to 255]]]**

Default Value: **None**

Merged Diff Color

Color to use on the source editor for already merged differences during a diff/merge session. The current mark is drawn in a lighter version of the same color. The within-difference change indicators are drawn transparently with the color set in the Text Selection Color preference.

Internal Name: **edit.qt-merged-diff-color**

Data Specification: **[None or [tuple length 3 of: [from 0 to 255], [from 0 to 255], [from 0 to 255]]]**

Default Value: **None**

- **Printing**

Print Header Format

Set the header format to use for printing. This can be any text with any of the following special fields mixed in: %basename% - base file name;

%prepend-fullpath% - full path file name; %prepend-relative% - relative path with from project file; %append-relative% - file name with relative path appended; %append-fullpath% - file name with full path appended; %file-time% - file modification time; %file-date% - file modification date; %current-time% - current time; %current-date% - current date; %page% - current page being printed

Internal Name: **edit.print-header-format**

Data Specification: **<type str>**

Default Value: **%prepend-fullpath%**

Print Footer Format

Set the footer format to use for printing. The values allowed are the same as those for print-header-format.

Internal Name: **edit.print-footer-format**

Data Specification: **<type str>**

Default Value: **Page %page%, last modified %file-date% %file-time%**

Print Header Font

Font to use in print header.

Internal Name: **edit.print-header-font**

Data Specification: **[None or <type str>]**

Default Value: **None**

Print Footer Font

Font to use in print footer.

Internal Name: **edit.print-footer-font**

Data Specification: **[None or <type str>]**

Default Value: **None**

Use Default Foreground Colors

Use default foreground colors for all text when printing. This is necessary when using a dark background in the GUI and printing on white paper.

Internal Name: **edit.use-default-foreground-when-printing**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

- **Context Menu**

Groups Shown

Controls which groups of menu items will be shown in the editor's context menu.

Internal Name: **edit.context-menu-groups**

Data Specification: **[tuple of: [comment, indent, clip, script, vcs, nav, debug]]**

Default Value: **['clip', 'nav', 'debug', 'comment', 'indent', 'vcs', 'script']**

Custom Items

Extra menu items to add to the editor context menu.

Internal Name: **edit.context-menu-custom-items**

Data Specification: **[tuple of: [tuple length 2 of: <type str>, <type str>]]**

Default Value: **()**

- **Advanced**

Brace Highlighting

Enabled to automatically highlight the matching braces next to the cursor or as they are typed.

Internal Name: **edit.auto-brace-match**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Maximum Non-Sticky Editors

Maximum number of non-sticky (auto-closing) editors to keep open at one time, in addition to any that are visible on screen

Internal Name: **gui.max-non-sticky-editors**

Data Specification: **<type int>**

Default Value: **1**

Selection Policy

This controls whether to retain selection in the editor after certain operations. The editor may always select the text that was operated on, only retain existing selections, or never select after the operation completes.

Internal Name: **edit.select-policy**

Data Specification: **[dict; keys: [(u'Indent Region', 'indent-region'), (u'Indent To Match', n-select'), (u'Always Select', 'always-select')]]**

Default Value: **{'uncomment-out-region': 'retain-select', 'outdent-region': 'retai n-select', 'com**

Mini-search Case Sensitivity

Whether or not mini-search is case sensitive. May match the current keyboard personality's default, use case sensitive search only if an upper case character is typed, always search case sensitive, or always search case insensitively.

Internal Name: **edit.minisearch-case-sensitive**

Data Specification: **[always, never, if-upper, match-mode]**

Default Value: **match-mode**

Symbol Menu Max Length

The maximum number of names allowed on a single symbol menu

Internal Name: **.edit.max-symbol-menu-name-count**

Data Specification: **<type int>**

Default Value: **200**

Debugger

Integer Display Mode

Select the display style for integer values.

Internal Name: **debug.default-integer-mode**

Data Specification: **[dec, hex, oct]**

Default Value: **dec**

Hover Over Symbols

Enable to display debug data values for any symbol on the editor when the mouse cursor hovers over it.

Internal Name: **debug.hover-over-symbols**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Hover Over Selection

Controls whether debug values are shown when the mouse hovers over a selection in the editor. This may be disabled, enabled for symbols (like x.y.z) only, or enabled for all selections including function or methods calls. WARNING: Enabling evaluation of any selection may result in function or method calls that have side effects such as altering the program state or even making unintended database or disk accesses!

Internal Name: **debug.hover-over-selections**

Data Specification: **[0, 1, all]**

Default Value: **1**

Run Marker Color

The color of the text highlight used for the run position during debugging

Internal Name: **debug.debug-marker-color**

Data Specification: **[None or [tuple length 3 of: [from 0 to 255], [from 0 to 255], [from 0 to 255]]]**

Default Value: **None**

Run Marker Alpha

Select transparency (0-160) of the text highlight used for the run position during debugging

Internal Name: **debug.run-marker-alpha**

Data Specification: **[None or <type int>]**

Default Value: **None**

Active Range Color

The color of the active range of code used for quick evaluation in the Python Shell or Debug Probe.

Internal Name: **debug.active-range-color**

Data Specification: **[None or [tuple length 3 of: [from 0 to 255], [from 0 to 255], [from 0 to 255]]]**

Default Value: **None**

Line Threshold

Defines the character length threshold under which a value will always be shown on a single line, even if the value is a complex type like a list or dict.

Internal Name: **debug.line-threshold**

Data Specification: **<type int>**

Default Value: **65**

Ignore Unsynchronized Files

Controls whether or not Wing ignores files that were not saved before starting debug or that have changed since they were loaded by the debug process. Wing normally will warn of unsynchronized files since breakpoints may not be reached and stepping through the files may not work properly if lines have moved. Checking this option turns off these warnings.

Internal Name: **gui.ignore-unsaved-before-action**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

- **Exceptions**

Report Exceptions

Controls how Wing reports exceptions that are raised by your debug process. By default, Wing shows exceptions at the time that the exception traceback would normally be printed. Alternatively, Wing can try to predict which exceptions are unhandled, and stop immediately when unhandled exceptions are raised so that any finally clauses can be stepped through in the debugger. Wing can also stop on all exceptions (even if handled) immediately when they are raised, or it can wait to report fatal exceptions as the debug process terminates. In the latter case Wing makes a best effort to stop before the debug process exits or at least to report the exception post-mortem, but one or both may fail if working with externally launched debug processes. In that case, we recommend using When Printed exception reporting mode.

Internal Name: **debug.exception-mode**

Data Specification: **[unhandled, always, never, printed]**

Default Value: **printed**

Report Logged Exceptions In When Printed Mode

Controls whether to stop on exceptions logged with logging.exception if the exception mode is set to 'When Printed'

Internal Name: **debug.stop-on-logged-exception**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

Never Report

Names of builtin exceptions to never report, even if the exception is not handled. This list takes precedence over the Always Report preference and the Report Exceptions preference when it is set to a value other than Always Immediately.

Internal Name: **debug.never-stop-exceptions**

Data Specification: **[tuple of: <type str>]**

Default Value: **['SystemExit', 'GeneratorExit']**

Always Report

Names of builtin exceptions to (nearly) always report. These exceptions are not reported only if they are explicitly caught by the specific subclass in the same frame in which they are raised.

Internal Name: **debug.always-stop-exceptions**

Data Specification: **[tuple of: <type str>]**

Default Value: **['AssertionError']**

- **I/O**

Debug I/O Encoding

Encoding of input/output in the Debug I/O panel

Internal Name: **debug.debug-io-encoding**

Data Specification: **[None or [Central and Eastern European iso8859-2, Japane se iso-2022-jp Danish, Norwegian cp865, Cyrillic Languages iso8859-5, Turkish cp1026, We stern Europea**

Default Value: **None**

Shell Encoding

Encoding of input/output in the integrated Python Shell and Debug Probe

Internal Name: **debug.debug-probe-encoding**

Data Specification: **[None or [Central and Eastern European iso8859-2, Japane se iso-2022-jp Chinese (PRC) hz, Portuguese cp860, Chinese (ROC) cp950, Unicode (UTF-16 , big endian)**

Default Value: **None**

Pretty Print in Shells

Enable to use pprint.pprint to display values in the Python Shell and Debug Probe.

Internal Name: **debug.pretty-print-in-shells**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

OS Commands Encoding

Default encoding of sub-process input/output when run in the OS Commands panel. This can be overridden on a per-command basis, in each command's properties.

Internal Name: **consoles.encoding**

Data Specification: **[None or [Central and Eastern European iso8859-2, Japane se iso-2022-jp Danish, Norwegian cp865, Cyrillic Languages iso8859-5, Turkish cp1026, We stern Europea**

Default Value: **None**

Use External Console

Selects whether to use the integrated Debug I/O tool for debug process input/output or an external terminal window. Use an external window if your debug process depends on details of the command prompt environment for cursor movement, color text, etc.

Internal Name: **debug.external-console**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

External Console Waits on Exit

Determines whether to leave up the console after normal program exit, or to close the console right away in all cases. This is only relevant when running with an external native console instead of using the integrated Debug I/O tool.

Internal Name: **debug.persist-console**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

External Consoles

A list of the terminal programs that are used with debug processes when running with an external console. Each is tried in turn until one is found to exist. If just the name is given, Wing will look for each first on the PATH and then in likely places. Specify the full path (starting with "/") to use a specific executable. If program arguments are specified, they must end with the argument that indicates that the rest of arguments are the program to run in the terminal. If the program name starts with ${WINGHOME} , ${WINGHOME} is replaced by the Wing install directory. On OS X if the program name ends is .applescript, the environment is loaded from a file before starting the debugger.

Internal Name: **debug.x-terminal**

Data Specification: **[tuple of: <type str>]**

Default Value: **['${WINGHOME}/resources/osx/run-in-terminal.applescript', 'gn ome-terminal**

- **Data Filters**

Omit Types

Defines types for which values are never shown by the debugger.

Internal Name: **debug.omit-types**

Data Specification: **[tuple of: <type str>]**

Default Value: **('function', 'builtin_function_or_method', 'class', 'classobj', 'inst ance method**

Omit Names

Defines variable/key names for which values are never shown by the debugger.

Internal Name: **debug.omit-names**

Data Specification: **[tuple of: <type str>]**

Default Value: **()**

Do Not Expand

Defines types for which values should never be probed for contents. These are types that are known to crash when the debugger probes them because they contain buggy data value extraction code. These values are instead shown as an opaque value with hex object instance id and are never accessed for runtime introspection.

Internal Name: **debug.no-probe-types**

Data Specification: **[tuple of: <type str>]**

Default Value: **('GdkColormap', 'IOBTree', 'JPackage')**

Huge List Threshold

Defines the length threshold over which a list, dict, or other complex type will be considered too large to show in the normal debugger. If this is set too large, the debugger will time out (see the Network Timeout preference)

Internal Name: **debug.huge-list-threshold**

Data Specification: **<type int>**

Default Value: **2000**

Huge String Threshold

Defines the length over which a string is considered too large to fetch for display in the debugger. If this is set too large, the debugger will time out (see the Network Timeout preference).

Internal Name: **debug.huge-string-threshold**

Data Specification: **<type int>**

Default Value: **64000**

- **External/Remote**

Accept Debug Connections

Controls whether or not the debugger listens for connections from an externally launched program. This should be enabled when the debug program is not launched by the IDE.

Internal Name: **debug.passive-listen**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

Allowed Hosts

Sets which hosts are allowed to connect to the debugger when it is listening for externally launched programs.

Internal Name: **debug.passive-hosts**

Data Specification: **[tuple of: <type str>]**

Default Value: **('127.0.0.1',)**

Server Host

Determines the network interface on which the debugger listens for connections. This can be a symbolic name, an IP address, or left unspecified to indicate that the debugger should listen on all valid network interfaces on the machine. Note that when a debug session is launched from within the IDE (with the Run button), it always connects from the loopback interface (127.0.0.1)

Internal Name: **debug.network-server**

Data Specification: **[None or <type str>]**

Default Value: **None**

Server Port

Determines the TCP/IP port on which the IDE will listen for the connection from the debug process. This needs to be unique for each developer working on a given host. The debug process, if launched from outside of the IDE, needs to be told the value specified here using kWingHostPort inside wingdbstub.py or by WINGDB_HOSTPORT environment variable before importing wingdbstub in the debug process.

Internal Name: **debug.network-port**

Data Specification: **[from 0 to 65535]**

Default Value: **50005**

Location Map

Defines a mapping between the remote and local locations of files for host-to-host debugging. For each IP address, a remote and local prefix is given. This should be used when full paths of files on the remote host do not match those for the same files on the local host. Wing assumes an external file server or synchronization protocol is in use and does not itself transfer the files.

Internal Name: **debug.location-map**

Data Specification: **[dict; keys: <ip4 address #.#.#.#>, values: [None or [list of: [tuple length 2**

Default Value: **{'127.0.0.1': None}**

Kill Externally Launched

Enable or disable the terminating debug processes that were launched from outside of the IDE. When disabled, Wing just detaches from the process, leaving it running.

Internal Name: **debug.enable-kill-external**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

Common Attach Hosts

List of host/port combinations that should be included by default in the attach request list shown with Attach to Process in the Debug menu, in addition to those that are registered at runtime. These are used primarily with externally launched debug processes, since Wing automatically shows IDE-launched processes for attach when appropriate. This value corresponds with kAttachPort configured in wingdbstub.py or by WINGDB_ATTACHPORT environment variable before importing wingdbstub in the debug process.

Internal Name: **debug.attach-defaults**

Data Specification: **[tuple of: [tuple length 2 of: <type str>, [from 0 to 65535]]]**

Default Value: **(('127.0.0.1', 50015),)**

- **Advanced**

Network Timeout

Controls the amount of time that the IDE will wait for the debug process to respond before it gives up. This protects the IDE from freezing up if your program running within the debug process crashes or becomes unavailable. It must also be taken into account when network connections are slow or if sending large data values (see the Huge List Threshold and Hug String Threshold preferences).

Internal Name: **debug.network-timeout**

Data Specification: **<type float>, <type int>**

Default Value: **10**

Resolve Properties

Set to show property values in the debug data views. This should be used with caution. It enables invocation of the fget() method on the property, which in some code bases can execute unwanted code, make unexpected changes to runtime state, hang on lengthy computations, trigger thread deadlocks, or crash on buggy user code while debug data is being displayed in the IDE.

Internal Name: **debug.resolve-properties**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

Allow Dynamic Introspection

Set to allow Python code and other dynamic calls to be invoked while introspecting values in the debugger, for display in the auto-completer, shells, and source

assistant. This should be used with caution. In some code bases, enabling this can execute unwanted code, make unexpected changes to runtime state, hang on lengthy computations, trigger thread deadlocks, or crash on buggy user code, while working in the IDE.

Internal Name: **debug.allow-dynamic-introspection**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

Call Python ____repr____ Methods

Allow __repr__ methodes implemented in Python to be invoked. Set to false if the __repr__ methods take too long to compute or fail due to other bugs

Internal Name: **debug.allow-bytecode-repr**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

Show Data Warnings

Controls whether or not time out, huge value, and error handling value errors are displayed by the debugger the first time they are encountered in each run of Wing.

Internal Name: **debug.show-debug-data-warnings**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Use sys.stdin Wrapper

Whether sys.stdin should be set a wrapper object for user input in the program being debugged. The wrapper allows debug commands, such as pause, to be executed while the program is waiting for user input. The wrapper may cause problems with multi-threaded programs that use C stdio functions to read directly from stdin and will be slower than the normal file object.However, turning this preference off means that your debug process will not pause or accept breakpoint changes while waiting for keyboard input, and any keyboard input that occurs as a side effect of commands typed in the Debug Probe will happen in unmodified stdin instead (even though output will still appear in the Debug Probe as always).

Internal Name: **debug.use-stdin-wrapper**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Show Editor on Exceptions in Shells

Controls whether the debugger raises source files to indicate exception locations encountered when working in the Debug Probe, and other debugger tools.

Internal Name: **debug.raise-from-tools**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Shells Ignore Editor Modes

Set to False so that shells will act modal in the same way as editors when working with a modal key bindings such as that for VI. When True, the shells always act as if in Insert mode.

Internal Name: **debug.shells-ignore-editor-modes**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Execute Pasted Lines in Shells Immediately

Whether to always execute immediately after text is pasted into a shell. Note that if the number of lines exceed the pasted line threshold, the lines are immediately executed.

Internal Name: **debug.shell-always-execute-on-paste**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

Auto-show Run Args Dialog

Controls whether the Debug Args dialog is shown before each debug run: Either never show the dialog or show it only if 'Show this dialog before each run' is checked in the file's properties (this is the default).

Internal Name: **debug.show-args-dialog**

Data Specification: **[per-file, never]**

Default Value: **per-file**

Default Watch Style

Sets the tracking style used when a value is double clicked in order to watch it. Values may be tracked by symbolic name, by object reference and attribute by name, and by direct object reference.

Internal Name: **debug.default-watch-style**

Data Specification: **[ref, parent-ref, symbolic]**

Default Value: **symbolic**

- **Diagnostics**

Debug Internals Log File

This is used to obtain verbose information about debugger internals in cases where you are having problems getting debugging working. Logging can be disabled, or sent to stderr, stdout, or a file.

Internal Name: **debug.logfile**

Data Specification: **[one of: None, [<stdout>, <stderr>], <type str>]**

Default Value: **None**

Extremely Verbose Internal Log

This is used to turn on very verbose and detailed logging from the debugger. Only recommended when debugging the debugger.

Internal Name: **debug.very-verbose-log**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

Python Shell Debug Log

This is used to obtain verbose information about the Python Shell internals in cases where you are having problems getting it working. Logging can be disabled, or sent to stderr, stdout, or a file.

Internal Name: **debug.shell-logfile**

Data Specification: **[one of: None, [<stdout>, <stderr>], <type str>]**

Default Value: **None**

Extremely Verbose Python Shell Debug Log

This is used to turn on very verbose and detailed logging from the Python Shell internals. Only recommended when debugging the Python Shell.

Internal Name: **debug.very-verbose-shell-log**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

Source Analysis

Analyze in Background

Whether Wing should try to analyze python source in the background.

Internal Name: **pysource.analyze-in-background**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Introspect Live Runtime

Set to introspect live Python runtimes for information displayed in autocompletion, the Source Assistant, and debug data value tooltips. Runtimes introspected include the Python Shell and live debug processes stopped at an exception or breakpoint.

Internal Name: **debug.introspect-in-shells**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Typing Suspend Timeout

Number of seconds between last key press and when analysis is re-enabled if analysis is to be suspended while typing occurs. If <= 0, analysis is not suspended.

Internal Name: **edit.suspend-analysis-timeout**

Data Specification: **<type float>, <type int>**

Default Value: **3**

Max Cache Size (MB)

The maximum size of the disk cache in megabytes

Internal Name: **pysource.max-disk-cache-size**

Data Specification: **[from 100 to 10000]**

Default Value: **500**

Max Memory Buffers

The maximum # of analysis info buffers that can be in-memory at once for files that are not open.

Internal Name: **pysource.max-background-buffers**

Data Specification: **[from 50 to 300]**

Default Value: **80**

- **Advanced**

Interface File Path

Path to search for interface files for extension modules. If directory name is relative, it will be interpreted as relative to the user settings directory (USER_SETTINGS_DIR)

Internal Name: **pysource.interfaces-path**

Data Specification: **[tuple of: <type str>]**

Default Value: **('pi-files',)**

Scrape Extension Modules

Enable to automatically load and introspect extension modules and other modules that cannot be statically analysed. These modules are loaded in another process space and 'scraped' to obtain at least some analysis of the module's contents.

Internal Name: **pysource.scrape-modules**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

Scraping Helper Snippets

This is a dictionary from module name to Python code that should be executed before attempting to load extension modules for scraping. This is needed in cases where the extension modules are designed to be loaded only after some configuration magic is performed. For most extension modules, no extra configuration should be needed.

Internal Name: **pysource.scrape-config**

Data Specification: **[dict; keys: <type str>, values: <type str>]**

Default Value: **{'QtSvg': 'try:\n from PyQt4 import QtSvg\nexcept:\n try:\n fr om PyQ tk\nvers = pygtk._get_available_versions().keys()\nvers.sort()\nvers.reverse()\ nfor v in vers ', 'gtk': 'import pygtk\nvers = pygtk._get_available_versions().keys()\nvers.so rt()\nvers.reve :\n from PyQt4 import QtScriptTools\nexcept:\n try:\n from PyQt5 import Q tScriptToo esigner\n', 'pango': 'import pygtk\nvers = pygtk._get_available_versions().key s()\nvers.sor yQt5 import QSci\n except:\n from PySide import QSci\n', 'atk': 'import pyg tk\nvers = pyg QtTest\n except:\n from PySide import QtTest\n', 'QtScript': 'try:\n from Py Qt4 import Qt**

Python Docs URL Prefix

Prefix for Python Standard Library Documentation. This should be in the form http://docs.python.org/library/ and Wing will append module and symbol specific to the given URL. To use locally stored documentation, you must run a local web server since # bookmarks do not work in file: URLs.

Internal Name: **pysource.python-doc-url-prefix**

Data Specification: **[None or <type int>]**

Default Value: **None**

Use SQLite Dot-file Locking

Use slower, dotfile locking for sqllite databases to work around buggy remote file servers. Only needed if the user cache directory is on a remote file system or can be accessed via a remote file system. It is recommended that the user cache directory be on the local file system for performance reasons.

Internal Name: **pysource.use-sqllite-dotfile-locking**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

Version Control

Enable built-in version control

Enable the integrated version control system.

Internal Name: **versioncontrol.enable-non-script**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

Save files without prompting

Save without prompting before running version control commands.

Internal Name: **versioncontrol.save-without-prompting**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

Track changes made in project tool

Track file add, remove, and rename operations made with Wing's Project view into the version control repository.

Internal Name: **versioncontrol.track-disk-operations**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

Automatically refresh status

Watch disk for version control changes and refresh the Project view and Project Status accordingly.

Internal Name: **versioncontrol.watch-disk**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

Enable diagnostic logging

Log all commands to the error-log.

Internal Name: **versioncontrol.log-all-commands**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

- **SVN**

Active

When Subversion version control support is active

Internal Name: **.versioncontrol.svn.active**

Data Specification: **[(u'Always Active', 'always-active'), (u'Not active', 'not-activ e'), (u'Active i**

Default Value: **active-if-project-dir**

SVN Executable

Executable command to run Subversion

Internal Name: **.versioncontrol.svn.executable**

Data Specification: **<type str>**

Default Value: **svn**

SVN Admin Executable

Executable command to run svn

Internal Name: **versioncontrol.svn.svnadmin-executable**

Data Specification: **<type str>**

Default Value: **svnadmin**

Extra global arguments

Extra arguments to pass to every command.

Internal Name: **versioncontrol.svn.extra-global-args**

Data Specification: **<type str>**

Default Value: **""**

- **Git**

Active

When Git version control support is active

Internal Name: **.versioncontrol.git.active**

Data Specification: **[(u'Always Active', 'always-active'), (u'Not active', 'not-activ e'), (u'Active i**

Default Value: **active-if-project-dir**

Git Executable

Executable command to run Git

Internal Name: **.versioncontrol.git.executable**

Data Specification: **<type str>**

Default Value: **git**

Use --porcelain

Use --porcelain output for git status

Internal Name: **versioncontrol.git.use-porcelain**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

- **BZR**

Active

When Bazaar version control support is active

Internal Name: **.versioncontrol.bzr.active**

Data Specification: **[(u'Always Active', 'always-active'), (u'Not active', 'not-activ e'), (u'Active i**

Default Value: **active-if-project-dir**

Bazaar Executable

Executable command to run Bazaar

Internal Name: **.versioncontrol.bzr.executable**

Data Specification: **<type str>**

Default Value: **bzr**

- **Mercurial**

Active

When Mercurial version control support is active

Internal Name: **.versioncontrol.hg.active**

Data Specification: **[(u'Always Active', 'always-active'), (u'Not active', 'not-activ e'), (u'Active i**

Default Value: **active-if-project-dir**

Mercurial Executable

Executable command to run Mercurial

Internal Name: **.versioncontrol.hg.executable**

Data Specification: **<type str>**

Default Value: **hg**

Extra global arguments

Extra arguments to pass to every command.

Internal Name: **versioncontrol.hg.extra-global-args**

Data Specification: **<type str>**

Default Value: **--encoding=utf8**

- **CVS**

Active

When CVS version control support is active

Internal Name: **.versioncontrol.cvs.active**

Data Specification: **[(u'Always Active', 'always-active'), (u'Not active', 'not-activ e'), (u'Active i**

Default Value: **active-if-project-dir**

CVS Executable

Executable command to run CVS

Internal Name: **.versioncontrol.cvs.executable**

Data Specification: **<type str>**

Default Value: **cvs**

Extra global arguments

Extra arguments to pass to every command.

Internal Name: **versioncontrol.cvs.extra-global-args**

Data Specification: **<type str>**

Default Value: **-z3**

- **Perforce**

Active

When Perforce version control support is active

Internal Name: **.versioncontrol.perforce.active**

Data Specification: **[(u'Always Active', 'always-active'), (u'Not active', 'not-activ e'), (u'Active i**

Default Value: **not-active**

Perforce Executable

Executable command to run Perforce

Internal Name: **.versioncontrol.perforce.executable**

Data Specification: **<type str>**

Default Value: **p4**

Extra Global Arguments

Extra arguments to pass to every command.

Internal Name: **versioncontrol.perforce.extra-global-args**

Data Specification: **<type str>**

Default Value: **""**

Don't Find Unregistered Files

Don't find unregistered files when scanning for file status. This can substantially reduce the time to scan large repositories.

Internal Name: **versioncontrol.perforce.dont-find-unregistered**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

IDE Extension Scripting

Search Path

Specifies the directories in which Wing will look for user-defined scripts that extend the functionality of the IDE itself. The directory names may contain environment variables in the $(envname) form. Use $(WING:PROJECT_DIR) for the project directory.For each directory, Wing will load all found Python modules and packages, treating any function whose name starts with a letter (not _ or __) as a script-provided command. Extension scripts found in files within directories later in the list will override scripts of the same name found earlier, except that scripts can never override commands that are defined internally in Wing itself (these are documented in the Command Reference in the users manual). See the Scripting and Extending chapter of the manual for more information on writing and using extension scripts. Note that WINGHOME/scripts is always appended to the given path since it contains scripts that ship with Wing.

Internal Name: **main.script-path**

Data Specification: **[list of: <type str>]**

Default Value: **[u'USER_SETTINGS_DIR/scripts']**

Auto-Reload Scripts on Save

When enabled, Wing will automatically reload scripts that extend the IDE when they are edited and saved from the IDE. This makes developing extension scripts for the IDE very fast, and should work in most cases. Disable this when working on extension scripts that do not reload properly, such as those that reach through the scripting API extensively.

Internal Name: **main.auto-reload-scripts**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

Network

HTTP Proxy Server

Allows manual configuration of an http proxy to be used for feedback, bug reports, and license activation, all of which result in Wing connecting to wingware.com via http. Leave user name and password blank if not required.

Internal Name: **main.http-proxy**

Data Specification: **[None or [tuple length 4 of: <type str>, <type int>, <type str>, <type str>]]**

Default Value: **None**

Internal Preferences

Core Preferences

main.debug-break-on-critical

If True and a gtk, gdk, or glib critical message is logged, Wing tries to start a C debugger and break at the current execution point

Internal Name: **main.debug-break-on-critical**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

main.documentation-language

The language to use for the documentation, when available (not all documentation is translated into all supported languages).

Internal Name: **main.documentation-language**

Data Specification: **[None, de, en, fr]**

Default Value: **en**

main.extra-mime-type-comments

This is a map from mime type to tuple of start/end comment characters for each mime type. One entry should be added for each new mime type added with the main.extra-mime-types preference.

Internal Name: **main.extra-mime-type-comments**

Data Specification: **[dict; keys: <type str>, values: [tuple length 2 of: <type str>, <type str>]]**

Default Value: **{}**

main.extra-mime-type-names

This is a map from mime type to displayable name for that mime type; one entry should be added for each new mime type added with the main.extra-mime-types preference.

Internal Name: **main.extra-mime-type-names**

Data Specification: **[dict; keys: <type str>, values: <type str>]**

Default Value: **{}**

main.help-font-zoom

The amount by which to zoom font sizes in or out in the documentation viewer.

Internal Name: **main.help-font-zoom**

Data Specification: **<type float>**

Default Value: **1.0**

main.ignored-updates

Used internally to keep track of updates the user is not interested in

Internal Name: **main.ignored-updates**

Data Specification: **[list of: <type str>]**

Default Value: **[]**

main.plugin-overrides

Defines which plugins are enabled or disabled.

Internal Name: **main.plugin-overrides**

Data Specification: **[dict; keys: <type str>, values: <boolean: 0 or 1>]**

Default Value: **{}**

main.sassist-allow-pep287-errors

Whether show docstrings as ReST even if they contain parse errors. When disabled, they are shown as plain text instead. When enabled, this may destroy formatting of some docstrings.

Internal Name: **main.sassist-allow-pep287-errors**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

main.sassist-always-show-docstrings

Whether to always show docstrings in the Source Assistant. When disabled, only the docstring for the last displayed symbol is shown.

Internal Name: **main.sassist-always-show-docstrings**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

main.sassist-tries-rewrap

Whether to rewrap plain text docstrings for display in the Source Assistant. This may destroy formatting of some docstrings.

Internal Name: **main.sassist-tries-rewrap**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

main.sassist-show-validity

Whether show docstring type and validity in the Source Assistant.

Internal Name: **main.sassist-show-validity**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

main.sassist-tries-pep287

Whether to try parsing docstrings as ReST format for display in the Source Assistant. This may destroy formatting of some docstrings.

Internal Name: **main.sassist-tries-pep287**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

User Interface Preferences

gui.all-editors-in-all-splits

Whether to show all open editors in a window in every split.

Internal Name: **gui.all-editors-in-all-splits**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

gui.alphabetize-tabs

Whether to keep tabs in alphabetical order.

Internal Name: **gui.alphabetize-tabs**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

gui.feedback-email

Email address to use by default in the Feedback and Bug Report dialogs

Internal Name: **gui.feedback-email**

Data Specification: **<type str>**

Default Value: **""**

gui.last-feedback-shown

Used internally to avoid showing the feedback dialog on exit over and over again.

Internal Name: **gui.last-feedback-shown**

Data Specification: **<type float>**

Default Value: **0.0**

gui.message-config

Controls the format and verbosity of messages shown to the user for each message domain in the message area. Each domain specifies the format (in Python 2.3 logging.Formatter format), and the minimum logging level that should be shown in the display. If a message domain is left unspecified, then the parent domain settings are used instead ("" is the parent of all domains).

Internal Name: **gui.message-config**

Data Specification: **[dict; keys: [search, debugger, analysis, general, project, e ditor, scripts,**

Default Value: **{'': ('%(message)s', 0, 100000)}**

gui.more-controls-for-search-in-files

Controls whether "Search in Files" dialog has an extra row of visible options as buttons.

Internal Name: **gui.more-controls-for-search-in-files**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

gui.prefered-symbol-order

Control preferred order in source index displays such as the editor browse menus. Either sort in the order found in the file or alphabetical order.

Internal Name: **gui.prefered-symbol-order**

Data Specification: **[file-order, alpha-order]**

Default Value: **alpha-order**

gui.reported-exceptions

Used internally to remember which unexpected exceptions have already been reported so we only show error reporting dialog once for each.

Internal Name: **gui.reported-exceptions**

Data Specification: **[dict; keys: <type str>, values: [dict; keys: <type str>, value s: <boolean:**

Default Value: **{}**

gui.scan-for-pythoncom-shell-extensions

Scan for pythoncom shell extensions on Windows

Internal Name: **gui.scan-for-pythoncom-shell-extensions**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

gui.set-win32-foreground-lock-timeout

Controls whether or not to set the foreground lock timeout on Windows XP, where normally Wing will be unable to bring source windows to front whenever the debug process has windows in the foreground. When this preference is true, the system-wide value that prevents background applications from raising windows is cleared whenever Wing is running. This means that other apps will also be able to raise windows without these restrictions while Wing is running. Set the preference to false to avoid this, but be prepared for windows to fail to raise in some instances. Note: If Wing is terminated abnormally or from the task manager, the changed value will persist until the user logs out.

Internal Name: **gui.set-win32-foreground-lock-timeout**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

gui.show-feedback-dialog

Whether feedback dialog is shown to user on quit.

Internal Name: **gui.show-feedback-dialog**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

gui.startup-show-wingtips

Controls whether or not the Wing Tips tool is shown automatically at startup of the IDE.

Internal Name: **gui.startup-show-wingtips**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

gui.use-system-qt

Use the system-provided Qt library (requires Qt4 4.8 or later). Wing comes with its own private copy of the Qt libraries for which it is built and tested. Use the system

Qt option to better integrate with the gnome or other desktop environment, however on some systems this may result in random crashing or other bugs resulting from binary incompatibilities in library versions. This preference may be overridden on the command line with the --system-qt and --private-qt command line options.

Internal Name: **gui.use-system-qt**

Data Specification: **[None, True, False]**

Default Value: **False**

gui.work-area-rect

Rectangle to use for the IDE work area on screen. All windows open within this area. Format is (x, y, width, height), or use None for full screen.

Internal Name: **gui.work-area-rect**

Data Specification: **[None or [tuple length 4 of: <type int>, <type int>, <type int>, <type int>]]**

Default Value: **None**

Editor Preferences

consoles.auto-clear

Automatically clear the OS Commands consoles each time the command is re-executed

Internal Name: **consoles.auto-clear**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

edit.autocomplete-autoshow

Controls whether or not the completer is shown automatically during typing. When disabled, it can still be shown on demand with the Show Completer item in the Source menu.

Internal Name: **edit.autocomplete-autoshow**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

edit.dataentry-undo-grouping

Whether or not to group all edits made during data entry mode into a single undo action.

Internal Name: **edit.dataentry-undo-grouping**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

edit.fold-mime-types

Selects the mime types for which folding should be allowed when folding in general is enabled.

Internal Name: **edit.fold-mime-types**

Data Specification: **[list of: <type str>]**

Default Value: **['text/x-python', 'text/x-c-source', 'text/x-cpp-source', 'text/x-jav a-source', 'tex**

edit.gtk-input-method

Input method used for typing characters. This is important primarily for non-Western European languages.

Internal Name: **edit.gtk-input-method**

Data Specification: **[]**

Default Value: **default**

consoles.wrap-long-lines

Wrap long output lines in OS Commands tool to fit within available display area.

Internal Name: **consoles.wrap-long-lines**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

consoles.python-prompt-after-execution

Drop into Python shell after executing any Python file in the OS Commands tool

Internal Name: **consoles.python-prompt-after-execution**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

edit.sassist-font-zoom

The amount by which to zoom font sizes in or out in the Source Assistant.

Internal Name: **edit.sassist-font-zoom**

Data Specification: **<type float>**

Default Value: **1.0**

edit.symbol-find-alpha-sort

Controls whether to sort Find Symbol dialog alphabetically or in natural file order

Internal Name: **edit.symbol-find-alpha-sort**

Data Specification: **<boolean: 0 or 1>**

Default Value: **True**

edit.symbol-find-include-args

Controls whether to include argument specs in the searchable text used in the Find Symbol dialog

Internal Name: **edit.symbol-find-include-args**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

Project Manager Preferences

proj.follow-editor

Controls whether or not the IDE will follow the current editor by expanding the project tree to show the file open in the editor.

Internal Name: **proj.follow-editor**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

proj.follow-selection

Controls whether or not the IDE will follow the current project manager selection by opening the corresponding source file in a non-sticky (auto-closing) editor. In either case, the project manager will always open a file in sticky mode when an item is double clicked or the Goto Source context menu item is used.

Internal Name: **proj.follow-selection**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

proj.open-from-project-full-paths

Match fragments to full path of the file name, rather than just the file name. Full path matching still occurs when the path separation character is included in the search pattern.

Internal Name: **proj.open-from-project-full-paths**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

Debugger Preferences

debug.auto-clear-debug-io

Enable to automatically clear the Debug I/O tool each time a new debug session is started

Internal Name: **debug.auto-clear-debug-io**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

debug.auto-show-debug-io

Controls whether and when to automatically show the Debug I/O tool when it receives output.

Internal Name: **debug.auto-show-debug-io**

Data Specification: **[False, True, first]**

Default Value: **1**

debug.default-python-exec

Sets the default Python Executable to use for debugging and source code analysis. This can be overridden on a project by project basis in Project Properties.

Internal Name: **debug.default-python-exec**

Data Specification: **[None or <type str>]**

Default Value: **None**

debug.filter-shell-history

Enable to filter shell history traversal when something is entered prior to starting traversal. When enabled, Wing will only show history items starting with the text between the start of the current item and the caret.

Internal Name: **debug.filter-shell-history**

Data Specification: **<boolean: 0 or 1>**

Default Value: **False**

main.launch-shared-file

Selects the file to use for storing and retrieving shared launch configurations. By default the file 'launch' in the user settings directory is used.

Internal Name: **main.launch-shared-file**

Data Specification: **[one of: <type NoneType>, <type str>]**

Default Value: **None**

debug.shell-auto-restart-before-eval

Auto-restart the Python Shell before a file is evaluated within it. When this is disabled, be aware that previously defined symbols will linger in the Python Shell environment.

Internal Name: **debug.shell-auto-restart-before-eval**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

debug.shell-eval-whole-lines

Evaluate whole lines from editor rather than the exact selection, when a selection from the editor is sent to the Python Shell tool.

Internal Name: **debug.shell-eval-whole-lines**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

debug.shell-pasted-line-threshold

The number of lines after which the Python Shell will just print a summary rather than the actual lines of code pasted, dragged, or other transferred to the shell.

Internal Name: **debug.shell-pasted-line-threshold**

Data Specification: **<type int>**

Default Value: **30**

debug.show-exceptions-tip

Used internally to show information about exception handling to new users. Once turned off, it is never turned on again

Internal Name: **debug.show-exceptions-tip**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

debug.stop-timeout

Number of seconds to wait before the debugger will stop in its own code after a pause request is received and no other Python code is reached.

Internal Name: **debug.stop-timeout**

Data Specification: **<type float>, <type int>**

Default Value: **3.0**

debug.use-members-attrib

Set this to true to have the debug server use the __members__ attribute to try to interpret otherwise opaque data values. This is a preference because some extension modules contain bugs that result in crashing if this attribute is accessed. Note that __members__ has been deprecated since Python version 2.2.

Internal Name: **debug.use-members-attrib**

Data Specification: **<boolean: 0 or 1>**

Default Value: **1**

debug.warn-stale-shell

Enable to display a dialog when the Python Shell state no longer matches the configured Python Executable and/or Python Path.

Internal Name: **debug.warn-stale-shell**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

debug.wrap-debug-io

Enables line wrapping in the integrated Debug I/O tool.

Internal Name: **debug.wrap-debug-io**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

debug.wrap-debug-probe

Enables line wrapping in the Debug Probe.

Internal Name: **debug.wrap-debug-probe**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

debug.wrap-python-shell

Enables line wrapping in the Python Shell.

Internal Name: **debug.wrap-python-shell**

Data Specification: **<boolean: 0 or 1>**

Default Value: **0**

Command Reference

This chapter describes the entire top-level command set of Wing IDE. Use this reference to look up command names for use in modified keyboard bindings.

Commands that list arguments of type **<numeric modifier>** accept either a number or previously entered numeric modifier. This is used with key bindings that provide a way to enter a numeric modifier (such as **Esc 1 2 3** in the emacs personality or typing numerals in browse mode in the vi personality).

20.1. Top-level Commands

Application Control Commands

These are the high level application control commands.

abandon-changes (confirm=True)

Abandon any changes in the current document and reload it from disk. Prompts for user to confirm the operation unless either there are no local changes being abandoned or confirm is set to False.

about-application ()

Show the application-wide about box

begin-visited-document-cycle (move_back=True, back_key=None, forward_key=None)

Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released. *Key Binding: Ctrl-Tab invokes begin-visited-document-cycle(move_back=True)*

bookmarks-menu-items ()

Returns list of menu items for selecting among defined bookmarks

check-for-updates ()

Check for updates to Wing IDE and offer to install any that are available

close (ignore_changes=False, close_window=True, can_quit=False)

Close active document. Abandon any changes when ignore_changes is True. Close empty windows when close_window is true and quit if all document windows closed when can_quit is true. *Key Bindings: Wing IDE: Ctrl-F4; Brief: Ctrl-F4; VI/VIM: Ctrl-F4; Eclipse: Ctrl-F4; Emacs: Ctrl-F4; Visual Studio: Ctrl-F4; OS X: Command-F4*

close-all (omit_current=False, ignore_changes=False, close_window=False)

Close all documents in the current window, or in all windows if in one-window-per-editor windowing policy. Leave currently visible documents (or active window in one-window-per-editor-mode) if omit_current is True. Abandons changes rather than saving them when ignore_changes is True. Close empty window and quit if all document windows closed when close_window is True. *Key Bindings: Eclipse: Ctrl-Shift-F2*

close-window ()

Close the current window and all documents and panels in it *Key Bindings: Wing IDE: Alt-F4; Brief: Alt-F4; VI/VIM: Alt-F4; Eclipse: Alt-F4; Emacs: Alt-F4; Visual Studio: Alt-F4; OS X: Option-F4*

command-by-name (command_name)

Execute given command by name, collecting any args as needed *Key Bindings: Wing IDE: Ctrl-F12; Brief: Ctrl-F12; VI/VIM: Ctrl-F12; Eclipse: Ctrl-F12; Emacs: Ctrl-F12; Visual Studio: Ctrl-F12; OS X: Command-F12*

copy-tutorial ()

Prompt user and copy the tutorial directory from the Wing IDE installation to the directory selected by the user

edit-file-sets ()

Show the File Sets preference editor

edit-preferences-file ()

Edit the preferences as a text file

enter-license ()

Enter a new license code, replacing any existing license activation

execute-file (loc=None)

Execute the file at the given location or use the active view if loc is None. *Key Bindings: Eclipse: Ctrl-U*

execute-os-command (title)

Execute one of the stored commands in the OS Commands tool, selecting it by its title

execute-os-command-by-id (id, raise_panel=True)

Execute one of the stored commands in the OS Commands tool, selecting it by its internal ID

execute-process (cmd_line)

Execute the given command line in the OS Commands tool using default run directory and environment as defined in project properties, or the values set in an existing command with the same command line in the OS Commands tool. *Key Bindings: Emacs: Alt-!*

fileset-load (name)

Load the given named file set

fileset-manage ()

Display the file set manager dialog

fileset-new-with-open-files (file_set_name)

Create a new named file set with the currently open files

fileset-new-with-selected-files (file_set_name)

Create a new named file set with the currently selected files

goto-bookmark (mark)

Goto named bookmark *Key Bindings: Wing IDE: Ctrl-Alt-G; Eclipse: Ctrl-Alt-G; Emacs: Ctrl-X R B; Visual Studio: Ctrl-Alt-G; OS X: Command-Shift-D*

goto-next-bookmark (current_file_only=False)

Go to the next bookmark, or the first one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True. *Key Bindings: Wing IDE: Ctrl-Alt-Right; Brief: Ctrl-Alt-Right; VI/VIM: Ctrl-Alt-Right; Eclipse: Ctrl-Alt-Right; Emacs: Ctrl-Alt-Right; Visual Studio: Ctrl-Alt-Right*

goto-previous-bookmark (current_file_only=False)

Go to the previous bookmark in the bookmark list, or the last one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True. *Key Bindings: Wing IDE: Ctrl-Alt-Left; Brief: Ctrl-Alt-Left; VI/VIM: Ctrl-Alt-Left; Eclipse: Ctrl-Alt-Left; Emacs: Ctrl-Alt-Left; Visual Studio: Ctrl-Alt-Left*

hide-line-numbers ()

Hide line numbers in editors

initiate-numeric-modifier (digit)

VI style repeat/numeric modifier for following command *Key Bindings: VI/VIM: 1 invokes initiate-numeric-modifier(digit=1)*

initiate-repeat ()

Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke. *Key Bindings: Emacs: Ctrl-U*

initiate-repeat-0 ()

Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke. *Key Bindings: Emacs: Alt-0*

initiate-repeat-1 ()

Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke. *Key Bindings: Emacs: Alt-1*

initiate-repeat-2 ()

Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke. *Key Bindings: Emacs: Alt-2*

initiate-repeat-3 ()

Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke. *Key Bindings: Emacs: Alt-3*

initiate-repeat-4 ()

Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke. *Key Bindings: Brief: Ctrl-R; Emacs: Alt-4*

initiate-repeat-5 ()

Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke. *Key Bindings: Emacs: Alt-5*

initiate-repeat-6 ()

Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke. *Key Bindings: Emacs: Alt-6*

initiate-repeat-7 ()

Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke. *Key Bindings: Emacs: Alt-7*

initiate-repeat-8 ()

Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke. *Key Bindings: Emacs: Alt-8*

initiate-repeat-9 ()

Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke. *Key Bindings: Emacs: Alt-9*

internal-profile-start ()

Start internal profiling. Profile information is collected for Wing IDE's internals until internal_profile_stop is executed.

internal-profile-stop ()

Stop internal profiling after earlier internal_profile_start command. The profile can be found in the error-log file or submitted to Wingware as part of the error log included with a bug report from the Help menu.

new-blank-file (filename)

Create a new blank file on disk, open it in an editor, and add it to the current project.

new-directory (filename)

Create a new directory on disk and add it to the current project.

new-document-window ()

Create a new document window with same documents and panels as in the current document window (if any; otherwise empty with default panels) *Key Bindings: Emacs: Ctrl-X 5 2; OS X: Shift-F4*

new-file (ext='.py')

Create a new file *Key Bindings: Wing IDE: Ctrl-N; Eclipse: Ctrl-N; Visual Studio: Ctrl-N; OS X: Command-N*

new-package (filename)

Create a new Python package directory on disk, add it to the current project, and open the new __init__.py in the editor.

new-panel-window (panel_type=None)

Create a new panel window of given type

next-document (repeat=<numeric modifier; default=1>)

Move to the next document alphabetically in the list of documents open in the current window *Key Bindings: Wing IDE: Ctrl-Page_Down; Brief: Ctrl-Page_Down; VI/VIM: Ctrl-Page_Down; Eclipse: Ctrl-Page_Down; Emacs: Ctrl-Page_Down; Visual Studio: Ctrl-Page_Down; OS X: Command-0*

next-window ()

Switch to the next window alphabetically by title *Key Bindings: Wing IDE: Ctrl-Comma; Eclipse: Ctrl-Comma; Emacs: Ctrl-O; Visual Studio: Ctrl-Comma*

nth-document (n=<numeric modifier; default=0>)

Move to the nth document alphabetically in the list of documents open in the current window *Key Bindings: VI/VIM: Ctrl-^*

open (filename)

Open a file from disk using keyboard-driven selection of the file

open-from-keyboard (filename)

Open a file from disk using keyboard-driven selection of the file *Key Bindings: Wing IDE: Ctrl-K; Eclipse: Ctrl-K; Emacs: Ctrl-X Ctrl-F; Visual Studio: Ctrl-K Ctrl-O*

open-from-project (fragment='', skip_if_unique=False)

Open document from the project via the Open From Project dialog. The given fragment is used as the initial fragment filter and if it is None, the selected text or the symbol under the cursor is used. If skip_if_unique is true, the file is opened without the dialog being displayed if only one filename matches the fragment. *Key Bindings: Wing IDE: Ctrl-Shift-O; VI/VIM: Ctrl-Shift-O; Eclipse: Ctrl-Shift-O; Emacs: Ctrl-X Ctrl-O; Visual Studio: Ctrl-Shift-O; OS X: Command-Shift-O*

open-gui (filename=None)

Open a file from disk, prompting with file selection dialog if necessary *Key Bindings: Wing IDE: Ctrl-O; Brief: Alt-E; Eclipse: Ctrl-O; Visual Studio: Ctrl-O; OS X: Command-O*

perspective-disable-auto ()

Disable auto-perspectives

perspective-enable-auto ()

Enable auto-perspectives

perspective-manage ()

Display the perspectives manager dialog

perspective-restore (name)

Restore the given named perspective.

previous-document (repeat=<numeric modifier; default=1>)

Move to the previous document alphabetically in the list of documents open in the current window *Key Bindings: Wing IDE: Ctrl-Page_Up; Brief: Ctrl-Page_Up; VI/VIM: Ctrl-Page_Up; Eclipse: Ctrl-Page_Up; Emacs: Ctrl-Page_Up; Visual Studio: Ctrl-Page_Up; OS X: Command-9*

previous-window ()

Switch to the previous window alphabetically by title

query-end-session ()

Process query-end-session message on win32

quit ()

Quit the application. *Key Bindings: Wing IDE: Ctrl-Q; Brief: Alt-X; Eclipse: Ctrl-Q; Emacs: Ctrl-X Ctrl-C; Visual Studio: Ctrl-Q; OS X: Command-Q*

recent-document ()

Switches to previous document most recently visited in the current window or window set if in one-window-per-editor windowing mode. *Key Bindings: Wing IDE: Ctrl-8; Eclipse: Ctrl-8; Emacs: Ctrl-X D; Visual Studio: Ctrl-8; OS X: Command-8*

reload-scripts ()

Force reload of all scripts, from all configured script directories. This is usually only needed when adding a new script file. Existing scripts are automatically reloaded when they change on disk.

remove-bookmark (mark)

Remove the given named bookmark

remove-bookmark-current ()

Remove bookmark at current line, if any. This command is only available if there is a bookmark on the line.

rename-current-file (filename)

Rename current file, moving the file on disk if it exists.

restart-wing ()

Restart the application

restore-default-tools ()

Hide/remove all tools and restore to original default state

save (close=False, force=False)

Save active document. Also close it if close is True. *Key Bindings: Wing IDE: Ctrl-S; Brief: Alt-W; VI/VIM: Ctrl-S; Eclipse: Ctrl-S; Emacs: Ctrl-X Ctrl-S; Visual Studio: Ctrl-S; OS X: Command-S*

save-all (close_window=False)

Save all unsaved items, prompting for names for any new items that don't have a filename already. *Key Bindings: Eclipse: Ctrl-Shift-S; Visual Studio: Ctrl-Shift-S*

save-as ()

Save active document to a new file *Key Bindings: Wing IDE: Ctrl-Shift-S; Eclipse: Ctrl-Shift-S; OS X: Command-Shift-S*

scratch-document (title='Scratch', mime_type='text/plain')

Create a new scratch buffer with given title and mime type. The buffer is never marked as changed but can be saved w/ save-as.

set-bookmark (mark)

Set a bookmark at current location on the editor. Mark is the project-wide textual name of the bookmark. *Key Bindings: Wing IDE: Ctrl-Alt-M; Brief: Alt-0 invokes set-bookmark(mark="0"); Eclipse: Ctrl-Alt-M; Emacs: Ctrl-X R M; Visual Studio: Ctrl-Alt-M; OS X: Command-D*

set-bookmark-default ()

Set a bookmark at current line, using a default bookmark name for that context. This command is only available if there is not already a bookmark on the line.

show-bookmarks ()

Show a list of all currently defined bookmarks *Key Bindings: Wing IDE: Ctrl-Alt-K; Brief: Alt-J; Eclipse: Ctrl-Alt-K; Emacs: Ctrl-X R Return; Visual Studio: Ctrl-Alt-K; OS X: Command-Shift-K*

show-bug-report-dialog ()

Show the bug reporting dialog

show-document (section='manual')

Show the given documentation section *Key Bindings: OS X: Command-?*

show-feedback-dialog ()

Show the feedback submission dialog

show-file-in-editor (filename, lineno=None, col=-1, length=0)

Show the given file in the editor. Selects the code starting and given column (if >= 0) and of given length.

show-file-in-os-file-manager (filename=None)

Show the selected file in the Explorer, Finder, or other OS-provided file manager. Shows the given file, if any, or the current file selected in the GUI.

show-howtos ()

Show the How-Tos index

show-html-document (section='manual')

Show the given document section in HTML format.

show-line-numbers (show=1)

Show the line numbers in editors

show-manual-html ()

Show the HTML version of the Wing IDE users manual

show-manual-pdf ()

Show the PDF version of the Wing IDE users manual for either US Letter or A4, depending on user's print locale

show-panel (panel_type, flash=True, grab_focus=None)

Show most recently visited panel instance of given type. If no such panel exists, add one to the primary window and show it. Returns the panel view object or None if not shown. Focus is shifted to panel if grab_focus is specified and is true; if grab_focus is not specified, it defaults to the value of flash.

The valid panel types are:

project (*) browser (**) batch-search (*) interactive-search source-assistant (**) debug-data debug-stack debug-io debug-exceptions debug-breakpoints (**) debug-probe (**) debug-watch (**) debug-modules (**) python-shell messages (*) help indent (**) bookmarks (**) testing (**) open-files (*) os-command (**) snippets (**) diff (**) uses (**) refactoring (**) versioncontrol.svn (**) versioncontrol.hg (**) versioncontrol.git (**) versioncontrol.bzr (**) versioncontrol.cvs (**) versioncontrol.perforce (**)

(*) Wing Personal and Pro only (**) Wing Pro only *Key Bindings: Eclipse: Ctrl-E invokes show-panel(panel_type="open-files")*

show-panel-batch-search (flash=True, grab_focus=None)

Not documented

show-panel-bookmarks (flash=True, grab_focus=None)

Not documented

show-panel-browser (flash=True, grab_focus=None)

Not documented

show-panel-debug-breakpoints (flash=True, grab_focus=None)

Not documented

show-panel-debug-data (flash=True, grab_focus=None)

Not documented

show-panel-debug-exceptions (flash=True, grab_focus=None)

Not documented

show-panel-debug-io (flash=True, grab_focus=None)

Not documented

show-panel-debug-modules (flash=True, grab_focus=None)

Not documented

show-panel-debug-probe (flash=True, grab_focus=None)

Not documented

show-panel-debug-stack (flash=True, grab_focus=None)

Not documented

show-panel-debug-watch (flash=True, grab_focus=None)

Not documented

show-panel-diff (flash=True, grab_focus=None)

Not documented

show-panel-help (flash=True, grab_focus=None)

Not documented

show-panel-indent (flash=True, grab_focus=None)

Not documented

show-panel-interactive-search (flash=True, grab_focus=None)

Not documented

show-panel-messages (flash=True, grab_focus=None)

Not documented

show-panel-open-files (flash=True, grab_focus=None)

Not documented

show-panel-os-command (flash=True, grab_focus=None)

Not documented

show-panel-project (flash=True, grab_focus=None)

Not documented

show-panel-python-shell (flash=True, grab_focus=None)

Not documented

show-panel-refactoring (flash=True, grab_focus=None)

Not documented

show-panel-snippets (flash=True, grab_focus=None)

Not documented

show-panel-source-assistant (flash=True, grab_focus=None)

Not documented

show-panel-testing (flash=True, grab_focus=None)

Not documented

show-panel-uses (flash=True, grab_focus=None)

Not documented

show-panel-versioncontrol-bzr (flash=True, grab_focus=None)

Not documented

show-panel-versioncontrol-cvs (flash=True, grab_focus=None)

Not documented

show-panel-versioncontrol-git (flash=True, grab_focus=None)

Not documented

show-panel-versioncontrol-hg (flash=True, grab_focus=None)

Not documented

show-panel-versioncontrol-perforce (flash=True, grab_focus=None)

Not documented

show-panel-versioncontrol-svn (flash=True, grab_focus=None)

Not documented

show-pdf-document (doc='manual')

Show the given document in PDF format. One of 'manual', 'intro', or 'howtos'.

show-plugins-gui ()

Show the plugins GUI for enabling and disabling plugins

show-preferences-gui (prefname=None)

Edit the preferences file using the preferences GUI, optionally opening to the section that contains the given preference by name *Key Bindings: OS X: Command-Comma*

show-python-donate-html ()

Show the Python donations web page

show-python-for-beginners-html ()

Show the Python for Beginners web page

show-python-manual-html ()

Show the Python users manual

show-python-org-html ()

Show the python.org site home page

show-python-org-search-html ()

Show the python.org site search page

show-quickstart ()

Show the quick start guide

show-success-stories-html ()

Show the Python Success Stories page

show-support-html ()

Show the Wing IDE support site home page

show-text-registers ()

Show the contents of all non-empty text registers in a temporary editor

show-tutorial ()

Show the tutorial

show-wingtip (section='/')

Show the Wing Tips window

show-wingware-store ()

Show the Wingware store for purchasing a license

show-wingware-website ()

Show the Wingware home page

show-wingware-wiki ()

Show the Wingware wiki for sharing scripts, tips, and tricks

switch-document (document_name)

Switches to named document. Name may either be the complete name or the last path component of a path name. *Key Bindings: Emacs: Ctrl-X B; Visual Studio: Ctrl-K Ctrl-S*

toggle-bookmark ()

Set or remove a bookmark at current location on the editor. When set, the name of the bookmark is set to an auto-generated default. *Key Bindings: Wing IDE: Ctrl-Alt-T; Eclipse: Ctrl-Alt-T; Emacs: Ctrl-X R T; Visual Studio: Ctrl-Alt-T; OS X: Command-H*

toggle-bookmark-at-click ()

Set or remove a bookmark at the position in the editor where the most recent mouse click occurred. When set, the name of the bookmark is set to an auto-generated default.

toolbar-search (text, next=False, set_anchor=True, forward=True)

Search using given text and the toolbar search area. The search is always forward from the current cursor or selection position

toolbar-search-focus ()

Move focus to toolbar search entry. *Key Bindings: Wing IDE: Ctrl-D; Eclipse: Ctrl-D; Visual Studio: Ctrl-D*

toolbar-search-next (set_anchor=True)

Move to next match of text already entered in the toolbar search area

toolbar-search-prev (set_anchor=True)

Move to previous match of text already entered in the toolbar search area

vi-delete-bookmark (marks)

Remove one or more bookmarks (pass in space separated list of names)

vi-goto-bookmark ()

Goto bookmark using single character name defined by the next pressed key *Key Bindings: VI/VIM: Grave*

vi-set-bookmark ()

Set a bookmark at current location on the editor using the next key press as the name of the bookmark. *Key Bindings: VI/VIM: m*

wing-tips ()

Display interactive tip manager

write-changed-file-and-close (filename)

Write current document to given location only if it contains any changes and close it. Writes to current file name if given filename is None.

write-file (filename, start_line=None, end_line=None, follow=True)

Write current file to a new location, optionally omitting all but the lines in the given range. The editor is changed to point to the new location when follow is True. If follow is 'untitled' then the editor is changed to point to the new location only if starting with an untitled buffer and saving the whole file. Note that the editor contents will be truncated to the given start/end lines when follow is True. *Key Bindings: Emacs: Ctrl-X Ctrl-W*

write-file-and-close (filename)

Write current document to given location and close it. Saves to current file name if the given filename is None. *Key Bindings: VI/VIM: Shift-Z Shift-Z invokes write-file-and-close(filename=None)*

Dock Window Commands

Commands for windows that contain dockable tool areas. These are available for the currently active window, if any.

display-toolbox-on-left ()

Display the tall toolbox on the right.

display-toolbox-on-right ()

Display the tall toolbox on the left.

enter-fullscreen ()

Hide both the vertical and horizontal tool areas and toolbar, saving previous state so it can be restored later with exit_fullscreen *Key Binding: Shift-F2*

exit-fullscreen ()

Restore previous non-fullscreen state of all tools and tool bar *Key Binding: Shift-F2*

hide-horizontal-tools ()

Hide the horizontal tool area

hide-toolbar ()

Hide toolbars in all document windows

hide-vertical-tools ()

Hide the vertical tool area

minimize-horizontal-tools ()

Minimize the horizontal tool area *Key Binding: F1*

minimize-vertical-tools ()

Minimize the vertical tool area *Key Binding: F2*

show-horizontal-tools ()

Show the horizontal tool area *Key Binding: F1*

show-toolbar ()

Show toolbars in all document windows

show-vertical-tools ()

Show the vertical tool area *Key Binding: F2*

toggle-horizontal-tools ()

Show or minimize the horizontal tool area

toggle-vertical-tools ()

Show or minimize the vertical tool area

Document Viewer Commands

Commands for the documentation viewer. These are available when the documentation viewer has the keyboard focus.

copy ()

Copy any selected text. *Key Bindings: Wing IDE: Ctrl-Insert; Brief: Ctrl-Insert; VI/VIM: Ctrl-Insert; Eclipse: Ctrl-Insert; Emacs: Ctrl-Insert; Visual Studio: Ctrl-Insert; OS X: Command-C*

document-back ()

Go back to prior page in the history of those that have been viewed

document-contents ()

Go to the document contents page

document-forward ()

Go forward to next page in the history of those that have been viewed

document-next ()

Go to the next page in the current document

document-previous ()

Go to the previous page in the current document

isearch-backward (search_string=None, repeat=<numeric modifier; default=1>)

Initiate incremental mini-search backward from the cursor position, optionally entering the given search string. *Key Bindings: Wing IDE: Ctrl-Shift-U; Eclipse: Ctrl-Shift-U; Emacs: Ctrl-R; Visual Studio: Ctrl-Shift-U; OS X: Command-Shift-U*

isearch-backward-regex (search_string=None, repeat=<numeric modifier; default=1>)

Initiate incremental regular expression mini-search backward from the cursor position, optionally entering the given search string. *Key Bindings: VI/VIM: ?; Emacs: Ctrl-Alt-R*

isearch-forward (search_string=None, repeat=<numeric modifier; default=1>)

Initiate incremental mini-search forward from the cursor position, optionally entering the given search string. *Key Bindings: Wing IDE: Ctrl-U; Eclipse: Ctrl-U; Emacs: Ctrl-S; Visual Studio: Ctrl-I; OS X: Command-U*

isearch-forward-regex (search_string=None, repeat=<numeric modifier; default=1>)

Initiate incremental regular expression mini-search forward from the cursor position, optionally entering the given search string. *Key Bindings: VI/VIM: /; Emacs: Ctrl-Alt-S*

isearch-repeat (reverse=False, repeat=<numeric modifier; default=1>)

Repeat the most recent isearch, using same string and regex/text. Reverse direction when reverse is True. *Key Bindings: VI/VIM: n*

isearch-sel-backward (persist=True, repeat=<numeric modifier; default=1>)

Initiate incremental mini-search backward from the cursor position, using current selection as the search string. Set persist=False to do the search but end the interactive search session immediately. *Key Bindings: Wing IDE: Ctrl-Shift-B; VI/VIM: # invokes isearch-sel-backward(persist=0, whole_word=1); Eclipse: Ctrl-Shift-B; Emacs: Ctrl-C R; Visual Studio: Ctrl-Shift-B*

isearch-sel-forward (persist=True, repeat=<numeric modifier; default=1>)

Initiate incremental mini-search forward from the cursor position, using current selection as the search string. Set persist=False to do the search but end the interactive search session immediately. *Key Bindings: Wing IDE: Ctrl-B; VI/VIM: * invokes isearch-sel-forward(persist=0, whole_word=1); Eclipse: Ctrl-B; Emacs: Ctrl-C S; Visual Studio: Ctrl-B*

zoom-in ()

Increase documentation font size *Key Binding: Ctrl-+*

zoom-out ()

Decrease documentation font size *Key Binding: Ctrl--*

zoom-reset ()

Reset documentation font size to default

Global Documentation Commands

Commands for the documentation viewer that are available regardless of where the focus is.

document-search (txt=None)

Search all documentation.

Window Commands

Commands for windows in general. These are available for the currently active window, if any.

focus-current-editor ()

Move focus back to the current editor, out of any tool, if there is an active editor. *Key Bindings: Eclipse: F12*

move-editor-focus (dir=1, wrap=True)

Move focus to next or previous editor split, optionally wrapping when the end is reached. *Key Bindings: VI/VIM: Ctrl-W j invokes move-editor-focus(wrap=False); Emacs: Ctrl-X O*

move-editor-focus-first ()

Move focus to first editor split *Key Bindings: VI/VIM: Ctrl-W t*

move-editor-focus-last ()

Move focus to last editor split *Key Bindings: VI/VIM: Ctrl-W b*

move-editor-focus-previous ()

Move focus to previous editor split *Key Bindings: VI/VIM: Ctrl-W p*

move-focus ()

Move the keyboard focus forward within the Window to the next editable area *Key Binding: Shift-F1*

Wing Tips Commands

Commands for the Wing Tips tool. These are only available when the tool is visible and has focus

wingtips-close ()

Close the Wing Tips window

wingtips-contents ()

Go to the Wing Tips contents page

wingtips-next ()

Go to the next page in Wing Tips

wingtips-next-unseen ()

Go to a next unseen Wing Tips page

wingtips-previous ()

Go to the previous page in Wing Tips

20.2. Project Manager Commands

Project Manager Commands

These commands act on the project manager or on the current project, regardless of whether the project list has the keyboard focus.

add-current-file-to-project ()

Add the frontmost currently open file to project *Key Bindings: Wing IDE: Ctrl-Shift-I; Brief: Ctrl-Shift-I; VI/VIM: Ctrl-Shift-I; Eclipse: Ctrl-Shift-I; Emacs: Ctrl-Shift-I; Visual Studio: Ctrl-Shift-I; OS X: Command-Shift-I*

add-directory-to-project (loc=None, recursive=True, filter='*', include_hidden=False, gui=True)

Add directory to project.

add-file-to-project ()

Add an existing file to the project.

browse-selected-from-project ()

Browse file currently selected in the project manager

clear-project-main-debug-file ()

Clear main debug entry point to nothing, so that debugging runs the file in the current editor by default

close-project ()

Close currently open project file

debug-selected-from-project ()

Start debugging the file currently selected in the project manager

execute-selected-from-project ()

Execute the file currently selected in the project manager

new-project (show_dialog=None)

Create a new blank project. Use show_dialog to control whether the New Project dialog is shown or instead a blank new project is created. By default, the Project > Show New Project Dialog preference is used.

open-ext-selected-from-project ()

Open file currently selected in the project manager

open-project (filename=None)

Open the given project file, or prompt the user to select a file if the filename is not given.

open-selected-from-project ()

Open files currently selected in the project manager

remove-directory-from-project (loc=None, gui=True)

Remove directory from project.

remove-selection-from-project ()

Remove currently selected file or package from the project

rescan-project-directories (dirs=None, recursive=True)

Scan project directories for changes. If list of directories is not specified, currently selected directories are used.

save-project ()

Save project file.

save-project-as (filename=None)

Save project file under the given name, or prompt user for a name if the filename is not given.

set-current-as-main-debug-file ()

Set current editor file as the main debug entry point for this project

set-selected-as-main-debug-file ()

Set selected file as the main debug file for this project

show-analysis-stats ()

Show the effective Python version and path for the current configuration. This command name will be deprecated in Wing 5 and removed in Wing 6. Use show-python-environment in any new code or key bindings.

show-current-file-in-project-tool ()

Show the currently selected file in the project view, if present. The selection may be the current editor, if it has focus, or files selected in other views.

show-project-window ()

Raise the project manager window

show-python-environment ()

Show the effective Python version and path for the current configuration

use-shared-project ()

Store project in sharable (two file) format. The .wpr file can be checked into revision control or other shared with other users and machines. This is the default and the format cannot be read by Wing IDE Personal.

use-single-user-project ()

Store project single-user (one file) format, which can also be read by Wing IDE Personal.

view-directory-properties (loc=None)

Show the project manager's directory properties dialog

view-file-properties (loc=None, page=None, highlighted_attribs=None)

View project properties for a particular file (current file if none is given) *Key Bindings: Eclipse: Alt-Enter; OS X: Command-I*

view-project-as-flat-tree ()

View project as flattened directory tree from project file

view-project-as-tree ()

View project as directory tree from project file

view-project-properties (highlighted_attrib=None)

View or change project-wide properties *Key Bindings: Visual Studio: Alt-F7*

Project View Commands

Commands that are available only when the project view has the keyboard focus.

browse-selected-from-project ()

Browse file currently selected in the project manager

debug-selected-from-project ()

Start debugging the file currently selected in the project manager

execute-selected-from-project ()

Execute the file currently selected in the project manager

move-files-selected-in-project-to-trash ()

Move the files and/or directories currently selected in the project view to the trash or recycling bin

open-ext-selected-from-project ()

Open file currently selected in the project manager

open-selected-from-project ()

Open files currently selected in the project manager

remove-selection-from-project ()

Remove currently selected file or package from the project

rename-selected-in-project (new_name)

Rename the currently selected file or directory in the project view

search-in-selected-from-project ()

Search in file or directory currently selected in the project manager

set-selected-as-main-debug-file ()

Set selected file as the main debug file for this project

view-project-as-flat-tree ()

View project as flattened directory tree from project file

view-project-as-tree ()

View project as directory tree from project file

20.3. Editor Commands

Editor Browse Mode Commands

Commands available only when the editor is in browse mode (used for VI bindings and possibly others)

enter-insert-mode (pos='before')

Enter editor insert mode *Key Bindings: VI/VIM: A invokes enter-insert-mode(pos="after")*

enter-replace-mode ()

Enter editor replace mode *Key Bindings: VI/VIM: Shift-R*

enter-visual-mode (unit='char')

Enter editor visual mode. Unit should be one of 'char', 'line', or 'block'.

previous-select ()

Turn on auto-select using previous mode and selection *Key Bindings: VI/VIM: g v*

start-select-block ()

Turn on auto-select block mode *Key Bindings: Wing IDE: Shift-Ctrl-F8; Brief: Shift-Ctrl-F8; VI/VIM: Shift-Ctrl-F8; Eclipse: Shift-Ctrl-F8; Emacs: Shift-Ctrl-F8; Visual Studio: Shift-Ctrl-F8; OS X: Shift-Command-F8*

start-select-char ()

Turn on auto-select mode character by character *Key Binding: Shift-F8*

start-select-line ()

Turn on auto-select mode line by line *Key Bindings: Wing IDE: Ctrl-F8; Brief: Ctrl-F8; VI/VIM: Ctrl-F8; Eclipse: Ctrl-F8; Emacs: Ctrl-F8; Visual Studio: Ctrl-F8; OS X: Command-F8*

vi-command-by-name ()

Execute a VI command (implements ":" commands from VI) *Key Bindings: VI/VIM: :*

vi-set (command)

Perform vi's :set action. The command is the portion after :set. Currently supports ic, noic, ai, noai, number or nu, nonumber or nonu, ro, noro, sm, and nosm. Multiple options can be specied in one call as for :set ic sm ai

Editor Insert Mode Commands

Commands available only when editor is in insert mode (used for VI bindings and possibly others)

enter-browse-mode (provisional=False)

Enter editor browse mode *Key Bindings: VI/VIM: Esc*

Editor Non Modal Commands

Commands available only when the editor is in non-modal editing mode

exit-visual-mode ()

Exit visual mode and return back to default mode *Key Binding: Esc*

start-select-block ()

Turn on auto-select block mode *Key Bindings: Wing IDE: Shift-Ctrl-F8; Brief: Shift-Ctrl-F8; VI/VIM: Shift-Ctrl-F8; Eclipse: Shift-Ctrl-F8; Emacs: Shift-Ctrl-F8; Visual Studio: Shift-Ctrl-F8; OS X: Shift-Command-F8*

start-select-char ()

Turn on auto-select mode character by character *Key Binding: Shift-F8*

start-select-line ()

Turn on auto-select mode line by line *Key Bindings: Wing IDE: Ctrl-F8; Brief: Ctrl-F8; VI/VIM: Ctrl-F8; Eclipse: Ctrl-F8; Emacs: Ctrl-F8; Visual Studio: Ctrl-F8; OS X: Command-F8*

Editor Panel Commands

Commands that control splitting up an editor panel. These are available when one split in the editor panel has the keyboard focus.

split-horizontally (new=0)

Split current view horizontally. *Key Bindings: VI/VIM: Ctrl-W v; Emacs: Ctrl-X 3*

split-horizontally-open-file (filename)

Split current view horizontally and open selected file

split-vertically (new=0)

Split current view vertically. Create new editor in new view when new==1. *Key Bindings: Brief: F3; VI/VIM: Ctrl-W s; Emacs: Ctrl-X 2*

split-vertically-open-file (filename)

Split current view vertically and open selected file

unsplit (action='current')

Unsplit all editors so there's only one. Action specifies how to choose the remaining displayed editor. One of:

```
current -- Show current editor
close   -- Close current editor before unsplitting
recent  -- Change to recent buffer before unsplitting
recent-or-close -- Change to recent buffer before closing
split, or close the current buffer if there is only
one split left.
```

NOTE: The parameters for this command are subject to change in the future. *Key Bindings: Brief: F4; VI/VIM: Ctrl-W q invokes unsplit(action="close"); Emacs: Ctrl-X 1*

Editor Replace Mode Commands

Commands available only when editor is in replace mode (used for VI bindings and possibly others)

enter-browse-mode (provisional=False)

Enter editor browse mode *Key Bindings: VI/VIM: Esc*

Editor Split Commands

Commands for a particular editor split, available when the editor in that split has the keyboard focus. Additional commands affecting the editor's content are defined separately.

activate-file-option-menu ()

Activate the file menu for the editor. *Key Bindings: Wing IDE: Ctrl-1; Brief: Ctrl-1; VI/VIM: Ctrl-1; Eclipse: Ctrl-1; Emacs: Ctrl-1; Visual Studio: Ctrl-1; OS X: Command-1*

grow-split-horizontally ()

Increase width of this split

grow-split-vertically ()

Increase height of this split *Key Bindings: VI/VIM: Ctrl-W +*

shrink-split-horizontally ()

Decrease width of this split

shrink-split-vertically ()

Decrease height of this split *Key Bindings: VI/VIM: Ctrl-W -*

visit-history-next ()

Move forward in history to next visited editor position *Key Bindings: Wing IDE: Alt-Right; Brief: Alt-Right; VI/VIM: Alt-Right; Eclipse: Alt-Right; Emacs: Alt-Right; Visual Studio: Alt-Right; OS X: Ctrl-.*

visit-history-previous ()

Move back in history to previous visited editor position *Key Bindings: Wing IDE: Alt-Left; Brief: Alt-Left; VI/VIM: Alt-Left; Eclipse: Alt-Left; Emacs: Alt-Left; Visual Studio: Alt-Left; OS X: Ctrl-Comma*

Editor Visual Mode Commands

Commands available only when the editor is in visual mode (used for VI bindings and some others)

enter-browse-mode ()

Enter editor browse mode *Key Bindings: VI/VIM: Esc*

enter-insert-mode (pos='delete-sel')

Enter editor insert mode *Key Bindings: VI/VIM: A invokes enter-insert-mode(pos="after")*

enter-visual-mode (unit='char')

Alter type of editor visual mode or exit back to browse mode. Unit should be one of 'char', 'line', or 'block'.

exit-visual-mode ()

Exit visual mode and return back to default mode *Key Binding: Esc*

vi-command-by-name ()

Execute a VI command (implements ":" commands from VI) *Key Bindings: VI/VIM: :*

Active Editor Commands

Commands that only apply to editors when they have the keyboard focus. These commands are also available for the Python Shell, Debug Probe, and Debug I/O tools, which subclass the source editor, although some of the commands are modified or disabled as appropriate in those contexts.

activate-symbol-option-menu-1 ()

Activate the 1st symbol menu for the editor. *Key Bindings: Wing IDE: Ctrl-2; Brief: Ctrl-2; VI/VIM: Ctrl-2; Eclipse: Ctrl-2; Emacs: Ctrl-2; Visual Studio: Ctrl-2; OS X: Command-2*

activate-symbol-option-menu-2 ()

Activate the 2nd symbol menu for the editor. *Key Bindings: Wing IDE: Ctrl-3; Brief: Ctrl-3; VI/VIM: Ctrl-3; Eclipse: Ctrl-3; Emacs: Ctrl-3; Visual Studio: Ctrl-3; OS X: Command-3*

activate-symbol-option-menu-3 ()

Activate the 3rd symbol menu for the editor. *Key Bindings: Wing IDE: Ctrl-4; Brief: Ctrl-4; VI/VIM: Ctrl-4; Eclipse: Ctrl-4; Emacs: Ctrl-4; Visual Studio: Ctrl-4; OS X: Command-4*

activate-symbol-option-menu-4 ()

Activate the 4th symbol menu for the editor. *Key Bindings: Wing IDE: Ctrl-5; Brief: Ctrl-5; VI/VIM: Ctrl-5; Eclipse: Ctrl-5; Emacs: Ctrl-5; Visual Studio: Ctrl-5; OS X: Command-5*

activate-symbol-option-menu-5 ()

Activate the 5th symbol menu for the editor. *Key Bindings: Wing IDE: Ctrl-6; Brief: Ctrl-6; VI/VIM: Ctrl-6; Eclipse: Ctrl-6; Emacs: Ctrl-6; Visual Studio: Ctrl-6; OS X: Command-6*

backward-char (wrap=1, repeat=<numeric modifier; default=1>)

Move cursor backward one character *Key Binding: Left*

backward-char-extend (wrap=1, repeat=<numeric modifier; default=1>)

Move cursor backward one character, adjusting the selection range to new position *Key Binding: Shift-Left*

backward-char-extend-rect (wrap=1, repeat=<numeric modifier; default=1>)

Move cursor backward one character, adjusting the rectangular selection range to new position *Key Bindings: Wing IDE: Shift-Alt-Left; Brief: Shift-Alt-Left; VI/VIM: Shift-Alt-Left; Eclipse: Shift-Alt-Left; Emacs: Shift-Alt-Left; Visual Studio: Shift-Alt-Left; OS X: Ctrl-Option-Left*

backward-delete-char (repeat=<numeric modifier; default=1>)

Delete one character behind the cursor, or the current selection if not empty. *Key Bindings: Wing IDE: BackSpace; Brief: BackSpace; VI/VIM: BackSpace; Eclipse: BackSpace; Emacs: BackSpace; Visual Studio: BackSpace; OS X: Backspace*

backward-delete-word (delimiters=None, repeat=<numeric modifier; default=1>)

Delete one word behind of the cursor *Key Bindings: Wing IDE: Ctrl-BackSpace; Brief: Ctrl-BackSpace; VI/VIM: Ctrl-BackSpace; Eclipse: Ctrl-BackSpace; Emacs: Ctrl-BackSpace; Visual Studio: Ctrl-BackSpace; OS X: Ctrl-Option-Delete*

backward-page (repeat=<numeric modifier; default=1>)

Move cursor backward one page *Key Bindings: Wing IDE: Prior; Brief: Prior; VI/VIM: Prior; Eclipse: Prior; Emacs: Prior; Visual Studio: Prior; OS X: Ctrl-Up*

backward-page-extend (repeat=<numeric modifier; default=1>)

Move cursor backward one page, adjusting the selection range to new position *Key Bindings: Wing IDE: Shift-Prior; Brief: Shift-Prior; VI/VIM: Shift-Prior; Eclipse: Shift-Prior; Emacs: Shift-Prior; Visual Studio: Shift-Prior; OS X: Shift-Page_Up*

backward-paragraph (repeat=<numeric modifier; default=1>)

Move cursor backward one paragraph (to next all-whitespace line). *Key Bindings: VI/VIM: {*

backward-paragraph-extend (repeat=<numeric modifier; default=1>)

Move cursor backward one paragraph (to next all-whitespace line), adjusting the selection range to new position.

backward-tab ()

Outdent line at current position *Key Binding: Shift-Tab*

backward-word (delimiters=None, gravity='start', repeat=<numeric modifier; default=1>)

Move cursor backward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word. *Key Bindings:*

Wing IDE: Ctrl-Left; Brief: Ctrl-Left; VI/VIM: Ctrl-Left; Eclipse: Ctrl-Left; Emacs: Ctrl-Left; Visual Studio: Ctrl-Left; OS X: Option-Left

backward-word-extend (delimiters=None, gravity='start', repeat=<numeric modifier; default=1>)

Move cursor backward one word, adjusting the selection range to new position. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word. *Key Bindings: Wing IDE: Ctrl-Shift-Left; Brief: Ctrl-Shift-Left; VI/VIM: Ctrl-Shift-Left; Eclipse: Ctrl-Shift-Left; Emacs: Ctrl-Shift-Left; Visual Studio: Ctrl-Shift-Left; OS X: Ctrl-Shift-Left invokes backward-word-extend(delimiters="_`~!@#$%^&*()+-={}[]\|;:'",.<>/? trn")*

beginning-of-line (toggle=True)

Move to beginning of current line. When toggle is True, moves to the end of the leading white space if already at the beginning of the line (and vice versa). *Key Bindings: Brief: Shift-Home; VI/VIM: 0 invokes beginning-of-line(toggle=0); Emacs: Home; OS X: Command-Left*

beginning-of-line-extend (toggle=True)

Move to beginning of current line, adjusting the selection range to the new position. When toggle is True, moves to the end of the leading white space if already at the beginning of the line (and vice versa). *Key Bindings: Emacs: Shift-Home; OS X: Command-Shift-Left*

beginning-of-line-text (toggle=True)

Move to end of the leading white space, if any, on the current line. If toggle is True, moves to the beginning of the line if already at the end of the leading white space (and vice versa). *Key Bindings: Wing IDE: Home; Brief: Home; VI/VIM: Home; Eclipse: Home; Emacs: Home; Visual Studio: Home*

beginning-of-line-text-extend (toggle=True)

Move to end of the leading white space, if any, on the current line, adjusting the selection range to the new position. If toggle is True, moves to the beginning of the line if already at the end of the leading white space (and vice versa). *Key Bindings: Wing IDE: Shift-Home; Brief: Shift-Home; VI/VIM: Shift-Home; Eclipse: Shift-Home; Emacs: Shift-Home; Visual Studio: Shift-Home*

beginning-of-screen-line ()

Move to beginning of current wrapped line *Key Bindings: VI/VIM: g 0*

beginning-of-screen-line-extend ()

Move to beginning of current wrapped line, extending selection

beginning-of-screen-line-text ()

Move to first non-blank character at beginning of current wrapped line *Key Bindings: VI/VIM: g ^*

beginning-of-screen-line-text-extend ()

Move to first non-blank character at beginning of current wrapped line, extending selection

brace-match ()

Match brace at current cursor position, selecting all text between the two and hilighting the braces *Key Bindings: Wing IDE: Ctrl-E; Eclipse: Ctrl-E; Emacs: Ctrl-M; Visual Studio: Ctrl-E; OS X: Command-B*

cancel ()

Cancel current editor command

cancel-autocompletion ()

Cancel any active autocompletion.

case-lower (repeat=<numeric modifier; default=1>)

Change case of the current selection, or character ahead of the cursor if there is no selection, to lower case *Key Bindings: Visual Studio: Ctrl-U*

case-lower-next-move (repeat=<numeric modifier; default=1>)

Change case of text spanned by next cursor movement to lower case *Key Bindings: VI/VIM: g u*

case-swap (repeat=<numeric modifier; default=1>)

Change case of the current selection, or character ahead of the cursor if there is no selection, so each letter is the opposite of its current case *Key Bindings: VI/VIM: ~*

case-swap-next-move (repeat=<numeric modifier; default=1>)

Change case of text spanned by next cursor movement so each letter is the opposite of its current case *Key Bindings: VI/VIM: g ~*

case-title (repeat=<numeric modifier; default=1>)

Change case of the current selection, or character ahead of the cursor if there is no selection, to title case (first letter of each word capitalized)

case-title-next-move (repeat=<numeric modifier; default=1>)

Change case of text spanned by next cursor movement to title case (first letter of each word capitalized)

case-upper (repeat=<numeric modifier; default=1>)

Change case of the current selection, or character ahead of the cursor if there is no selection, to upper case *Key Bindings: Visual Studio: Ctrl-Shift-U*

case-upper-next-move (repeat=<numeric modifier; default=1>)

Change case of text spanned by next cursor movement to upper case *Key Bindings: VI/VIM: g Shift-U*

center-cursor ()

Scroll so cursor is centered on display *Key Bindings: Brief: Ctrl-C; VI/VIM: z .; Emacs: Ctrl-L*

clear ()

Clear selected text

clear-move-command ()

Clear any pending move command action, as for VI mode *Key Bindings: VI/VIM: Esc*

complete-autocompletion (append='')

Complete the current active autocompletion.

copy ()

Copy selected text *Key Bindings: Wing IDE: Ctrl-Insert; Brief: Ctrl-Insert; VI/VIM: Ctrl-Insert; Eclipse: Ctrl-Insert; Emacs: Ctrl-Insert; Visual Studio: Ctrl-Insert; OS X: Command-C*

copy-line ()

Copy the current lines(s) to clipboard

copy-range (start_line, end_line, target_line)

Copy the given range of lines to the given target line. Copies to current line if target_line is '.'.

copy-selection-or-line ()

Copy the current selection or current line if there is no selection. The text is placed on the clipboard.

cursor-move-to-bottom (offset=<numeric modifier; default=0>)

Move cursor to bottom of display (without scrolling), optionally at an offset of given number of lines before bottom *Key Bindings: VI/VIM: Shift-L*

cursor-move-to-center ()

Move cursor to center of display (without scrolling) *Key Bindings: VI/VIM: Shift-M*

cursor-move-to-top (offset=<numeric modifier; default=0>)

Move cursor to top of display (without scrolling), optionally at an offset of given number of lines below top *Key Bindings: VI/VIM: Shift-H*

cursor-to-bottom ()

Scroll so cursor is centered at bottom of display *Key Bindings: VI/VIM: z -*

cursor-to-top ()

Scroll so cursor is centered at top of display *Key Bindings: VI/VIM: z Return*

cut ()

Cut selected text *Key Bindings: Wing IDE: Shift-Delete; Brief: Shift-Delete; VI/VIM: Shift-Delete; Eclipse: Shift-Delete; Emacs: Shift-Delete; Visual Studio: Shift-Delete; OS X: Command-X*

cut-line ()

Cut the current line(s) to clipboard. *Key Bindings: Visual Studio: Ctrl-L*

cut-selection-or-line ()

Cut the current selection or current line if there is no selection. The text is placed on the clipboard. *Key Bindings: Visual Studio: Shift-Delete*

delete-line (repeat=<numeric modifier; default=1>)

Delete the current line or lines when the selection spans multiple lines or given repeat is > 1 *Key Bindings: Wing IDE: Ctrl-Shift-C; Eclipse: Ctrl-Shift-C*

delete-line-insert (repeat=<numeric modifier; default=1>)

Delete the current line or lines when the selection spans multiple lines or given repeat is > 1. Enters insert mode (when working with modal key bindings). *Key Bindings: VI/VIM: Shift-S*

delete-next-move (repeat=<numeric modifier; default=1>)

Delete the text covered by the next cursor move command. *Key Bindings: VI/VIM: d*

delete-next-move-insert (repeat=<numeric modifier; default=1>)

Delete the text covered by the next cursor move command and then enter insert mode (when working in a modal editor key binding) *Key Bindings: VI/VIM: c*

delete-range (start_line, end_line, register=None)

Delete given range of lines, copying them into given register (or currently selected default register if register is None

delete-to-end-of-line (repeat=<numeric modifier; default=1>, post_offset=0)

Delete everything between the cursor and end of line *Key Bindings: VI/VIM: Shift-D invokes delete-to-end-of-line(post_offset=-1)*

delete-to-end-of-line-insert (repeat=<numeric modifier; default=1>)

Delete everything between the cursor and end of line and enter insert move (when working in a modal editor key binding) *Key Bindings: VI/VIM: Shift-C*

delete-to-start-of-line ()

Delete everything between the cursor and start of line *Key Bindings: VI/VIM: Ctrl-U*

duplicate-line (pos='below')

Duplicate the current line or lines. Places the duplicate on the line following the selection if pos is 'below' or before the selection if it is 'above'. *Key Bindings: Wing IDE: Ctrl-Shift-V; Eclipse: Ctrl-Shift-V*

duplicate-line-above ()

Duplicate the current line or lines above the selection. *Key Bindings: Wing IDE: Ctrl-Shift-Y; Eclipse: Ctrl-Shift-Y*

enclose (start='(', end=')')

Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text. *Key Bindings: Wing IDE: Ctrl-(invokes enclose(start="(", end=")"); Brief: Ctrl-(invokes enclose(start="(", end=")"); VI/VIM: Ctrl-(invokes enclose(start="(", end=")"); Eclipse: Ctrl-(invokes enclose(start="(", end=")"); Emacs: Ctrl-(invokes enclose(start="(", end=")"); Visual Studio: Ctrl-(invokes enclose(start="(", end=")")*

end-of-document ()

Move cursor to end of document *Key Bindings: Wing IDE: Ctrl-End; Brief: Ctrl-End; VI/VIM: Ctrl-End; Eclipse: Ctrl-End; Emacs: Ctrl-End; Visual Studio: Ctrl-End; OS X: Command-Down*

end-of-document-extend ()

Move cursor to end of document, adjusting the selection range to new position *Key Bindings: Wing IDE: Ctrl-Shift-End; Brief: Ctrl-Shift-End; VI/VIM: Ctrl-Shift-End; Eclipse: Ctrl-Shift-End; Emacs: Ctrl-Shift-End; Visual Studio: Ctrl-Shift-End; OS X: Command-Shift-Down*

end-of-line (count=<numeric modifier; default=1>)

Move to end of current line *Key Bindings: Wing IDE: End; Brief: End; VI/VIM: End; Eclipse: End; Emacs: End; Visual Studio: End; OS X: Command-Right*

end-of-line-extend (count=<numeric modifier; default=1>)

Move to end of current line, adjusting the selection range to new position *Key Bindings: Wing IDE: Shift-End; Brief: Shift-End; VI/VIM: Shift-End; Eclipse: Shift-End; Emacs: Shift-End; Visual Studio: Shift-End; OS X: Command-Shift-Right*

end-of-screen-line (count=<numeric modifier; default=1>)

Move to end of current wrapped line *Key Bindings: VI/VIM: g $*

end-of-screen-line-extend (count=<numeric modifier; default=1>)

Move to end of current wrapped line, extending selection

exchange-point-and-mark ()

When currently marking text, this exchanges the current position and mark ends of the current selection *Key Bindings: VI/VIM: o; Emacs: Ctrl-X Ctrl-X*

filter-next-move (repeat=<numeric modifier; default=1>)

Filter the lines covered by the next cursor move command through an external command and replace the lines with the result *Key Bindings: VI/VIM: !*

filter-range (cmd, start_line=0, end_line=-1)

Filter a range of lines in the editor through an external command and replace the lines with the result. Filters the whole file by default. Filters nothing and opens up a scratch buffer with the output of the command if start_line and end_line are both -1.

filter-selection (cmd)

Filter the current selection through an external command and replace the lines with the result *Key Bindings: VI/VIM: !*

form-feed ()

Place a form feed character at the current cursor position

forward-char (wrap=1, repeat=<numeric modifier; default=1>)

Move cursor forward one character *Key Binding: Right*

forward-char-extend (wrap=1, repeat=<numeric modifier; default=1>)

Move cursor forward one character, adjusting the selection range to new position *Key Binding: Shift-Right*

forward-char-extend-rect (wrap=1, repeat=<numeric modifier; default=1>)

Move cursor forward one character, adjusting the rectangular selection range to new position *Key Bindings: Wing IDE: Shift-Alt-Right; Brief: Shift-Alt-Right; VI/VIM: Shift-Alt-Right; Eclipse: Shift-Alt-Right; Emacs: Shift-Alt-Right; Visual Studio: Shift-Alt-Right; OS X: Ctrl-Option-Right*

forward-delete-char (repeat=<numeric modifier; default=1>)

Delete one character in front of the cursor *Key Binding: Delete*

forward-delete-char-insert (repeat=<numeric modifier; default=1>)

Delete one char in front of the cursor and enter insert mode (when working in modal key bindings) *Key Bindings: VI/VIM: s*

forward-delete-char-within-line (repeat=<numeric modifier; default=1>)

Delete one character in front of the cursor unless at end of line, in which case delete backward. Do nothing if the line is empty. This is VI style 'x' in browser mode. *Key Bindings: VI/VIM: x*

forward-delete-word (delimiters=None, repeat=<numeric modifier; default=1>)

Delete one word in front of the cursor *Key Bindings: Wing IDE: Ctrl-Delete; Brief: Ctrl-Delete; VI/VIM: Ctrl-Delete; Eclipse: Ctrl-Delete; Emacs: Ctrl-Delete; Visual Studio: Ctrl-Delete; OS X: Option-Delete*

forward-delete-word-insert (delimiters=None, repeat=<numeric modifier; default=1>)

Delete one word in front of the cursor and enter insert mode (when working in modal key bindings)

forward-page (repeat=<numeric modifier; default=1>)

Move cursor forward one page *Key Bindings: Wing IDE: Next; Brief: Next; VI/VIM: Next; Eclipse: Next; Emacs: Next; Visual Studio: Next; OS X: Ctrl-Down*

forward-page-extend (repeat=<numeric modifier; default=1>)

Move cursor forward one page, adjusting the selection range to new position *Key Bindings: Wing IDE: Shift-Next; Brief: Shift-Next; VI/VIM: Shift-Next; Eclipse: Shift-Next; Emacs: Shift-Next; Visual Studio: Shift-Next; OS X: Shift-Page_Down*

forward-paragraph (repeat=<numeric modifier; default=1>)

Move cursor forward one paragraph (to next all-whitespace line). *Key Bindings: VI/VIM: }*

forward-paragraph-extend (repeat=<numeric modifier; default=1>)

Move cursor forward one paragraph (to next all-whitespace line), adjusting the selection range to new position.

forward-tab ()

Place a tab character at the current cursor position *Key Binding: Ctrl-T*

forward-word (delimiters=None, gravity='start', repeat=<numeric modifier; default=1>)

Move cursor forward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word. *Key Bindings: Wing IDE: Ctrl-Right; Brief: Ctrl-Right; VI/VIM: Ctrl-Right; Eclipse: Ctrl-Right; Emacs: Ctrl-Right; Visual Studio: Ctrl-Right; OS X: Ctrl-Right invokes forward-word(delimiters="_`~!@#$%^&*()+-={}[]\|;:'",.<>/? trn")*

forward-word-extend (delimiters=None, gravity='start', repeat=<numeric modifier; default=1>)

Move cursor forward one word, adjusting the selection range to new position. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word. *Key Bindings: Wing IDE: Ctrl-Shift-Right; Brief:*

Ctrl-Shift-Right; VI/VIM: Ctrl-Shift-Right; Eclipse: Ctrl-Shift-Right; Emacs: Ctrl-Shift-Right; Visual Studio: Ctrl-Shift-Right; OS X: Option-Shift-Right

goto-overridden-method ()

Goes to the method that is overridden by the current method

hide-selection ()

Turn off display of the current text selection

indent-to-match (toggle=False)

Indent the current line or selected region to match indentation of preceding non-blank line. Set toggle=True to indent instead of one level higher if already at the matching position. *Key Binding: Ctrl-=*

indent-to-next-indent-stop ()

Indent to next indent stop from the current position. Acts like indent command if selection covers multiple lines.

isearch-backward (search_string=None, repeat=<numeric modifier; default=1>)

Initiate incremental mini-search backward from the cursor position, optionally entering the given search string *Key Bindings: Wing IDE: Ctrl-Shift-U; Eclipse: Ctrl-Shift-U; Emacs: Ctrl-R; Visual Studio: Ctrl-Shift-U; OS X: Command-Shift-U*

isearch-backward-regex (search_string=None, repeat=<numeric modifier; default=1>)

Initiate incremental regular expression mini-search backward from the cursor position, optionally entering the given search string *Key Bindings: VI/VIM: ?; Emacs: Ctrl-Alt-R*

isearch-forward (search_string=None, repeat=<numeric modifier; default=1>)

Initiate incremental mini-search forward from the cursor position, optionally entering the given search string *Key Bindings: Wing IDE: Ctrl-U; Eclipse: Ctrl-U; Emacs: Ctrl-S; Visual Studio: Ctrl-I; OS X: Command-U*

isearch-forward-regex (search_string=None, repeat=<numeric modifier; default=1>)

Initiate incremental regular expression mini-search forward from the cursor position, optionally entering the given search string *Key Bindings: VI/VIM: /; Emacs: Ctrl-Alt-S*

isearch-repeat (reverse=False, repeat=<numeric modifier; default=1>)

Repeat the most recent isearch, using same string and regex/text. Reverse direction when reverse is True. *Key Bindings: VI/VIM: n*

isearch-sel-backward (persist=True, whole_word=False, repeat=<numeric modifier; default=1>)

Initiate incremental mini-search backward from the cursor position, using current selection as the search string. Set persist=False to do the search but end the interactive search session immediately. *Key Bindings: Wing IDE: Ctrl-Shift-B; VI/VIM: # invokes isearch-sel-backward(persist=0, whole_word=1); Eclipse: Ctrl-Shift-B; Emacs: Ctrl-C R; Visual Studio: Ctrl-Shift-B*

isearch-sel-forward (persist=True, whole_word=False, repeat=<numeric modifier; default=1>)

Initiate incremental mini-search forward from the cursor position, using current selection as the search string. Set persist=False to do the search but end the interactive search session immediately. *Key Bindings: Wing IDE: Ctrl-B; VI/VIM: * invokes isearch-sel-forward(persist=0, whole_word=1); Eclipse: Ctrl-B; Emacs: Ctrl-C S; Visual Studio: Ctrl-B*

kill-line ()

Kill rest of line from cursor to end of line, and place it into the clipboard with any other contiguously removed lines. End-of-line is removed only if there is nothing between the cursor and the end of the line. *Key Bindings: Brief: Alt-D; Emacs: Ctrl-K; OS X: Ctrl-k*

middle-of-screen-line ()

Move to middle of current wrapped line *Key Bindings: VI/VIM: g m*

middle-of-screen-line-extend ()

Move to middle of current wrapped line, extending selection

move-line-down (indent=True, repeat=<numeric modifier; default=1>)

Move the current line or lines up down line, optionally indenting to match the new position *Key Bindings: Wing IDE: Ctrl-Shift-Down; Eclipse: Ctrl-Shift-Down*

move-line-up (indent=True, repeat=<numeric modifier; default=1>)

Move the current line or lines up one line, optionally indenting to match the new position *Key Bindings: Wing IDE: Ctrl-Shift-Up; Eclipse: Ctrl-Shift-Up*

move-range (start_line, end_line, target_line)

Move the given range of lines to the given target line. Moves to current line if target_line is '.'.

move-to-register (unit='char', cut=0, num=<numeric modifier; default=1>)

Cut or copy a specified number of characters or lines, or the current selection. Set cut=1 to remove the range of text from the editor after moving to register (otherwise it is just copied). Unit should be one of 'char' or 'line' or 'sel' for current selection. *Key Bindings: VI/VIM: Shift-Y invokes move-to-register(unit="line")*

move-to-register-next-move (cut=0, repeat=<numeric modifier; default=1>)

Move the text spanned by the next cursor motion to a register *Key Bindings: VI/VIM: y*

new-line ()

Place a new line at the current cursor position *Key Binding: Return*

new-line-after ()

Place a new line after the current line *Key Bindings: Wing IDE: Ctrl-Return; Brief: Ctrl-Return; VI/VIM: Ctrl-Return; Eclipse: Ctrl-Return; Emacs: Ctrl-Return; Visual Studio: Ctrl-Return*

new-line-before ()

Place a new line before the current line *Key Bindings: Wing IDE: Shift-Return; Brief: Shift-Return; VI/VIM: Shift-Return; Eclipse: Shift-Return; Emacs: Shift-Return; Visual Studio: Shift-Return*

next-blank-line (threshold=0, repeat=<numeric modifier; default=1>)

Move to the next blank line in the file, if any. If threshold>0 then a line is considered blank if it contains less than that many characters after leading and trailing whitespace are removed. *Key Bindings: Emacs: Alt-} invokes next-blank-line(threshold=1)*

next-block (count=1, ignore_indented=True)

Select the next block. Will ignore indented blocks under the current block unless ignore_indented is False. Specify a count of more than 1 to go forward multiple blocks.

next-line (cursor='same', repeat=<numeric modifier; default=1>)

Move to screen next line, optionally repositioning character within line: 'same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char. *Key Binding: Down*

next-line-extend (cursor='same', repeat=<numeric modifier; default=1>)

Move to next screen line, adjusting the selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, 'fnb' for first non-blank char, or 'xcode' to simulate XCode style Shift-Alt line selection. *Key Binding: Shift-Down*

next-line-extend-rect (cursor='same', repeat=<numeric modifier; default=1>)

Move to next screen line, adjusting the rectangular selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char. *Key Bindings: Wing IDE: Shift-Alt-Down; Brief: Shift-Alt-Down; VI/VIM: Shift-Alt-Down; Eclipse: Shift-Alt-Down; Emacs: Shift-Alt-Down; Visual Studio: Shift-Alt-Down; OS X: Ctrl-Option-Down*

next-line-in-file (cursor='start', repeat=<numeric modifier; default=1>)

Move to next line in file, repositioning character within line: 'start' at start, 'end' at end, or 'fnb' for first non-blank char. *Key Bindings: VI/VIM: + invokes next-line-in-file(cursor="fnb")*

next-scope (count=1, sibling_only=False)

Select the next scope. Specify a count of more than 1 to go forward multiple scopes. If sibling_only is true, move only to other scopes of the same parent. *Key Bindings: Eclipse: Ctrl-Shift-Down*

next-statement (count=1, ignore_indented=True)

Select the next statement. Will ignore indented statements under the current statements unless ignore_indented is False. Specify a count of more than 1 to go forward multiple statements. *Key Bindings: Eclipse: Alt-Shift-Right*

open-line ()

Open the current line by inserting a newline after the caret *Key Bindings: Emacs: Ctrl-O*

paste ()

Paste text from clipboard *Key Bindings: Wing IDE: Shift-Insert; Brief: Shift-Insert; VI/VIM: Shift-Insert; Eclipse: Shift-Insert; Emacs: Shift-Insert; Visual Studio: Shift-Insert; OS X: Ctrl-y*

paste-register (pos=1, indent=0, cursor=-1)

Paste text from register as before or after the current position. If the register contains only lines, then the lines are pasted before or after current line (rather than at cursor). If the register contains fragments of lines, the text is pasted over the current selection or either before or after the cursor. Set pos = 1 to paste after, or -1 to paste before. Set indent=1 to indent the pasted text to match current line. Set cursor=-1 to place cursor before lines or cursor=1 to place it after lines after paste completes. *Key Bindings: VI/VIM: p*

previous-blank-line (threshold=0, repeat=<numeric modifier; default=1>)

Move to the previous blank line in the file, if any. If threshold>0 then a line is considered blank if it contains less than that many characters after leading and trailing whitespace are removed. *Key Bindings: Emacs: Alt-{ invokes previous-blank-line(threshold=1)*

previous-block (count=1, ignore_indented=True)

Select the previous block. Will ignore indented blocks under the current block unless ignore_indented is False. Specify a count of more than 1 to go backward multiple blocks.

previous-line (cursor='same', repeat=<numeric modifier; default=1>)

Move to previous screen line, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char. *Key Binding: Up*

previous-line-extend (cursor='same', repeat=<numeric modifier; default=1>)

Move to previous screen line, adjusting the selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, 'fnb' for first non-blank char, or 'xcode' to simulate XCode style Shift-Alt line selection. *Key Binding: Shift-Up*

previous-line-extend-rect (cursor='same', repeat=<numeric modifier; default=1>)

Move to previous screen line, adjusting the rectangular selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char. *Key Bindings: Wing IDE: Shift-Alt-Up; Brief: Shift-Alt-Up; VI/VIM: Shift-Alt-Up; Eclipse: Shift-Alt-Up; Emacs: Shift-Alt-Up; Visual Studio: Shift-Alt-Up; OS X: Ctrl-Option-Up*

previous-line-in-file (cursor='start', repeat=<numeric modifier; default=1>)

Move to previous line in file, repositioning character within line: 'start' at start, 'end' at end, or 'fnb' for first non-blank char. *Key Bindings: VI/VIM: - invokes previous-line-in-file(cursor="fnb")*

previous-scope (count=1, sibling_only=False)

Select the previous scope. Specify a count of more than 1 to go backward multiple scopes. If sibling_only is true, move only to other scopes of the same parent. *Key Bindings: Eclipse: Ctrl-Shift-Up*

previous-statement (count=1, ignore_indented=True)

Select the previous statement. Will ignore indented statements under the current statements unless ignore_indented is False. Specify a count of more than 1 to go back multiple statements. *Key Bindings: Eclipse: Alt-Shift-Left*

profile-editor-start ()

Turn on profiling for the current source editor

profile-editor-stop ()

Stop profiling and print stats to stdout

reanalyze-file ()

Rescan file for code analysis.

redo ()

Redo last action *Key Bindings: Wing IDE: Ctrl-Y; Brief: Ctrl-U; VI/VIM: Ctrl-R; Eclipse: Ctrl-Y; Emacs: Ctrl-.; Visual Studio: Ctrl-Y; OS X: Command-Shift-Z*

repeat-command (repeat=<numeric modifier; default=1>)

Repeat the last editor command *Key Bindings: VI/VIM: .*

repeat-search-char (opposite=0, repeat=<numeric modifier; default=1>)

Repeat the last search_char operation, optionally in the opposite direction. *Key Bindings: VI/VIM: ;*

rstrip-each-line ()

Strip trailing whitespace from each line.

scroll-text-down (repeat=<numeric modifier; default=1>, move_cursor=True)

Scroll text down a line w/o moving cursor's relative position on screen. Repeat is number of lines or if >0 and <1.0 then percent of screen. Set move_cursor to False to leave cursor in current position within the source, otherwise it is moved so the cursor remains on same screen line. *Key Bindings: Wing IDE: Ctrl-Shift-Down; Brief: Ctrl-Shift-Down; VI/VIM: Ctrl-Shift-Down; Eclipse: Ctrl-Shift-Down; Emacs: Ctrl-Shift-Down; Visual Studio: Ctrl-Shift-Down*

scroll-text-left (repeat=<numeric modifier; default=1>)

Scroll text left a column w/o moving cursor's relative position on screen. Repeat is number of columns or if >0 and <1.0 then percent of screen. *Key Bindings: VI/VIM: z l*

scroll-text-page-down (repeat=<numeric modifier; default=1>, move_cursor=True)

Scroll text down a page w/o moving cursor's relative position on screen. Repeat is number of pages or if >0 and <1.0 then percent of screen. Set move_cursor to False to leave cursor in current position within the source, otherwise it is moved so the cursor remains on same screen line.

scroll-text-page-up (repeat=<numeric modifier; default=1>, move_cursor=True)

Scroll text up a page w/o moving cursor's relative position on screen. Repeat is number of pages or if >0 and <1.0 then percent of screen. Set move_cursor to False to leave cursor in current position within the source, otherwise it is moved so the cursor remains on same screen line.

scroll-text-right (repeat=<numeric modifier; default=1>)

Scroll text right a column w/o moving cursor's relative position on screen. Repeat is number of columns or if >0 and <1.0 then percent of screen. *Key Bindings: VI/VIM: z h*

scroll-text-up (repeat=<numeric modifier; default=1>, move_cursor=True)

Scroll text up a line w/o moving cursor's relative position on screen. Repeat is number of lines or if >0 and <1.0 then percent of screen. Set move_cursor to False to leave cursor in current position within the source, otherwise it is moved so the cursor remains on same screen line. *Key Bindings: Wing IDE: Ctrl-Shift-Up; Brief:*

Ctrl-Shift-Up; VI/VIM: Ctrl-Shift-Up; Eclipse: Ctrl-Shift-Up; Emacs: Ctrl-Shift-Up; Visual Studio: Ctrl-Shift-Up

scroll-to-cursor ()

Scroll to current cursor position, if not already visible

scroll-to-end (move_caret=False)

Scroll to the end of the text in the editor. Set move_caret to control whether the caret is moved. *Key Bindings: OS X: End*

scroll-to-start (move_caret=False)

Scroll to the top of the text in the editor. Set move_caret to control whether the the caret is moved. *Key Bindings: OS X: Home*

search-char (dir=1, pos=0, repeat=<numeric modifier; default=1>, single_line=0)

Search for the given character. Searches to right if dir > 0 and to left if dir < 0. Optionally place cursor pos characters to left or right of the target (e.g., use -1 to place one to left). If repeat > 1, the Nth match is found. Set single_line=1 to search only within the current line. *Key Bindings: VI/VIM: F invokes search-char(dir=1, single_line=1)*

select-all ()

Select all text in the editor *Key Bindings: Wing IDE: Ctrl-A; Eclipse: Ctrl-A; Visual Studio: Ctrl-A; OS X: Command-A*

select-block ()

Select the block the cursor is in.

select-less ()

Select less code; undoes the last select-more command *Key Bindings: Wing IDE: Ctrl-Down; Brief: Ctrl-Down; VI/VIM: Ctrl-Down; Eclipse: Ctrl-Down; Emacs: Ctrl-Down; Visual Studio: Ctrl-Down*

select-lines ()

Select the current line or lines

select-more ()

Select more code on either the current line or larger multi-line blocks. *Key Bindings: Wing IDE: Ctrl-Up; Brief: Ctrl-Up; VI/VIM: Ctrl-Up; Eclipse: Ctrl-Up; Emacs: Ctrl-Up; Visual Studio: Ctrl-Up; OS X: Option-Up*

select-scope ()

Select the scope the cursor is in.

select-statement ()

Select the statement the cursor is in.

selection-add-next-occurence (skip_current=False, reverse=False)

Add another selection containing the text of the current selection. If skip_current is true, the current selection will be deselected. If nothing is currently selected, select the current word. Searches backwards if reverse is true. *Key Bindings: Wing IDE: Ctrl-Shift-D; Eclipse: Ctrl-Shift-D; Visual Studio: Ctrl-Shift-D; OS X: Command-Y*

set-mark-command (unit='char')

Set start of text marking for selection at current cursor position. Subsequently, all cursor move operations will automatically extend the text selection until stop-mark-command is issued. Unit defines what is selected: can be one of char, line, or block (rectangle). *Key Bindings: Emacs: Ctrl-Space*

set-register ()

Set the register to use for subsequent cut/copy/paste operations *Key Bindings: VI/VIM: "*

show-autocompleter ()

Show the auto-completer for current cursor position *Key Bindings: Wing IDE: Ctrl-space; Eclipse: Ctrl-space; Emacs: Alt-Tab; Visual Studio: Ctrl-J; OS X: Ctrl-space*

show-selection ()

Turn on display of the current text selection

smart-tab ()

Implement smart handling of tab key. The behavior varies by context as follows:

- In Non-Python code, always indents to the next indent stop
- On a non-blank line when cursor is at end or before a comment, insert tab
- On a where indent does not match the computed indent level, move to the matching indent level
- Otherwise decrease indent one level (thus a non-blank line toggles between matching position and one block higher)

start-of-document ()

Move cursor to start of document *Key Bindings: Wing IDE: Ctrl-Home; Brief: Ctrl-Home; VI/VIM: Ctrl-Home; Eclipse: Ctrl-Home; Emacs: Ctrl-Home; Visual Studio: Ctrl-Home; OS X: Command-Up*

start-of-document-extend ()

Move cursor to start of document, adjusting the selection range to new position *Key Bindings: Wing IDE: Ctrl-Shift-Home; Brief: Ctrl-Shift-Home; VI/VIM:*

Ctrl-Shift-Home; Eclipse: Ctrl-Shift-Home; Emacs: Ctrl-Shift-Home; Visual Studio: Ctrl-Shift-Home; OS X: Command-Shift-Up

stop-mark-command (deselect=True)

Stop text marking for selection at current cursor position, leaving the selection set as is. Subsequent cursor move operations will deselect the range and set selection to cursor position. Deselect immediately when deselect is True. *Key Bindings: Emacs: Ctrl-G*

swap-lines (previous=False)

Swap the line at start of current selection with the line that follows it, or the preceding line if previous is True. *Key Bindings: Wing IDE: Ctrl-Shift-L; Eclipse: Ctrl-Shift-L; Emacs: Ctrl-X Ctrl-T invokes swap-lines(previous=True)*

tab-key ()

Implement the tab key, the action of which is configurable by preference *Key Binding: Tab*

undo ()

Undo last action *Key Bindings: Wing IDE: Ctrl-Z; Brief: Alt-U; VI/VIM: u; Eclipse: Ctrl-Z; Emacs: Ctrl-/; Visual Studio: Ctrl-Z; OS X: Command-Z*

yank-line ()

Yank contents of kill buffer created with kill-line into the edit buffer *Key Bindings: Emacs: Ctrl-Y*

General Editor Commands

Editor commands that act on the current (most recently active) source editor, whether or not it currently has the keyboard focus.

check-indent-consistency ()

Check whether indents consistently use spaces or tabs throughout the file.

comment-out-region (style=None)

Comment out the selected region. The style of commenting can be controlled with the style argument: 'indented' uses the default comment style indented at end of leading white space and 'block' uses a block comment in column zero. If not given, the style configured with the Editor / Block Comment Style preference is used. Each call adds a level of commenting. *Key Bindings: Wing IDE: Ctrl-/; Eclipse: Ctrl-/; Emacs: Ctrl-C C; Visual Studio: Ctrl-K Ctrl-C; OS X: Command-'*

comment-out-toggle (style=None)

Comment out the selected lines. This command is not available if they lines are already commented out. The style of commenting can be controlled with the style argument: 'indented' uses the default comment style indented at end of leading

white space and 'block' uses a block comment in column zero. If not given, the style configured with the Editor / Block Comment Style preference is used.

comment-toggle (style=None)

Toggle commenting out of the selected lines. The style of commenting can be controlled with the style argument: 'indented' uses the default comment style indented at end of leading white space and 'block' uses a block comment in column zero. If not given, the style configured with the Editor / Block Comment Style preference is used. *Key Bindings: Wing IDE: Ctrl-.; Eclipse: Ctrl-.; Emacs: Ctrl-C #; Visual Studio: Ctrl-K Ctrl-T; OS X: Command-;*

convert-indents-to-mixed (indent_size)

Convert all lines with leading spaces to mixed tabs and spaces.

convert-indents-to-spaces-only (indent_size)

Convert all lines containing leading tabs to spaces only.

convert-indents-to-tabs-only ()

Convert all indentation to use tab characters only and no spaces

evaluate-file-in-shell (restart_shell=None)

Run the contents of the editor within the Python Shell *Key Bindings: Wing IDE: Ctrl-Alt-V; Eclipse: Ctrl-Alt-V*

evaluate-sel-in-debug-probe (whole_lines=None)

Evaluate the current selection from the editor within the Debug Probe tool. When whole_lines is set, the selection is rounded to whole lines before evaluation. When unspecified (set to None), the setting from the Shell's Option menu is used instead. *Key Bindings: Wing IDE: Ctrl-Alt-D; Eclipse: Ctrl-Alt-D*

evaluate-sel-in-shell (restart_shell=False, whole_lines=None)

Evaluate the current selection from the editor within the Python Shell tool, optionally restarting the shell first. When whole_lines is set, the selection is rounded to whole lines before evaluation. When unspecified (set to None), the setting from the Shell's Option menu is used instead. *Key Bindings: Wing IDE: Ctrl-Alt-E; Eclipse: Ctrl-Alt-E; Emacs: Ctrl-C |*

execute-kbd-macro (register='a', repeat=<numeric modifier; default=1>)

Execute most recently recorded keyboard macro. If register is None then the user is asked to enter a letter a-z for the register where the macro is filed. Otherwise, register 'a' is used by default. *Key Bindings: Wing IDE: Ctrl-M; Brief: F8; VI/VIM: @ invokes execute-kbd-macro(register=None); Eclipse: Ctrl-M; Emacs: Ctrl-X E; Visual Studio: Ctrl-M; OS X: Command-M*

fill-paragraph ()

Attempt to auto-justify the paragraph around the current start of selection *Key Bindings: Wing IDE: Ctrl-J; VI/VIM: g q q; Eclipse: Ctrl-J; Emacs: Ctrl-J; Visual Studio: Ctrl-K Ctrl-F; OS X: Command-J*

find-symbol ()

Allow user to visit point of definition of a source symbol in the current editor context by typing a fragment of the name *Key Bindings: Wing IDE: Ctrl-Shift-T; VI/VIM: Ctrl-Shift-T; Eclipse: Ctrl-Shift-T; Emacs: Ctrl-X G; Visual Studio: Ctrl-Shift-T; OS X: Command-Shift-T*

fold-collapse-all ()

Collapse all fold points in the current file *Key Bindings: Wing IDE: Alt-Home; Brief: Alt-Home; VI/VIM: Alt-Home; Eclipse: Alt-Home; Emacs: Alt-Home; Visual Studio: Alt-Home; OS X: Command-Ctrl--*

fold-collapse-all-clicked ()

Collapse the clicked fold point completely

fold-collapse-all-current ()

Collapse the current fold point completely *Key Bindings: Wing IDE: Alt-Page_Up; Brief: Alt-Page_Up; VI/VIM: Alt-Page_Up; Eclipse: Alt-Page_Up; Emacs: Alt-Page_Up; Visual Studio: Alt-Page_Up; OS X: Command--*

fold-collapse-current ()

Collapse the current fold point *Key Bindings: VI/VIM: z c; Eclipse: Ctrl--*

fold-collapse-more-clicked ()

Collapse the clicked fold point one more level

fold-collapse-more-current ()

Collapse the current fold point one more level *Key Bindings: Wing IDE: Alt-Up; Brief: Alt-Up; VI/VIM: Alt-Up; Eclipse: Alt-Up; Emacs: Alt-Up; Visual Studio: Alt-Up; OS X: Command-_*

fold-expand-all ()

Expand all fold points in the current file *Key Bindings: Wing IDE: Alt-End; Brief: Alt-End; VI/VIM: Alt-End; Eclipse: Alt-End; Emacs: Alt-End; Visual Studio: Alt-End; OS X: Command-Ctrl-**

fold-expand-all-clicked ()

Expand the clicked fold point completely

fold-expand-all-current ()

Expand the current fold point completely *Key Bindings: Wing IDE: Alt-Page_Down; Brief: Alt-Page_Down; VI/VIM: Alt-Page_Down; Eclipse: Alt-Page_Down; Emacs: Alt-Page_Down; Visual Studio: Alt-Page_Down; OS X: Command-**

fold-expand-current ()

Expand the current fold point *Key Bindings: VI/VIM: z o; Eclipse: Ctrl-+*

fold-expand-more-clicked ()

Expand the clicked fold point one more level

fold-expand-more-current ()

Expand the current fold point one more level *Key Bindings: Wing IDE: Alt-Down; Brief: Alt-Down; VI/VIM: Alt-Down; Eclipse: Alt-Down; Emacs: Alt-Down; Visual Studio: Alt-Down; OS X: Command-+*

fold-toggle ()

Toggle the current fold point *Key Bindings: Wing IDE: Alt-/; Brief: Alt-/; VI/VIM: Alt-/; Eclipse: Alt-/; Emacs: Alt-/; Visual Studio: Alt-/; OS X: Command-/*

fold-toggle-clicked ()

Toggle the clicked fold point

force-indent-style-to-match-file ()

Force the indent style of the editor to match the indent style found in the majority of the file

force-indent-style-to-mixed ()

Force the indent style of the editor to mixed use of tabs and spaces, regardless of the file contents

force-indent-style-to-spaces-only ()

Force the indent style of the editor to use spaces only, regardless of file contents

force-indent-style-to-tabs-only ()

Force the indent style of the editor to use tabs only, regardless of file contents

goto-column (column=<numeric modifier; default=0>)

Move cursor to given column *Key Bindings: VI/VIM: |*

goto-line (lineno=<numeric modifier>)

Position cursor at start of given line number *Key Bindings: Wing IDE: Ctrl-L; Brief: Alt-G; Eclipse: Ctrl-L; Emacs: Alt-G; Visual Studio: Ctrl-G; OS X: Command-L*

goto-line-select (lineno=<numeric modifier>)

Scroll to and select the given line number

goto-nth-line (lineno=<numeric modifier; default=1>, cursor='start')

Position cursor at start of given line number (1=first, -1 = last). This differs from goto-line in that it never prompts for a line number but instead uses the previously entered numeric modifier or defaults to going to line one. The cursor can be positioned at 'start', 'end', or 'fnb' for first non-blank character. *Key Bindings: VI/VIM: g g invokes goto-nth-line(cursor="fnb")*

goto-nth-line-default-end (lineno=<numeric modifier; default=0>, cursor='start')

Same as goto_nth_line but defaults to end of file if no lineno is given *Key Bindings: VI/VIM: Shift-G invokes goto-nth-line-default-end(cursor="fnb")*

goto-percent-line (percent=<numeric modifier; default=0>, cursor='start')

Position cursor at start of line at given percent in file. This uses the previously entered numeric modifier or defaults to going to line one. The cursor can be positioned at 'start', 'end', or 'fnb' for first non-blank character, or in VI mode it will do brace matching operation to reflect how VI overrides this command. *Key Bindings: VI/VIM: % invokes goto-percent-line(cursor="fnb")*

hide-all-whitespace ()

Turn off all special marks for displaying white space and end-of-line

hide-eol ()

Turn off special marks for displaying end-of-line chars

hide-indent-guides ()

Turn off special marks for displaying indent level

hide-whitespace ()

Turn off special marks for displaying white space

indent-lines (lines=None, levels=<numeric modifier; default=1>)

Indent selected number of lines from cursor position. Set lines to None to indent all the lines in current selection. Set levels to indent more than one level at a time. *Key Bindings: VI/VIM: >; Eclipse: Ctrl-| invokes indent-lines(lines=1)*

indent-next-move (num=<numeric modifier; default=1>)

Indent lines spanned by next cursor move *Key Bindings: VI/VIM: >*

indent-region (sel=None)

Indent the selected region one level of indentation. Set sel to None to use preference to determine selection behavior, or "never-select" to unselect after indent, "always-select" to always select after indent, or "retain-select" to retain current selection after indent. *Key Bindings: Wing IDE: Ctrl->; VI/VIM: Ctrl-T; Eclipse: Ctrl->; Emacs: Ctrl-C >; Visual Studio: Ctrl->; OS X: Command-]*

indent-to-match-next-move (num=<numeric modifier; default=1>)

Indent lines spanned by next cursor move to match, based on the preceding line *Key Bindings: VI/VIM: =*

insert-command (cmd)

Insert the output for the given command at current cursor position. Some special characters in the command line (if not escaped with) will be replaced as follows:

```
% -- Current file's full path name
# -- Previous file's full path name
```

insert-file (filename)

Insert a file at current cursor position, prompting user for file selection *Key Bindings: Brief: Alt-R; Emacs: Ctrl-X I*

join-lines (delim=' ', num=<numeric modifier; default=2>)

Join together specified number of lines after current line (replace newlines with the given delimiter (single space by default) *Key Bindings: VI/VIM: Shift-J*

join-selection (delim=' ')

Join together all lines in given selection (replace newlines with the given delimiter (single space by default) *Key Bindings: VI/VIM: Shift-J*

kill-buffer ()

Close the current text file *Key Bindings: Brief: Ctrl--; Emacs: Ctrl-X K*

outdent-lines (lines=None, levels=<numeric modifier; default=1>)

Outdent selected number of lines from cursor position. Set lines to None to indent all the lines in current selection. Set levels to outdent more than one level at a time. *Key Bindings: VI/VIM: <*

outdent-next-move (num=<numeric modifier; default=1>)

Outdent lines spanned by next cursor move *Key Bindings: VI/VIM: <*

outdent-region (sel=None)

Outdent the selected region one level of indentation. Set sel to None to use preference to determine selection behavior, or "never-select" to unselect after indent, "always-select" to always select after indent, or "retain-select" to retain current selection after indent. *Key Bindings: Wing IDE: Ctrl-<; VI/VIM: Ctrl-D; Eclipse: Ctrl-<; Emacs: Ctrl-C <; Visual Studio: Ctrl-<; OS X: Command-[*

page-setup ()

Show printing page setup dialog

print-view ()

Print active editor document *Key Bindings: Wing IDE: Ctrl-P; Eclipse: Ctrl-P; Visual Studio: Ctrl-P; OS X: Command-P*

query-replace (search_string, replace_string)

Initiate incremental mini-search query/replace from the cursor position. *Key Bindings: Wing IDE: Alt-comma; Eclipse: Alt-comma; Emacs: Alt-%; Visual Studio: Alt-comma; OS X: Ctrl-R*

query-replace-regex (search_string, replace_string)

Initiate incremental mini-search query/replace from the cursor position. The search string is treated as a regular expression. *Key Bindings: Wing IDE: Ctrl-Alt-Comma; Eclipse: Ctrl-Alt-Comma; Emacs: Ctrl-Alt-%; Visual Studio: Ctrl-Alt-Comma*

range-replace (search_string, replace_string, confirm, range_limit, match_limit, regex)

Initiate incremental mini-search query/replace within the given selection. This is similar to query_replace but allows some additional options:

```
confirm -- True to confirm each replace
range_limit -- None to replace between current selection start and end of document,
  1 to limit operation to current selection or to current line if selection is empty,
  (start, end) to limit operation to within given selection range, or "first|last"
  to limit operating withing given range of lines (1=first).
match_limit -- None to replace any number of matches, or limit of number of replaces.
  When set to "l" plus a number, limits to that number of matches per line,
  rather than as a whole.
regex -- Treat search string as a regular expression
```

repeat-replace (repeat=<numeric modifier; default=1>)

Repeat the last query replace or range replace operation on the current line. The first match is replaced without confirmation. *Key Bindings: VI/VIM: &*

replace-char (line_mode='multiline', num=<numeric modifier; default=1>)

Replace num characters with given character. Set line_mode to multiline to allow replacing across lines, extend to replace on current line and then extend the line length, and restrict to replace only if enough characters exist on current line after cursor position. *Key Bindings: VI/VIM: r invokes replace-char(line_mode="restrict")*

replace-string (search_string, replace_string)

Replace all occurrences of a string from the cursor position to end of file. *Key Bindings: Wing IDE: Alt-.; Eclipse: Alt-.; Emacs: Alt-@; Visual Studio: Alt-.*

replace-string-regex (search_string, replace_string)

Replace all occurrences of a string from the cursor position to end of file. The search string is treated as a regular expression. *Key Bindings: Wing IDE: Ctrl-Alt-.; Eclipse: Ctrl-Alt-.; Emacs: Ctrl-Alt-@; Visual Studio: Ctrl-Alt-.*

save-buffer ()

Save the current text file to disk

set-readonly ()

Set editor to be readonly. This cannot be done if the editor contains any unsaved edits.

set-visit-history-anchor ()

Set anchor in the visit history to go back to

set-writable ()

Set editor to be writable. This can be used to override the read-only state used initially for editors displaying files that are read-only on disk.

show-all-whitespace ()

Turn on all special marks for displaying white space and end-of-line

show-eol ()

Turn on special marks for displaying end-of-line chars

show-indent-guides ()

Turn on special marks for displaying indent level

show-indent-manager ()

Display the indentation manager for this editor file

show-whitespace ()

Turn on special marks for displaying white space

start-kbd-macro (register='a')

Start definition of a keyboard macro. If register=None then the user is prompted to enter a letter a-z under which to file the macro. Otherwise, register 'a' is used by default. *Key Bindings: Wing IDE: Ctrl-(; Brief: F7; VI/VIM: q invokes start-kbd-macro(register=None); Eclipse: Ctrl-(; Emacs: Ctrl-X (; Visual Studio: Ctrl-(; OS X: Command-(*

stop-kbd-macro ()

Stop definition of a keyboard macro *Key Bindings: Wing IDE: Ctrl-); Brief: Shift-F7; VI/VIM: q; Eclipse: Ctrl-); Emacs: Ctrl-X); Visual Studio: Ctrl-); OS X: Command-)*

toggle-auto-editing ()

Toggle the global auto-editing switch. When enabled, the editor performs the auto-edits that have been selected in the Editor > Auto-Editing preferences group.

toggle-line-wrapping ()

Toggles line wrapping preference for all editors

toggle-overtype ()

Toggle status of overtyping mode *Key Bindings: Wing IDE: Insert; Brief: Insert; VI/VIM: Insert; Eclipse: Insert; Emacs: Insert; Visual Studio: Insert*

uncomment-out-region (one_level=True)

Uncomment out the selected region if commented out. If one_level is True then each call removes only one level of commenting. *Key Bindings: Wing IDE: Ctrl-?; Eclipse: Ctrl-?; Emacs: Ctrl-C U; Visual Studio: Ctrl-K Ctrl-U; OS X: Command-"*

uncomment-out-toggle (style=None)

Remove commenting from the selected lines, if any. This command is not available if the lines are not commented out.

use-lexer-ada ()

Force syntax highlighting Ada source

use-lexer-apache-conf ()

Force syntax highlighting for Apache configuration file format

use-lexer-asm ()

Force syntax highlighting for Masm assembly language

use-lexer-ave ()

Force syntax highlighting for Avenue GIS language

use-lexer-baan ()

Force syntax highlighting for Baan

use-lexer-bash ()

Force syntax highlighting for bash scripts

use-lexer-bullant ()

Force syntax highlighting for Bullant

use-lexer-by-doctype ()

Use syntax highlighting appropriate to the file type

use-lexer-coffee-script ()

Force syntax highlighting for Coffee Script source file

use-lexer-cpp ()

Force syntax highlighting for C/C++ source *Key Bindings: Wing IDE: Ctrl-7 C; Eclipse: Ctrl-7 C; Emacs: Ctrl-X L C; Visual Studio: Ctrl-7 C; OS X: Command-7 C*

use-lexer-css2 ()

Force syntax highlighting for CSS2

use-lexer-cython ()

Force syntax highlighting for Cython source

use-lexer-diff ()

Force syntax highlighting for diff/cdiff files

use-lexer-django ()

Force syntax highlighting for Django template file

use-lexer-dos-batch ()

Force syntax highlighting for DOS batch files

use-lexer-eiffel ()

Force syntax highlighting for Eiffel source

use-lexer-errlist ()

Force syntax highlighting for error list format

use-lexer-escript ()

Force syntax highlighting for EScript

use-lexer-fortran ()

Force syntax highlighting for Fortran

use-lexer-html ()

Force syntax highlighting for HTML *Key Bindings: Wing IDE: Ctrl-7 H; Eclipse: Ctrl-7 H; Emacs: Ctrl-X L H; Visual Studio: Ctrl-7 H; OS X: Command-7 H*

use-lexer-idl ()

Force syntax highlighting for XP IDL

use-lexer-java ()

Force syntax highlighting for Java source

use-lexer-javascript ()

Force syntax highlighting for Javascript

use-lexer-latex ()

Force syntax highlighting for LaTeX

use-lexer-lisp ()

Force syntax highlighting for Lisp source

use-lexer-lout ()

Force syntax highlighting for LOUT typesetting language

use-lexer-lua ()

Force syntax highlighting for Lua

use-lexer-makefile ()

Force syntax highlighting for make files *Key Bindings: Wing IDE: Ctrl-7 M; Eclipse: Ctrl-7 M; Emacs: Ctrl-X L M; Visual Studio: Ctrl-7 M; OS X: Command-7 M*

use-lexer-mako ()

Force syntax highlighting for Mako template file

use-lexer-matlab ()

Force syntax highlighting for Matlab

use-lexer-mmixal ()

Force syntax highlighting for MMIX assembly language

use-lexer-msidl ()

Force syntax highlighting for MS IDL

use-lexer-nncrontab ()

Force syntax highlighting for NNCrontab files

use-lexer-none ()

Use no syntax highlighting *Key Bindings: Wing IDE: Ctrl-7 N; Eclipse: Ctrl-7 N; Emacs: Ctrl-X L N; Visual Studio: Ctrl-7 N; OS X: Command-7 N*

use-lexer-nsis ()

Force syntax highlighting for NSIS

use-lexer-pascal ()

Force syntax highlighting for Pascal source

use-lexer-perl ()

Force syntax highlighting for Perl source

use-lexer-php ()

Force syntax highlighting for PHP source

use-lexer-plsql ()

Force syntax highlighting for PL/SQL files

use-lexer-pov ()

Force syntax highlighting for POV ray tracer scene description language

use-lexer-properties ()

Force syntax highlighting for properties files

use-lexer-ps ()

Force syntax highlighting for Postscript

use-lexer-python ()

Force syntax highlighting for Python source *Key Bindings: Wing IDE: Ctrl-7 P; Eclipse: Ctrl-7 P; Emacs: Ctrl-X L P; Visual Studio: Ctrl-7 P; OS X: Command-7 P*

use-lexer-qss ()

Force syntax highlighting for QSS (Qt Style sheets)

use-lexer-r ()

Force syntax highlighting for R source file

use-lexer-rc ()

Force syntax highlighting for RC file format

use-lexer-ruby ()

Force syntax highlighting for Ruby source

use-lexer-scriptol ()

Force syntax highlighting for Scriptol

use-lexer-scss ()

Force syntax highlighting for SCSS formatted SASS

use-lexer-sql ()

Force syntax highlighting for SQL *Key Bindings: Wing IDE: Ctrl-7 S; Eclipse: Ctrl-7 S; Emacs: Ctrl-X L S; Visual Studio: Ctrl-7 S; OS X: Command-7 S*

use-lexer-tcl ()

Force syntax highlighting for TCL

use-lexer-vb ()

Force syntax highlighting for Visual Basic

use-lexer-vxml ()

Force syntax highlighting for VXML

use-lexer-xcode ()

Force syntax highlighting for XCode files

use-lexer-xml ()

Force syntax highlighting for XML files *Key Bindings: Wing IDE: Ctrl-7 X; Eclipse: Ctrl-7 X; Visual Studio: Ctrl-7 X; OS X: Command-7 X*

use-lexer-yaml ()

Force syntax highlighting for YAML

zoom-in ()

Zoom in, increasing the text display size temporarily by one font size *Key Binding: Ctrl-+*

zoom-out ()

Zoom out, increasing the text display size temporarily by one font size *Key Binding: Ctrl--*

zoom-reset ()

Reset font zoom factor back to zero

Shell Or Editor Commands

Commands available when working either in the shell or editor

goto-clicked-symbol-defn (other_split=False)

Goto the definition of the source symbol that was last clicked on, optionally showing the definition in another split if one is available and other_split is True. *Key Bindings: Wing IDE: Ctrl-Left_Click; Brief: Ctrl-Left_Click; VI/VIM: Ctrl-Left_Click; Eclipse: Ctrl-Left_Click; Emacs: Ctrl-Left_Click; Visual Studio: Ctrl-Left_Click; OS X: Command-Left_Click*

goto-selected-symbol-defn (other_split=False)

Goto the definition of the selected source symbol, optionally showing the definition in another split if one is available and other_split is True. *Key Binding: F4*

Bookmark View Commands

Commands available on a specific instance of the bookmark manager tool

bookmarks-remove-all (confirm=0)

Remove all bookmarks

bookmarks-selected-goto ()

Goto the selected bookmarks

bookmarks-selected-remove ()

Remove the selected bookmark

bookmarks-show-docs ()

Show the Wing IDE documentation section for the bookmarks manager

Snippet Commands

Top-level commands for code snippets

snippet (snippet_name)

Insert given snippet into current editor, selecting the snippet appropriate for that file type from universal snippets if not found. This will preprocess the snippet to match indentation style to the target file, adjusts indentation based on context, and starts inline argument collection..

snippet-file (snippet_name, mime_type='', context='all')

Create a new file with given snippet and start inline snippet argument collection. If mime type is given, a file of that type is created. Otherwise, all snippets are searched and the first found snippet of given name is used, and file type matches the type of the snippet

Snippet View Commands

Commands available on a specific instance of the snippet manager tool

snippet-add (new_snippet_name, ttype='')

Add a new snippet to the current Snippets tool page or the given page

snippet-add-file-type (file_extension)

Add a file type to the snippet manager. The file type is the file extension. It is added to the last directory on the snippet path.

snippet-assign-key-binding ()

Assign/reassign/unassign the key binding associated with the given snippet by name.

snippet-clear-key-binding ()

Clear the key binding associated with the given snippet

snippet-reload-all ()

Reload all the snippet files. The snippet manager does this automatically most of the time, but reload can be useful to cause the snippet panel display to update when snippets are added or removed from outside of Wing.

snippet-remove-file-type ()

Remove a file type from the snippet manager, including any snippets defined for it. This operates only on the last directory on the snippet path.

snippet-rename-file-type (new_file_extension)

Rename a file type to the snippet manager. The file type is the file extension. This operates on the last directory on the snippet path.

snippet-selected-copy (new_name)

Copy the selected snippet to a new name in the same context

snippet-selected-edit ()

Edit the selected snippet

snippet-selected-new-file ()

Paste the currently selected snippet into a new editor

snippet-selected-paste ()

Paste the currently selected snippet into the current editor

snippet-selected-remove ()

Remove the selected snippet

snippet-selected-rename (new_name)

Rename the selected snippet

snippet-show-docs ()

Show the Wing IDE documentation section for the snippet manager

20.4. Search Manager Commands

Toolbar Search Commands

Commands available when the tool bar search entry area has the keyboard focus.

backward-char ()

Move backward one character *Key Binding: Left*

backward-char-extend ()

Move backward one character, extending the selection *Key Binding: Shift-Left*

backward-delete-char ()

Delete character behind the cursor *Key Bindings: Wing IDE: BackSpace; Brief: BackSpace; VI/VIM: BackSpace; Eclipse: BackSpace; Emacs: BackSpace; Visual Studio: BackSpace; OS X: Backspace*

backward-delete-word ()

Delete word behind the cursor *Key Bindings: Wing IDE: Ctrl-BackSpace; Brief: Ctrl-BackSpace; VI/VIM: Ctrl-BackSpace; Eclipse: Ctrl-BackSpace; Emacs: Ctrl-BackSpace; Visual Studio: Ctrl-BackSpace; OS X: Ctrl-Option-Delete*

backward-word ()

Move backward one word *Key Bindings: Wing IDE: Ctrl-Left; Brief: Ctrl-Left; VI/VIM: Ctrl-Left; Eclipse: Ctrl-Left; Emacs: Ctrl-Left; Visual Studio: Ctrl-Left; OS X: Option-Left*

backward-word-extend ()

Move backward one word, extending the selection *Key Bindings: Wing IDE: Ctrl-Shift-Left; Brief: Ctrl-Shift-Left; VI/VIM: Ctrl-Shift-Left; Eclipse: Ctrl-Shift-Left; Emacs: Ctrl-Shift-Left; Visual Studio: Ctrl-Shift-Left; OS X: Ctrl-Shift-Left invokes backward-word-extend(delimiters="_`~!@#$%^&*()+-={}[]\|;:'",.<>/? trn")*

beginning-of-line ()

Move to the beginning of the toolbar search entry *Key Bindings: Brief: Shift-Home; VI/VIM: 0 invokes beginning-of-line(toggle=0); Emacs: Home; OS X: Command-Left*

beginning-of-line-extend ()

Move to the beginning of the toolbar search entry, extending the selection *Key Bindings: Emacs: Shift-Home; OS X: Command-Shift-Left*

copy ()

Cut selection *Key Bindings: Wing IDE: Ctrl-Insert; Brief: Ctrl-Insert; VI/VIM: Ctrl-Insert; Eclipse: Ctrl-Insert; Emacs: Ctrl-Insert; Visual Studio: Ctrl-Insert; OS X: Command-C*

cut ()

Cut selection *Key Bindings: Wing IDE: Shift-Delete; Brief: Shift-Delete; VI/VIM: Shift-Delete; Eclipse: Shift-Delete; Emacs: Shift-Delete; Visual Studio: Shift-Delete; OS X: Command-X*

end-of-line ()

Move to the end of the toolbar search entry *Key Bindings: Wing IDE: End; Brief: End; VI/VIM: End; Eclipse: End; Emacs: End; Visual Studio: End; OS X: Command-Right*

end-of-line-extend ()

Move to the end of the toolbar search entry, extending the selection *Key Bindings: Wing IDE: Shift-End; Brief: Shift-End; VI/VIM: Shift-End; Eclipse: Shift-End; Emacs: Shift-End; Visual Studio: Shift-End; OS X: Command-Shift-Right*

forward-char ()

Move forward one character *Key Binding: Right*

forward-char-extend ()

Move forward one character, extending the selection *Key Binding: Shift-Right*

forward-delete-char ()

Delete character in front of the cursor *Key Binding: Delete*

forward-delete-word ()

Delete word in front of the cursor *Key Bindings: Wing IDE: Ctrl-Delete; Brief: Ctrl-Delete; VI/VIM: Ctrl-Delete; Eclipse: Ctrl-Delete; Emacs: Ctrl-Delete; Visual Studio: Ctrl-Delete; OS X: Option-Delete*

forward-word ()

Move forward one word *Key Bindings: Wing IDE: Ctrl-Right; Brief: Ctrl-Right; VI/VIM: Ctrl-Right; Eclipse: Ctrl-Right; Emacs: Ctrl-Right; Visual Studio: Ctrl-Right; OS X: Ctrl-Right invokes forward-word(delimiters="_`~!@#$%^&*()+-={}[]\|;:'",.<>/? trn")*

forward-word-extend ()

Move forward one word, extending the selection *Key Bindings: Wing IDE: Ctrl-Shift-Right; Brief: Ctrl-Shift-Right; VI/VIM: Ctrl-Shift-Right; Eclipse: Ctrl-Shift-Right; Emacs: Ctrl-Shift-Right; Visual Studio: Ctrl-Shift-Right; OS X: Option-Shift-Right*

paste ()

Paste from clipboard *Key Bindings: Wing IDE: Shift-Insert; Brief: Shift-Insert; VI/VIM: Shift-Insert; Eclipse: Shift-Insert; Emacs: Shift-Insert; Visual Studio: Shift-Insert; OS X: Ctrl-y*

Search Manager Commands

Globally available commands defined for the search manager. These commands are available regardless of whether a search manager is visible or has keyboard focus.

batch-replace (look_in=None, use_selection=True)

Display search and replace in files tool. *Key Bindings: Wing IDE: Ctrl-Shift-R; VI/VIM: Ctrl-Shift-G; Eclipse: Ctrl-Shift-R; Emacs: Ctrl-); Visual Studio: Ctrl-Shift-R; OS X: Command-Shift-R*

batch-search (look_in=None, use_selection=True, search_text=None)

Search on current selection using the Search in Files tool. The look_in argument gets entered in the look in field if not None or ''. The current selection is put into the search field if it doesn't span multiple lines and either use_selection is true or there's nothing in the search field. The given search text is used instead, if provided *Key Bindings: Wing IDE: Ctrl-Shift-F; VI/VIM: Ctrl-Shift-F; Eclipse: Ctrl-Shift-F; Emacs: Ctrl-(; Visual Studio: Ctrl-Shift-F; OS X: Command-Shift-F*

batch-search-backward ()

Move to the previous found match in the Search in Files tool.

batch-search-forward ()

Move to the next found match in the Search in Files tool.

batch-search-pause ()

Pause the currently running batch search, if any

replace ()

Bring up the search manager in replace mode. *Key Bindings: Wing IDE: Ctrl-R; Brief: F6; Eclipse: Ctrl-R; Emacs: Ctrl-0; Visual Studio: Ctrl-R; OS X: Command-R*

replace-again ()

Replace current selection with the search manager.

replace-and-search ()

Replace current selection and search again. *Key Bindings: Wing IDE: Ctrl-I; Brief: Shift-F6; Eclipse: Ctrl-I; OS X: Command-Ctrl-R*

search ()

Bring up the search manager in search mode. *Key Bindings: Wing IDE: Alt-F3; Brief: Alt-F3; VI/VIM: Alt-F3; Eclipse: Alt-F3; Emacs: Alt-F3; Visual Studio: Alt-F3; OS X: Option-F3*

search-again (search_string='', direction=1)

Search again using the search manager's current settings.

search-backward (search_string=None)

Search again using the search manager's current settings in backward direction *Key Binding: Shift-F3*

search-forward (search_string='')

Search again using the search manager's current settings in forward direction *Key Binding: F3*

search-sel ()

Search forward using current selection

search-sel-backward ()

Search backward using current selection *Key Bindings: Wing IDE: Ctrl-Shift-F3; Brief: Ctrl-Shift-F3; VI/VIM: Ctrl-Shift-F3; Eclipse: Ctrl-Shift-F3; Emacs: Ctrl-Shift-F3; Visual Studio: Ctrl-Shift-F3; OS X: Command-Shift-F3*

search-sel-forward ()

Search forward using current selection *Key Bindings: Wing IDE: Ctrl-F3; Brief: Ctrl-F3; VI/VIM: Ctrl-F3; Eclipse: Ctrl-F3; Emacs: Ctrl-F3; Visual Studio: Ctrl-F3; OS X: Command-F3*

Search Manager Instance Commands

Commands for a particular search manager instance. These are only available when the search manager has they keyboard focus.

clear ()

Clear selected text

copy ()

Copy selected text *Key Bindings: Wing IDE: Ctrl-Insert; Brief: Ctrl-Insert; VI/VIM: Ctrl-Insert; Eclipse: Ctrl-Insert; Emacs: Ctrl-Insert; Visual Studio: Ctrl-Insert; OS X: Command-C*

cut ()

Cut selected text *Key Bindings: Wing IDE: Shift-Delete; Brief: Shift-Delete; VI/VIM: Shift-Delete; Eclipse: Shift-Delete; Emacs: Shift-Delete; Visual Studio: Shift-Delete; OS X: Command-X*

forward-tab ()

Place a forward tab at the current cursor position in search or replace string *Key Binding: Ctrl-T*

paste ()

Paste text from clipboard *Key Bindings: Wing IDE: Shift-Insert; Brief: Shift-Insert; VI/VIM: Shift-Insert; Eclipse: Shift-Insert; Emacs: Shift-Insert; Visual Studio: Shift-Insert; OS X: Ctrl-y*

20.5. Unit Testing Commands

Unit Testing Commands

Globally available commands defined for the unit testing manager. These commands are available regardless of whether a testing manager is visible or has keyboard focus.

abort-tests ()

Abort any running tests.

add-testing-file (add_current=False)

Add a file to the set of unit tests. Adds the current editor file if add_current=True. Otherwise, asks the user to select a file.

add-testing-files (locs=None)

Add a file or files to the set of unit tests. locs can be a list of filenames or locations or a single filename or location. Adds the current editor file if locs is None.

debug-all-tests ()

Debug all the tests in testing panel. *Key Bindings: Wing IDE: Ctrl-Shift-F6; Brief: Ctrl-Shift-F6; VI/VIM: Ctrl-Shift-F6; Eclipse: Ctrl-Shift-F6; Emacs: Ctrl-Shift-F6; Visual Studio: Ctrl-Shift-F6; OS X: Command-Shift-F6*

debug-clicked-tests ()

Runs the clicked test or tests, if possible. The tests are determined by the last clicked position in the active view.

debug-current-tests ()

Runs the current test or tests, if possible. The current tests are determined by the current position in the active view. *Key Bindings: Wing IDE: Ctrl-Shift-F7; Brief: Ctrl-Shift-F7; VI/VIM: Ctrl-Shift-F7; Eclipse: Ctrl-Shift-F7; Emacs: Ctrl-Shift-F7; Visual Studio: Ctrl-Shift-F7; OS X: Command-Shift-F7*

debug-failed-tests ()

Re-run all the previously failed tests in the debugger. *Key Bindings: Wing IDE: Ctrl-Alt-F6; Brief: Ctrl-Alt-F6; VI/VIM: Ctrl-Alt-F6; Eclipse: Ctrl-Alt-F6; Emacs: Ctrl-Alt-F6; Visual Studio: Ctrl-Alt-F6; OS X: Command-Option-F6*

debug-last-tests ()

Debug the last group of tests that were run. *Key Bindings: Wing IDE: Ctrl-Alt-F7; Brief: Ctrl-Alt-F7; VI/VIM: Ctrl-Alt-F7; Eclipse: Ctrl-Alt-F7; Emacs: Ctrl-Alt-F7; Visual Studio: Ctrl-Alt-F7; OS X: Command-Option-F7*

debug-selected-tests ()

Debug the tests currently selected in the testing panel.

debug-test-files (locs=None)

Debug the tests in the current editor. Uses the given file or files if locs is not None. The locations can be a list of filenames or locations or a single filename or location.

load-test-results (filename)

Load all test results from a file.

run-all-tests (debug=False)

Runs all the tests in testing panel. *Key Binding: Shift-F6*

run-clicked-tests (debug=False)

Runs the clicked test or tests, if possible. The tests are determined by the last clicked position in the active view. The tests are debugged when debug is True.

run-current-tests (debug=False)

Runs the current test or tests, if possible. The current tests are determined by the current position in the active view. The tests are debugged when debug is True. *Key Binding: Shift-F7*

run-failed-tests (debug=False)

Re-run all the previously failed tests. The tests are debugged when debug is True. *Key Bindings: Wing IDE: Alt-F6; Brief: Alt-F6; VI/VIM: Alt-F6; Eclipse: Alt-F6; Emacs: Alt-F6; Visual Studio: Alt-F6; OS X: Option-F6*

run-last-tests (debug=False)

Run again the last group of tests that were run. The tests are debugged when debug is True. *Key Bindings: Wing IDE: Alt-F7; Brief: Alt-F7; VI/VIM: Alt-F7; Eclipse: Alt-F7; Emacs: Alt-F7; Visual Studio: Alt-F7; OS X: Option-F7*

run-selected-tests (debug=False)

Run the tests currently selected in the testing panel. The tests are debugged when debug is True.

run-test-files (locs=None, debug=False)

Run or debug the tests in the current editor. Uses the given file or files instead if locs is not None. The locations list may be a list of locations or filenames or a single location or filename. The tests are debugged if debug=True.

save-all-test-results (filename)

Save all test results to a file.

scan-for-unittests (doc=None)

Scan or re-scan the current editor file for unittests

20.6. Version Control Commands

Subversion Commands

Subversion revision control system commands

svn-blame (locs=<selected files>)

Show blame / praise / annotate for selected files.

svn-checkout ()

Start the initial checkout from svn repository. Repository and working directory must be entered before the checkout.

svn-diff-recent (locs=<selected files>)

Show diffs for most recent checkin

svn-log (locs=<selected files>)

Show the revision log for the selected files in the Subversion repository

svn-resolved (locs=<selected files>)

Indicate that any conflicts are resolved

svn-update (locs=<selected files>)

Update the selected files from the Subversion repository

svn-update-project ()

Update files in project

Git Commands

git revision control system commands

git-blame (locs=<selected files>)

Show the annotated blame/praise for the selected files in the git repository

git-fetch-repository (locs=<selected files>)

Pull from repository.

git-list (locs=<selected files>)

Show the status of the given files in the git repository

git-list-branches (locs=<selected files>)

List all branches

git-log (locs=<selected files>)

Show the revision log for the selected files in the git repository

git-pull-branch (locs=<selected files>)

Pull branch from other git repository

git-push-branch (locs=<selected files>)

Push branch to other git repository

git-switch-branch (locs=<selected files>)

Switch to another branch

Bazaar Commands

Subversion revision control system commands

bzr-log (locs=<selected files>)

Show the revision log for the selected files in the bzr repository

bzr-merge-entire-branch (locs=<selected files>)

Update the selected files from the bzr repository

bzr-push-entire-branch (locs=<selected files>)

Update the selected files from the bzr repository

C V S Commands

CVS revision control system commands

cvs-add (locs=<selected files>)

Add the files to cvs

cvs-checkout ()

Start the initial checkout from cvs repository. Repository and working directory must be entered before the checkout.

cvs-commit (locs=<selected files>)

Commit the selected files to the CVS repository

cvs-commit-project ()

Commit files in project

cvs-configure ()

Configure the CVS integration

cvs-diff (locs=<selected files>)

Show the differences between working version of given files and the corresponding revision in the CVS repository

cvs-log (locs=<selected files>)

Show the revision log for the selected files in the CVS repository

cvs-project-status ()

Run status for entire project.

cvs-remove (locs=<selected files>)

Remove the selected files

cvs-revert (locs=<selected files>)

Revert the selected files

cvs-status (locs=<selected files>)

View the CVS repository status for the selected files

cvs-update (locs=<selected files>)

Update the selected files from the CVS repository

cvs-update-project ()

Update files in project

Mercurial Commands

Mercurial revision control system commands

hg-annotate (locs=<selected files>)

Show user and revision for every line in the file(s)

hg-log (locs=<selected files>)

Show the revision log for the selected files in the hg repository

hg-merge (locs=<selected files>)

Merge working directory with changes in repository

hg-pull-entire-repository (locs=<selected files>)

Pull all changes from remote repository to local repository

hg-push-entire-repository (locs=<selected files>)

Update the selected files from the hg repository

hg-update (locs=<selected files>)

Update working directory from repository

Perforce Commands

Perforce revision control system commands

perforce-add (locs=<selected files>)

Add the files to perforce

perforce-blame (locs=<selected files>)

Show blame / praise / annotate for selected files.

perforce-commit (locs=<selected files>)

Commit the selected files to the Perforce repository

perforce-commit-project ()

Commit files in project

perforce-diff (locs=<selected files>)

Show the differences between working version of given files and the corresponding revision in the Perforce repository

perforce-edit (locs=<selected files>)

Copy the selected files from the Perforce repository

perforce-log (locs=<selected files>)

Show the revision log for the selected files in the Perforce repository

perforce-project-status ()

Run status for entire project.

perforce-remove (locs=<selected files>)

Remove the selected files

perforce-resolved (locs=<selected files>)

Indicate that any conflicts are resolved

perforce-revert (locs=<selected files>)

Revert the selected files

perforce-status (locs=<selected files>)

View the Perforce repository status for the selected files

perforce-sync (locs=<selected files>)

Copy the selected files from the Perforce repository

perforce-sync-project ()

Update files in project

Version Control Command Map

Base class for version control command map classes. This provides the common (shared) commands for version control by creating them dynamically at runtime. The common commands are as follows:

```
add -- Add a file or files to the repository
remove -- Remove from the repository
commit -- Commit changes for selected files to the repository
commit-project -- Commit all changes in the project
diff -- Display differences in the local copy of a file or files
status -- Display version control status of a file or files
project-status -- Display status for all files in the project
revert -- Revert local changes to a file or files
```

Each of these will be renamed at runtime according to which version control system is active by prepending the code name of the version control system (for example, 'diff' becomes 'svn-diff'). The prepended names for all supported version control systems are:

```
svn -- Subversion
cvs -- CVS
bzr -- Bazaar
hg -- Mercurial
git -- Git
perforce -- Perforce
```

20.7. Debugger Commands

Debugger Commands

Commands that control the debugger and current debug process, if any.

break-clear ()

Clear the breakpoint on the current line *Key Binding: F9*

break-clear-all ()

Clear all breakpoints *Key Bindings: Wing IDE: Ctrl-F9; Brief: Ctrl-F9; VI/VIM: Ctrl-F9; Eclipse: Ctrl-F9; Emacs: Ctrl-F9; Visual Studio: Ctrl-F9; OS X: Command-F9*

break-clear-clicked ()

Clear the breakpoint at current click location

break-disable ()

Disable the breakpoint on current line *Key Binding: Shift-F9*

break-disable-all ()

Disable all breakpoints *Key Bindings: Wing IDE: Ctrl-Shift-F9; Brief: Ctrl-Shift-F9; VI/VIM: Ctrl-Shift-F9; Eclipse: Ctrl-Shift-F9; Emacs: Ctrl-Shift-F9; Visual Studio: Ctrl-Shift-F9*

break-disable-clicked ()

Disable the breakpoint at current click location

break-edit-cond ()

Edit condition for the breakpoint on current line

break-edit-cond-clicked ()

Edit condition for the breakpoint at the current mouse click location

break-enable ()

Enable the breakpoint on the current line *Key Binding: Shift-F9*

break-enable-all ()

Enable all breakpoints *Key Bindings: Wing IDE: Ctrl-Shift-F9; Brief: Ctrl-Shift-F9; VI/VIM: Ctrl-Shift-F9; Eclipse: Ctrl-Shift-F9; Emacs: Ctrl-Shift-F9; Visual Studio: Ctrl-Shift-F9*

break-enable-clicked ()

Enable the breakpoint at current click location

break-enable-toggle ()

Toggle whether breakpoint on current line is enabled or disabled

break-ignore ()

Ignore the breakpoint on current line for N iterations

break-ignore-clicked ()

Ignore the breakpoint at the current mouse click location for N iterations

break-set ()

Set a new regular breakpoint on current line *Key Binding: F9*

break-set-clicked ()

Set a new regular breakpoint at the current mouse click location

break-set-cond ()

Set a new conditional breakpoint on current line

break-set-cond-clicked ()

Set a new conditionalbreakpoint at the current mouse click location

break-set-temp ()

Set a new temporary breakpoint on current line

break-set-temp-clicked ()

Set a new temporary breakpoint at the current mouse click location

break-toggle ()

Toggle breakpoint at current line (creates new regular bp when one is created)

clear-exception-ignores-list ()

Clear list of exceptions being ignored during debugging

clear-var-errors ()

Clear stored variable errors so they get refetched

collapse-tree-more ()

Collapse whole selected variables display subtree one more level

create-launch-config (name)

Create a new launch configuration with the given name if it does not already exist, and then open the launch configuration attribute dialog.

create-named-entry-point (name)

Create a new named entry point if it does not already exist, and then open the named entry point attribute dialog.

debug-attach ()

Attach to an already-running debug process

debug-continue (show_dialog=None)

Continue (or start) running, to next breakpoint *Key Binding: F5*

debug-detach ()

Detach from the debug process and let it run

debug-file (show_dialog=None)

Start debugging the current file (rather than the main entry point) *Key Binding: Shift-F5*

debug-kill ()

Stop debugging *Key Bindings: Wing IDE: Ctrl-F5; Brief: Ctrl-F5; VI/VIM: Ctrl-F5; Eclipse: Ctrl-F5; Emacs: Ctrl-F5; Visual Studio: Ctrl-F5; OS X: Command-F5*

debug-move-counter ()

Move program counter to caret

debug-named-entry-point (name)

Debug the named entry point

debug-probe-clear ()

Clear debug probe.

debug-probe-evaluate-active-range ()

Evaluate the active range in the Debug Probe, if any is set

debug-probe-show-active-range ()

Show the active range set in the Debug Probe in the editor.

debug-probe-toggle-active-range ()

Toggle the active range in the Debug Probe: The active range is cleared if already set, or otherwise set using the current editor selection.

debug-restart ()

Stop and restart the current debug process

debug-show-environment ()

Show the debug run arguments and environment configuration dialog for the main entry point or current file

debug-stack-menu-items ()

Not documented

debug-stop ()

Pause free-running execution at current program counter *Key Bindings: Wing IDE: Ctrl-Shift-F5; Brief: Ctrl-Shift-F5; VI/VIM: Ctrl-Shift-F5; Eclipse: Ctrl-Shift-F5; Emacs: Ctrl-Shift-F5; Visual Studio: Ctrl-Shift-F5; OS X: Command-Shift-F5*

debug-to-clicked ()

Debug to the line at the current mouse click location

exception-always-stop ()

Always stop on exceptions, even if they are handled by the code

exception-never-stop ()

Never stop on exceptions, even if they are unhandled in the code

exception-stop-when-printed ()

Stop only on exceptions when they are about to be printed

exception-unhandled-stop ()

Stop only on exceptions that are not handled by the code

execute-named-entry-point (name)

Execute (without debugging) the named entry point

expand-tree-more ()

Expand whole selected variables display subtree deeper

force-var-reload ()

Force refetch of a value from server

frame-down ()

Move down the current debug stack *Key Binding: F12*

frame-show ()

Show the position (thread and stack frame) where the debugger originally stopped *Key Bindings: Wing IDE: Shift-F11; Brief: Shift-F11; VI/VIM: Shift-F11; Eclipse: Shift-F11; Emacs: Shift-F11; Visual Studio: Shift-F11*

frame-up ()

Move up the current debug stack *Key Binding: F11*

hide-detail ()

Show the textual value detail area

internal-extra-debugger-logging-start ()

Turn on additional logging for diagnosing problems with the debugger

internal-extra-debugger-logging-stop ()

Turn off additional logging for diagnosing problems with the debugger

manage-launch-configs ()

Display the launch config manager

manage-named-entry-points ()

Display the named entry point manager

python-shell-clear ()

Clear python shell.

python-shell-evaluate-active-range ()

Evaluate the active range in the Python Shell, if any is set

python-shell-kill ()

Kill python shell process.

python-shell-restart ()

Restart python shell.

python-shell-show-active-range ()

Show the active range set in the Python Shell in the editor.

python-shell-toggle-active-range ()

Toggle the active range in the Python Shell: The active range is cleared if already set, or otherwise set using the current editor selection.

run-build-command ()

Execute the build command defined in the project, if any

run-to-cursor ()

Run to current cursor position *Key Bindings: Wing IDE: Alt-F5; Brief: Alt-F5; VI/VIM: Alt-F5; Eclipse: Alt-F5; Emacs: Alt-F5; Visual Studio: Alt-F5*

shell-copy-with-prompts (shell=None)

Copy text from shell, including all prompts

shell-ctrl-down ()

Not documented

shell-ctrl-return ()

Not documented

shell-ctrl-up ()

Not documented

show-detail ()

Show the textual value detail area

step-into (show_dialog=None)

Step into current execution point, or start debugging at first line *Key Binding: F7*

step-out ()

Step out of the current function or method *Key Binding: F8*

step-over ()

Step over current instruction *Key Bindings: Wing IDE: Ctrl-F6; Brief: Ctrl-F6; VI/VIM: Ctrl-F6; Eclipse: Ctrl-F6; Emacs: Ctrl-F6; Visual Studio: Ctrl-F6*

step-over-block ()

Step over current block

step-over-line ()

Step over current line

step-over-statement ()

Step over current statement *Key Binding: F6*

watch (style='ref')

Watch selected variable using a direct object reference to track it

watch-expression (expr=None)

Add a new expression to the watch list

watch-module-ref ()

Watch selected value relative to a module looked up by name in sys.modules

watch-parent-ref ()

Watch selected variable using a reference to the value's parent and the key slot for the value

watch-ref ()

Watch selected variable using a direct object reference to track it

watch-symbolic ()

Watch selected value using the symbolic path to it

Debugger Watch Commands

Commands for the debugger's Watch tool (Wing IDE Professional only). These are available only when the watch tool has key board focus.

watch-clear-all ()

Clear all entries from the watch list

watch-clear-selected ()

Clear selected entry from the watch list

Call Stack View Commands

Commands available on a specific instance of the call stack tool

callstack-copy-to-clipboard ()

Copy the call stack to the clipboard, as text

callstack-set-codeline-mode (mode)

Set the code line display mode for this call stack

callstack-show-docs ()

Show documentation for the call stack manager

Exceptions Commands

Commands available when the debugger's Exceptions tool has the keyboard focus.

clear ()

Clear the exception currently shown on the display

copy ()

Copy the exception traceback to the clipboard *Key Bindings: Wing IDE: Ctrl-Insert; Brief: Ctrl-Insert; VI/VIM: Ctrl-Insert; Eclipse: Ctrl-Insert; Emacs: Ctrl-Insert; Visual Studio: Ctrl-Insert; OS X: Command-C*

Breakpoint View Commands

Commands available on a specific instance of the breakpoint manager tool

bpmanager-clear-selected ()

Clear breakpoints currently selected on the breakpoint manager

bpmanager-show-docs ()

Show documentation for the breakpoint manager

bpmanager-show-selected ()

Show source location for breakpoint currently selected on the breakpoint manager

20.8. Script-provided Add-on Commands

Debugger Extensions Script

Scripts that extend the debugger in various ways.

set-breaks-from-markers (app=[])

Scan current file for markers in the form %BP% and places breakpoints on all lines where those markers are found. A conditional breakpoint can be set if a condition follows the marker, for example %BP%:x > 10. Removes all old breakpoints first.

Django Script

A plugin that provides Django-specific functionality when a project looks like it contains Django files.

django-validate ()

Run manage.py validate

django-sql (appname)

Run manage.py sql for given app name and display the output in a scratch buffer.

django-show-docs ()

Show documentation for using Wing IDE and Django together

django-start-project (django_admin, parent_directory, site_name, superuser, superuser_email, superuser_password)

Start a new Django project with given site name and superuser account. This will prompt for the location of django-admin.py, the parent directory, and the site name to use. It then runs django-admin.py startproject, edits settings.py fields DATABASE_ENGINE and DATABASE_NAME to use sqlite3 by default, adds django.contrib.admin to INSTALLED_APPS in settings.py, runs syncdb (creating superuser account if one was given), sets up the default admin templates by copying base_site.html into the project, and then offers to create a new project in Wing and run the django-setup-wing-project command to configure the Wing IDE project for use with the new Django project.

django-run-tests-to-scratch-buffer ()

Run manage.py tests with output in a scratch buffer

django-setup-wing-project ()

Sets up a Wing project to work with an existing Django project. This assumes that you have already added files to the project so that your manage.py and settings.py files are in the project. It sets up the Python Executable project property, sets "manage.py runserver --noreload 8000" as the main debug file, sets up the Wing project environment by defining DJANGO_SITENAME and DJANGO_SETTINGS_MODULE, adds the site directory to the Python Path in the Wing project, sets TEMPLATE_DEBUG = True in the settings.py file, ensures that the Template Debugging project property is enabled, sets up the unit testing framework and file patterns in project properties.

django-start-app (appname)

Start a new application within the current Django project and add it to the INSTALLED_APPS list in the project's settings.py file.

django-sync-db ()

Run manage.py syncdb

django-run-tests ()

Run manage.py unit tests in the Testing tool

django-restart-shell ()

Show and restart the Python Shell tool, which works in the same environment as "manage.py shell". To show the tool without restarting it, use the Tools menu.

Django Script

A plugin that provides Django-specific functionality when a project looks like it contains Django files.

django-setup-wing-project ()

Sets up a Wing project to work with an existing Django project. This assumes that you have already added files to the project so that your manage.py and settings.py files are in the project. It sets up the Python Executable project property, sets "manage.py runserver --noreload 8000" as the main debug file, sets up the Wing project environment by defining DJANGO_SITENAME and DJANGO_SETTINGS_MODULE, adds the site directory to the Python Path in the Wing project, sets TEMPLATE_DEBUG = True in the settings.py file, ensures that the Template Debugging project property is enabled, sets up the unit testing framework and file patterns in project properties.

django-start-project (django_admin, parent_directory, site_name, superuser, superuser_email, superuser_password)

Start a new Django project with given site name and superuser account. This will prompt for the location of django-admin.py, the parent directory, and the site name to use. It then runs django-admin.py startproject, edits settings.py fields DATABASE_ENGINE and DATABASE_NAME to use sqlite3 by default, adds django.contrib.admin to INSTALLED_APPS in settings.py, runs syncdb (creating superuser account if one was given), sets up the default admin templates by copying base_site.html into the project, and then offers to create a new project in Wing and run the django-setup-wing-project command to configure the Wing IDE project for use with the new Django project.

Editor Extensions Script

Editor extensions that also serve as examples for scripting Wing IDE.

vs-tab (app=[])

Modified tab indentation command that acts like tab in Visual Studio.

fold-python-methods ()

Fold up all Python methods, expand all classes, and leave other fold points alone *Key Bindings: Wing IDE: Alt-1; Brief: Alt-1; VI/VIM: Alt-1; Eclipse: Alt-1; Emacs: Alt-1; Visual Studio: Alt-1; OS X: Command-Alt--*

word-list-completion (word)

Provide simple word-list driven auto-completion on the current editor

smart-cut ()

Implement a variant of clipboard cut that cuts the whole current line if there is no selection on the editor.

kill-line-with-eol (ed=[])

Variant of emacs style kill-line command that always kills the eol characters

upper-case (editor=[])

Change current selection to all upper case *Key Bindings: Eclipse: Ctrl-Shift-Y*

smart-copy ()

Implement a variant of clipboard copy that copies the whole current line if there is no selection on the editor.

hyphen-to-under (editor=[])

Change hyphens (dashes) to underscores in current selection

batch-search-current-directory ()

Not documented

lower-case (editor=[])

Change current selection to all lower case *Key Bindings: Eclipse: Ctrl-Shift-X*

cc-checkout (app=[])

Check the current file out of clearcase. This is best used with Wing configured to auto-reload unchanged files.

describe-key-briefly (key)

Display the commands that a key is bound to in the status area. Does not fully work for the vi binding.

insert-spaces-to-tab-stop (tab_size=0)

Insert spaces to reach the next tab stop (units of given tab size or editor's tab size if none is given)

fold-python-classes ()

Fold up all Python classes but leave other fold points alone *Key Bindings: Wing IDE: Alt-2; Brief: Alt-2; VI/VIM: Alt-2; Eclipse: Alt-2; Emacs: Alt-2; Visual Studio: Alt-2; OS X: Command-Ctrl-/*

vi-fold-less ()

Approximation of zm key binding in vim *Key Bindings: VI/VIM: z m*

indent-new-comment-line (app=[], ed=[])

Enter a newline, indent to match previous line and insert a comment character and a space. Assumes that auto-indent is enabled.

cursor-home ()

Bring cursor to start of line, to start of visible area, or to start of document each successive consecutive invocation of this command. *Key Bindings: Brief: Home*

open-filename-from-editor ()

Not documented

open-clicked-url-from-editor ()

Not documented

sort-selected (app=[])

Sort selected lines of text alphabetically

search-python-docs ()

Do a search on the Python documentation for the selected text in the current editor

toggle-mark-command (style='char', select_right=0)

Change between text-marking and non-text-marking mode. Style is "char" for stream select, "block" for rectangular select, and "line" for line select. Set select_right=1 to select the character to right of the cursor when marking is toggled on. *Key Bindings: Brief: Alt-M invokes toggle-mark-command(select_right=1)*

delete-selected-lines (app=[])

Delete the line or range of lines that contain the current selection. This duplicates what the editor command delete-line does. *Key Bindings: Brief: Alt-D*

insert-debug-print (app=[])

Insert a print statement to print a selected variable name and value, along with the file and line number.

under-to-hyphen (editor=[])

Change underscores to hyphens (dashes) in current selection

open-url-from-editor ()

Not documented

smart-paste (editor=[])

A variant of paste that inserts line just copied with smart-copy above current line.

remove-prompts-and-paste (ed=[])

Paste from clipboard after removing any >>> and ... prompts

convert-to-lf-lineends (app=[])

Convert the current editor to use LF style line endings

vi-fold-more ()

Approximation of zr key binding in vim *Key Bindings: VI/VIM: z r*

insert-text (text)

Insert given text at current caret location, replacing any existing selected text

convert-to-cr-lineends (app=[])

Convert the current editor to use CR style line endings

cursor-end ()

Bring cursor to end of line, to end of visible area, or to end of document each successive consecutive invocation of this command. *Key Bindings: Brief: End*

title-case (editor=[])

Change current selection to capitalize first letter of each word *Key Bindings: Emacs: Alt-C*

watch-selection ()

Add a debug watch for the selected text in the current editor

copy-filename-to-clipboard (fn=[])

Copy the filename for the currently selected file to the clipboard

comment-block-toggle ()

Toggle block comment (with ## at start) on the selected lines in editor. This is a different style of block commenting than Wing implements by default (the default in Wing is intended to work better with some of the other editor functionality) *Key Bindings: Eclipse: Ctrl-Shift-C*

surround (char)

Surround selected text with (), [], {}, "", '', <>, or ``. Arg char should be the opening character.

copy-reference (include_text=True)

Copy 'filename, lineno (scope)' optionally followed by the current line or selected lines to the clipboard. The scope is omitted if there isn't one or in a non-Python file.

open-clicked-filename-from-editor ()

Not documented

toggle-vertical-split ()

If editor is split, unsplit it and show the vertical tools panel. Otherwise, hide the vertical tools and split the editor left-right Assumes default windowing policy (combined toolbox & editor windows). Thanks to Jonathan March for this script.

convert-to-crlf-lineends (app=[])

Convert the current editor to use CR + LF style line endings

fold-python-classes-and-defs ()

Fold up all Python classes, methods, and functions but leave other fold points alone *Key Bindings: Wing IDE: Alt-3; Brief: Alt-3; VI/VIM: Alt-3; Eclipse: Alt-3; Emacs: Alt-3; Visual Studio: Alt-3; OS X: Command-=*

toggle-toolbox-separate ()

Toggle between moving the toolboxes to a separate window and the default single-window mode

Emacs Extensions Script

This file contains scripts that add emacs-like functionality not found in Wing's internal emacs support layer.

add-change-log-entry (user_name=None, email=None, changelog=None, changed_file=None, func=None, other_window=False, new_entry=False)

Add a change log entry *Key Bindings: Emacs: Ctrl-X 4 A*

Pylintpanel Script

PyLint integration for Wing IDE.

pylint-show-docs ()

Show the Wing IDE documentation section for the PyLint integration

pylint-execute (show_panel=True)

Execute pylint for the current editor

pylint-package-execute (show_panel=True)

Execute pylint on all files in the package to which the file in the current editor belongs

pylint-copy-results ()

Copies all results from the displayed pylint results list.

pylint-configure ()

Show the pylint configuration file so it can be edited

Testapi Script

Tests for Wing's scripting API.

test-api (verbose=0)

Test Wing's scripting API

Key Binding Reference

This chapter documents all the default key bindings found in the keyboard personalities provided by Wing, set by the **Personality** preference. Key bindings are listed alphabetically. In some cases commands of the same name are provided by different implementations that are selected according to keyboard focus.

When multiple commands are defined for a single key binding, the first available command in the list is invoked. In this way a single binding can, for example, show or hide a tool panel.

Additional key bindings can be added as described in keyboard bindings. All available commands are documented in the Command Reference.

21.1. Wing IDE Personality

This section documents all the default key bindings for the **Wing IDE** keyboard personality, set by the **Personality** preference.

Alt-1: **fold-python-methods** - Fold up all Python methods, expand all classes, and leave other fold points alone

Alt-2: **fold-python-classes** - Fold up all Python classes but leave other fold points alone

Alt-3: **fold-python-classes-and-defs** - Fold up all Python classes, methods, and functions but leave other fold points alone

Alt-BackSpace: **backward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word behind of the cursor ; *Toolbar Search Commands*: Delete word behind the cursor

Alt-Delete: **backward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word behind of the cursor ; *Toolbar Search Commands*: Delete word behind the cursor

Alt-Down: **fold-expand-more-current** - Expand the current fold point one more level

Alt-End: **fold-expand-all** - Expand all fold points in the current file

Alt-F11: **prev-points-of-use-match** - Display the previous match in the active points of use tool

Alt-F12: **next-points-of-use-match** - Display the next match in the active points of use tool

Alt-F3: **search** - Bring up the search manager in search mode.

Alt-F4: **close-window** - Close the current window and all documents and panels in it

Alt-F5: **run-to-cursor** - Run to current cursor position

Alt-F6: **run-failed-tests** - Re-run all the previously failed tests. The tests are debugged when debug is True.

Alt-F7: **run-last-tests** - Run again the last group of tests that were run. The tests are debugged when debug is True.

Alt-Home: **fold-collapse-all** - Collapse all fold points in the current file

Alt-Left: **visit-history-previous** - Move back in history to previous visited editor position

Alt-Page_Down: **fold-expand-all-current** - Expand the current fold point completely

Alt-Page_Up: **fold-collapse-all-current** - Collapse the current fold point completely

Alt-Return: **new-line** - Place a new line at the current cursor position

Alt-Right: **visit-history-next** - Move forward in history to next visited editor position

Alt-Slash: **fold-toggle** - Toggle the current fold point

Alt-Up: **fold-collapse-more-current** - Collapse the current fold point one more level

Alt-comma: **query-replace** - Initiate incremental mini-search query/replace from the cursor position.

Alt-period: **replace-string** - Replace all occurrences of a string from the cursor position to end of file.

BackSpace: **backward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character behind the cursor, or the current selection if not empty. ; *Toolbar Search Commands*: Delete character behind the cursor

Ctrl-0: **next-document** - Move to the next document alphabetically in the list of documents open in the current window

Ctrl-1: **activate-file-option-menu** - Activate the file menu for the editor.

Ctrl-2: **activate-symbol-option-menu-1** - Activate the 1st symbol menu for the editor.

Ctrl-3: **activate-symbol-option-menu-2** - Activate the 2nd symbol menu for the editor.

Ctrl-4: **activate-symbol-option-menu-3** - Activate the 3rd symbol menu for the editor.

Ctrl-5: **activate-symbol-option-menu-4** - Activate the 4th symbol menu for the editor.

Ctrl-6: **activate-symbol-option-menu-5** - Activate the 5th symbol menu for the editor.

Ctrl-7 C: **use-lexer-cpp** - Force syntax highlighting for C/C++ source

Ctrl-7 H: **use-lexer-html** - Force syntax highlighting for HTML

Ctrl-7 M: **use-lexer-makefile** - Force syntax highlighting for make files

Ctrl-7 N: **use-lexer-none** - Use no syntax highlighting

Ctrl-7 P: **use-lexer-python** - Force syntax highlighting for Python source

Ctrl-7 S: **use-lexer-sql** - Force syntax highlighting for SQL

Ctrl-7 X: **use-lexer-xml** - Force syntax highlighting for XML files

Ctrl-8: **recent-document** - Switches to previous document most recently visited in the current window or window set if in one-window-per-editor windowing mode.

Ctrl-9: **previous-document** - Move to the previous document alphabetically in the list of documents open in the current window

Ctrl-=: **indent-to-match** - Indent the current line or selected region to match indentation of preceding non-blank line. Set toggle=True to indent instead of one level higher if already at the matching position.

Ctrl-A: **select-all** - Select all text in the editor

Ctrl-Alt-B: **search-sel-backward** - Search backward using current selection

Ctrl-Alt-Comma: **query-replace-regex** - Initiate incremental mini-search query/replace from the cursor position. The search string is treated as a regular expression.

Ctrl-Alt-D: **evaluate-sel-in-debug-probe** - Evaluate the current selection from the editor within the Debug Probe tool. When whole_lines is set, the selection is rounded to whole lines before evaluation. When unspecified (set to None), the setting from the Shell's Option menu is used instead.

Ctrl-Alt-D: **selection-add-next-occurence(skip_current=True)** - Add another selection containing the text of the current selection. If skip_current is true, the

current selection will be deselected. If nothing is currently selected, select the current word. Searches backwards if reverse is true.

Ctrl-Alt-Down: **goto-next-bookmark(current_file_only=True)** - Go to the next bookmark, or the first one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-Alt-E: **evaluate-sel-in-shell** - Evaluate the current selection from the editor within the Python Shell tool, optionally restarting the shell first. When whole_lines is set, the selection is rounded to whole lines before evaluation. When unspecified (set to None), the setting from the Shell's Option menu is used instead.

Ctrl-Alt-F: **search-sel-forward** - Search forward using current selection

Ctrl-Alt-F6: **debug-failed-tests** - Re-run all the previously failed tests in the debugger.

Ctrl-Alt-F7: **debug-last-tests** - Debug the last group of tests that were run.

Ctrl-Alt-G: **goto-bookmark** - Goto named bookmark

Ctrl-Alt-K: **show-bookmarks** - Show a list of all currently defined bookmarks

Ctrl-Alt-Left: **goto-previous-bookmark** - Go to the previous bookmark in the bookmark list, or the last one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-Alt-M: **set-bookmark** - Set a bookmark at current location on the editor. Mark is the project-wide textual name of the bookmark.

Ctrl-Alt-Right: **goto-next-bookmark** - Go to the next bookmark, or the first one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-Alt-T: **toggle-bookmark** - Set or remove a bookmark at current location on the editor. When set, the name of the bookmark is set to an auto-generated default.

Ctrl-Alt-Up: **goto-previous-bookmark(current_file_only=True)** - Go to the previous bookmark in the bookmark list, or the last one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-Alt-V: **evaluate-file-in-shell** - Run the contents of the editor within the Python Shell

Ctrl-Alt-period: **replace-string-regex** - Replace all occurrences of a string from the cursor position to end of file. The search string is treated as a regular expression.

Ctrl-Apostrophe: **enclose(start="", end="")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-B: **isearch-sel-forward** - Action varies according to focus: *Active Editor Commands*: Initiate incremental mini-search forward from the cursor position, using current selection as the search string. Set persist=False to do the search but end the interactive search session immediately.; *Document Viewer Commands*: Initiate incremental mini-search forward from the cursor position, using current selection as the search string. Set persist=False to do the search but end the interactive search session immediately.

Ctrl-BackSpace: **backward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word behind of the cursor ; *Toolbar Search Commands*: Delete word behind the cursor

Ctrl-Braceleft: **enclose(start="{", end="}")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-Bracketleft: **enclose(start="[", end="]")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-C: **copy** - Action varies according to focus: *Active Editor Commands*: Copy selected text ; *Document Viewer Commands*: Copy any selected text. ; *Exceptions Commands*: Copy the exception traceback to the clipboard ; *Search Manager Instance Commands*: Copy selected text ; *Toolbar Search Commands*: Cut selection

Ctrl-Comma: **next-window** - Switch to the next window alphabetically by title

Ctrl-D: **toolbar-search-focus** - Move focus to toolbar search entry.

Ctrl-Delete: **forward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word in front of the cursor ; *Toolbar Search Commands*: Delete word in front of the cursor

Ctrl-Down: **select-less** - Select less code; undoes the last select-more command

Ctrl-E: **brace-match** - Match brace at current cursor position, selecting all text between the two and hilighting the braces

Ctrl-End: **end-of-document** - Move cursor to end of document

Ctrl-F: **search** - Bring up the search manager in search mode.

Ctrl-F12: **command-by-name** - Execute given command by name, collecting any args as needed

Ctrl-F3: **search-sel-forward** - Search forward using current selection

Ctrl-F4: **close** - Close active document. Abandon any changes when ignore_changes is True. Close empty windows when close_window is true and quit if all document windows closed when can_quit is true.

Ctrl-F5: **debug-kill** - Stop debugging

Ctrl-F6: **step-over** - Step over current instruction

Ctrl-F8: **start-select-line** - Turn on auto-select mode line by line

Ctrl-F9: **break-clear-all** - Clear all breakpoints

Ctrl-G: **search-forward** - Search again using the search manager's current settings in forward direction

Ctrl-H: **replace** - Bring up the search manager in replace mode.

Ctrl-Home: **start-of-document** - Move cursor to start of document

Ctrl-I: **replace-and-search** - Replace current selection and search again.

Ctrl-Insert: **copy** - Action varies according to focus: *Active Editor Commands*: Copy selected text ; *Document Viewer Commands*: Copy any selected text. ; *Exceptions Commands*: Copy the exception traceback to the clipboard ; *Search Manager Instance Commands*: Copy selected text ; *Toolbar Search Commands*: Cut selection

Ctrl-J: **fill-paragraph** - Attempt to auto-justify the paragraph around the current start of selection

Ctrl-K: **open-from-keyboard** - Open a file from disk using keyboard-driven selection of the file

Ctrl-L: **goto-line** - Position cursor at start of given line number

Ctrl-Left: **backward-word** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move backward one word

Ctrl-Less: **enclose(start="<", end=">")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-M: **execute-kbd-macro** - Execute most recently recorded keyboard macro. If register is None then the user is asked to enter a letter a-z for the register where the macro is filed. Otherwise, register 'a' is used by default.

Ctrl-Minus: **zoom-out** - Action varies according to focus: *Document Viewer Commands*: Decrease documentation font size; *General Editor Commands*: Zoom out, increasing the text display size temporarily by one font size

Ctrl-N: **new-file** - Create a new file

Ctrl-Next: **forward-page** - Move cursor forward one page

Ctrl-O: **open-gui** - Open a file from disk, prompting with file selection dialog if necessary

Ctrl-P: **print-view** - Print active editor document

Ctrl-Page_Down: **next-document** - Move to the next document alphabetically in the list of documents open in the current window

Ctrl-Page_Up: **previous-document** - Move to the previous document alphabetically in the list of documents open in the current window

Ctrl-Parenleft: **enclose(start="(", end=")")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-Period: **comment-toggle** - Toggle commenting out of the selected lines. The style of commenting can be controlled with the style argument: 'indented' uses the default comment style indented at end of leading white space and 'block' uses a block comment in column zero. If not given, the style configured with the Editor / Block Comment Style preference is used.

Ctrl-Plus: **zoom-in** - Action varies according to focus: *Document Viewer Commands*: Increase documentation font size; *General Editor Commands*: Zoom in, increasing the text display size temporarily by one font size

Ctrl-Pointer_Button1: **goto-clicked-symbol-defn** - Goto the definition of the source symbol that was last clicked on, optionally showing the definition in another split if one is available and other_split is True.

Ctrl-Prior: **backward-page** - Move cursor backward one page

Ctrl-Q: **quit** - Quit the application.

Ctrl-Quotedbl: **enclose(start='"', end='"')** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-Quoteleft: **begin-visited-document-cycle(move_back=True, back_key="Ctrl-Quoteleft", forward_key="Ctrl-AsciiTilde")** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-R: **replace** - Bring up the search manager in replace mode.

Ctrl-Return: **new-line-after** - Place a new line after the current line

Ctrl-Right: **forward-word** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move forward one word

Ctrl-S: **save** - Save active document. Also close it if close is True.

Ctrl-Shift-B: **isearch-sel-backward** - Initiate incremental mini-search backward from the cursor position, using current selection as the search string. Set persist=False to do the search but end the interactive search session immediately.

Ctrl-Shift-C: **delete-line** - Delete the current line or lines when the selection spans multiple lines or given repeat is > 1

Ctrl-Shift-D: **selection-add-next-occurence** - Add another selection containing the text of the current selection. If skip_current is true, the current selection will be deselected. If nothing is currently selected, select the current word. Searches backwards if reverse is true.

Ctrl-Shift-Delete: **delete-lines**

Ctrl-Shift-Down: **move-line-down** - Move the current line or lines up down line, optionally indenting to match the new position

Ctrl-Shift-Down: **scroll-text-down** - Scroll text down a line w/o moving cursor's relative position on screen. Repeat is number of lines or if >0 and <1.0 then percent of screen. Set move_cursor to False to leave cursor in current position within the source, otherwise it is moved so the cursor remains on same screen line.

Ctrl-Shift-End: **end-of-document-extend** - Move cursor to end of document, adjusting the selection range to new position

Ctrl-Shift-F: **batch-search** - Search on current selection using the Search in Files tool. The look_in argument gets entered in the look in field if not None or ''. The current selection is put into the search field if it doesn't span multiple lines and either use_selection is true or there's nothing in the search field. The given search text is used instead, if provided

Ctrl-Shift-F3: **search-sel-backward** - Search backward using current selection

Ctrl-Shift-F5: **debug-stop** - Pause free-running execution at current program counter

Ctrl-Shift-F6: **debug-all-tests** - Debug all the tests in testing panel.

Ctrl-Shift-F7: **debug-current-tests** - Runs the current test or tests, if possible. The current tests are determined by the current position in the active view.

Ctrl-Shift-F9: Multiple commands (first available is executed):

- **break-disable-all** - Disable all breakpoints
- **break-enable-all** - Enable all breakpoints

Ctrl-Shift-G: **search-backward** - Search again using the search manager's current settings in backward direction

Ctrl-Shift-H: **batch-replace** - Display search and replace in files tool.

Ctrl-Shift-Home: **start-of-document-extend** - Move cursor to start of document, adjusting the selection range to new position

Ctrl-Shift-I: **add-current-file-to-project** - Add the frontmost currently open file to project

Ctrl-Shift-ISO_Left_Tab: **begin-visited-document-cycle(move_back=False)** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-Shift-L: **swap-lines** - Swap the line at start of current selection with the line that follows it, or the preceding line if previous is True.

Ctrl-Shift-Left: **backward-word-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one word, adjusting the selection range to new position. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move backward one word, extending the selection

Ctrl-Shift-Next: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Ctrl-Shift-O: **open-from-project** - Open document from the project via the Open From Project dialog. The given fragment is used as the initial fragment filter and if it is None, the selected text or the symbol under the cursor is used. If skip_if_unique is true, the file is opened without the dialog being displayed if only one filename matches the fragment.

Ctrl-Shift-Page_Down: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Ctrl-Shift-Page_Up: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Ctrl-Shift-Prior: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Ctrl-Shift-R: **batch-replace** - Display search and replace in files tool.

Ctrl-Shift-Right: **forward-word-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one word, adjusting the selection range to new position. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move forward one word, extending the selection

Ctrl-Shift-S: **save-as** - Save active document to a new file

Ctrl-Shift-T: **find-symbol** - Allow user to visit point of definition of a source symbol in the current editor context by typing a fragment of the name

Ctrl-Shift-Tab: **begin-visited-document-cycle(move_back=False)** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-Shift-U: **isearch-backward** - Action varies according to focus: *Active Editor Commands*: Initiate incremental mini-search backward from the cursor position, optionally entering the given search string ; *Document Viewer Commands*: Initiate incremental mini-search backward from the cursor position, optionally entering the given search string.

Ctrl-Shift-Up: **move-line-up** - Move the current line or lines up one line, optionally indenting to match the new position

Ctrl-Shift-Up: **scroll-text-up** - Scroll text up a line w/o moving cursor's relative position on screen. Repeat is number of lines or if >0 and <1.0 then percent of screen. Set move_cursor to False to leave cursor in current position within the source, otherwise it is moved so the cursor remains on same screen line.

Ctrl-Shift-V: **duplicate-line** - Duplicate the current line or lines. Places the duplicate on the line following the selection if pos is 'below' or before the selection if it is 'above'.

Ctrl-Shift-Y: **duplicate-line-above** - Duplicate the current line or lines above the selection.

Ctrl-Slash: **comment-out-region** - Comment out the selected region. The style of commenting can be controlled with the style argument: 'indented' uses the default comment style indented at end of leading white space and 'block' uses a block comment in column zero. If not given, the style configured with the Editor / Block Comment Style preference is used. Each call adds a level of commenting.

Ctrl-T: **forward-tab** - Action varies according to focus: *Active Editor Commands*: Place a tab character at the current cursor position ; *Search Manager Instance Commands*: Place a forward tab at the current cursor position in search or replace string

Ctrl-Tab: **begin-visited-document-cycle(move_back=True)** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-U: **isearch-forward** - Action varies according to focus: *Active Editor Commands*: Initiate incremental mini-search forward from the cursor position, optionally entering the given search string ; *Document Viewer Commands*: Initiate incremental mini-search forward from the cursor position, optionally entering the given search string.

Ctrl-Up: **select-more** - Select more code on either the current line or larger multi-line blocks.

Ctrl-V: **paste** - Action varies according to focus: *Active Editor Commands*: Paste text from clipboard ; *Search Manager Instance Commands*: Paste text from clipboard ; *Toolbar Search Commands*: Paste from clipboard

Ctrl-W: **close** - Close active document. Abandon any changes when ignore_changes is True. Close empty windows when close_window is true and quit if all document windows closed when can_quit is true.

Ctrl-X: **cut** - Action varies according to focus: *Active Editor Commands*: Cut selected text ; *Search Manager Instance Commands*: Cut selected text ; *Toolbar Search Commands*: Cut selection

Ctrl-Y: **redo** - Redo last action

Ctrl-Z: **undo** - Undo last action

Ctrl-]: **brace-match** - Match brace at current cursor position, selecting all text between the two and hilighting the braces

Ctrl-greater: **indent-region** - Indent the selected region one level of indentation. Set sel to None to use preference to determine selection behavior, or "never-select" to unselect after indent, "always-select" to always select after indent, or "retain-select" to retain current selection after indent.

Ctrl-less: **outdent-region** - Outdent the selected region one level of indentation. Set sel to None to use preference to determine selection behavior, or "never-select" to unselect after indent, "always-select" to always select after indent, or "retain-select" to retain current selection after indent.

Ctrl-parenleft: **start-kbd-macro** - Start definition of a keyboard macro. If register=None then the user is prompted to enter a letter a-z under which to file the macro. Otherwise, register 'a' is used by default.

Ctrl-parenright: **stop-kbd-macro** - Stop definition of a keyboard macro

Ctrl-question: **uncomment-out-region** - Uncomment out the selected region if commented out. If one_level is True then each call removes only one level of commenting.

Ctrl-space: **show-autocompleter** - Show the auto-completer for current cursor position

Delete: **forward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character in front of the cursor ; *Toolbar Search Commands*: Delete character in front of the cursor

Down: **next-line** - Move to screen next line, optionally repositioning character within line: 'same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

End: **end-of-line** - Action varies according to focus: *Active Editor Commands*: Move to end of current line; *Toolbar Search Commands*: Move to the end of the toolbar search entry

F1: Multiple commands (first available is executed):

- **show-horizontal-tools** - Show the horizontal tool area
- **minimize-horizontal-tools** - Minimize the horizontal tool area

F11: **frame-up** - Move up the current debug stack

F12: **frame-down** - Move down the current debug stack

F2: Multiple commands (first available is executed):

- **show-vertical-tools** - Show the vertical tool area
- **minimize-vertical-tools** - Minimize the vertical tool area

F3: **search-forward** - Search again using the search manager's current settings in forward direction

F4: **goto-selected-symbol-defn** - Goto the definition of the selected source symbol, optionally showing the definition in another split if one is available and other_split is True.

F5: **debug-continue** - Continue (or start) running, to next breakpoint

F6: **step-over-statement** - Step over current statement

F7: **step-into** - Step into current execution point, or start debugging at first line

F8: **step-out** - Step out of the current function or method

F9: Multiple commands (first available is executed):

- **break-set** - Set a new regular breakpoint on current line
- **break-clear** - Clear the breakpoint on the current line

Home: **beginning-of-line-text** - Move to end of the leading white space, if any, on the current line. If toggle is True, moves to the beginning of the line if already at the end of the leading white space (and vice versa).

ISO_Left_Tab: **backward-tab** - Outdent line at current position

Insert: **toggle-overtype** - Toggle status of overtyping mode

Left: **backward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one character ; *Toolbar Search Commands*: Move backward one character

Next: **forward-page** - Move cursor forward one page

Page_Down: **forward-page** - Move cursor forward one page

Page_Up: **backward-page** - Move cursor backward one page

Prior: **backward-page** - Move cursor backward one page

Return: **new-line** - Place a new line at the current cursor position

Right: **forward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one character ; *Toolbar Search Commands*: Move forward one character

Shift-Alt-A: **diff-merge-a-b**

Shift-Alt-B: **diff-merge-b-a**

Shift-Alt-Down: **next-line-extend-rect** - Move to next screen line, adjusting the rectangular selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Shift-Alt-Left: **backward-char-extend-rect** - Move cursor backward one character, adjusting the rectangular selection range to new position

Shift-Alt-N: **diff-next**

Shift-Alt-P: **diff-previous**

Shift-Alt-Right: **forward-char-extend-rect** - Move cursor forward one character, adjusting the rectangular selection range to new position

Shift-Alt-Up: **previous-line-extend-rect** - Move to previous screen line, adjusting the rectangular selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Shift-BackSpace: **backward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character behind the cursor, or the current selection if not empty. ; *Toolbar Search Commands*: Delete character behind the cursor

Shift-Ctrl-F8: **start-select-block** - Turn on auto-select block mode

Shift-Delete: **cut** - Action varies according to focus: *Active Editor Commands*: Cut selected text ; *Search Manager Instance Commands*: Cut selected text ; *Toolbar Search Commands*: Cut selection

Shift-Down: **next-line-extend** - Move to next screen line, adjusting the selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, 'fnb' for first non-blank char, or 'xcode' to simulate XCode style Shift-Alt line selection.

Shift-End: **end-of-line-extend** - Action varies according to focus: *Active Editor Commands*: Move to end of current line, adjusting the selection range to new position ; *Toolbar Search Commands*: Move to the end of the toolbar search entry, extending the selection

Shift-F1: **move-focus** - Move the keyboard focus forward within the Window to the next editable area

Shift-F11: **frame-show** - Show the position (thread and stack frame) where the debugger originally stopped

Shift-F2: Multiple commands (first available is executed):

- **enter-fullscreen** - Hide both the vertical and horizontal tool areas and toolbar, saving previous state so it can be restored later with exit_fullscreen
- **exit-fullscreen** - Restore previous non-fullscreen state of all tools and tool bar

Shift-F3: **search-backward** - Search again using the search manager's current settings in backward direction

Shift-F4: **find-points-of-use** - Find points of use for a symbol. The symbol defaults to the active selection.

Shift-F5: **debug-file** - Start debugging the current file (rather than the main entry point)

Shift-F6: **run-all-tests** - Runs all the tests in testing panel.

Shift-F7: **run-current-tests** - Runs the current test or tests, if possible. The current tests are determined by the current position in the active view. The tests are debugged when debug is True.

Shift-F8: **start-select-char** - Turn on auto-select mode character by character

Shift-F9: Multiple commands (first available is executed):

- **break-enable** - Enable the breakpoint on the current line
- **break-disable** - Disable the breakpoint on current line

Shift-Home: **beginning-of-line-text-extend** - Move to end of the leading white space, if any, on the current line, adjusting the selection range to the new position. If toggle is True, moves to the beginning of the line if already at the end of the leading white space (and vice versa).

Shift-Insert: **paste** - Action varies according to focus: *Active Editor Commands*: Paste text from clipboard ; *Search Manager Instance Commands*: Paste text from clipboard ; *Toolbar Search Commands*: Paste from clipboard

Shift-Left: **backward-char-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one character, adjusting the selection range to new position ; *Toolbar Search Commands*: Move backward one character, extending the selection

Shift-Next: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Shift-Page_Down: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Shift-Page_Up: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Shift-Prior: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Shift-Return: **new-line-before** - Place a new line before the current line

Shift-Right: **forward-char-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one character, adjusting the selection range to new position ; *Toolbar Search Commands*: Move forward one character, extending the selection

Shift-Tab: **backward-tab** - Outdent line at current position

Shift-Up: **previous-line-extend** - Move to previous screen line, adjusting the selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, 'fnb' for first non-blank char, or 'xcode' to simulate XCode style Shift-Alt line selection.

Tab: **tab-key** - Implement the tab key, the action of which is configurable by preference

Up: **previous-line** - Move to previous screen line, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Visual-Esc: **exit-visual-mode** - Exit visual mode and return back to default mode

21.2. Emacs Personality

This section documents all the default key bindings for the **Emacs** keyboard personality, set by the **Personality** preference.

Alt-!: **execute-process** - Execute the given command line in the OS Commands tool using default run directory and environment as defined in project properties, or the values set in an existing command with the same command line in the OS Commands tool.

Alt-0: **initiate-repeat-0** - Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke.

Alt-1: **fold-python-methods** - Fold up all Python methods, expand all classes, and leave other fold points alone

Alt-1: **initiate-repeat-1** - Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke.

Alt-2: **fold-python-classes** - Fold up all Python classes but leave other fold points alone

Alt-2: **initiate-repeat-2** - Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke.

Alt-3: **fold-python-classes-and-defs** - Fold up all Python classes, methods, and functions but leave other fold points alone

Alt-3: **initiate-repeat-3** - Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke.

Alt-4: **initiate-repeat-4** - Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke.

Alt-5: **initiate-repeat-5** - Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke.

Alt-6: **initiate-repeat-6** - Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke.

Alt-7: **initiate-repeat-7** - Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke.

Alt-8: **initiate-repeat-8** - Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke.

Alt-9: **initiate-repeat-9** - Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke.

Alt-@: **replace-string** - Replace all occurrences of a string from the cursor position to end of file.

Alt-B: **backward-word** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move backward one word

Alt-BackSpace: **backward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word behind of the cursor ; *Toolbar Search Commands*: Delete word behind the cursor

Alt-Backslash: **fold-toggle** - Toggle the current fold point

Alt-C: **title-case** - Change current selection to capitalize first letter of each word

Alt-D: **forward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word in front of the cursor ; *Toolbar Search Commands*: Delete word in front of the cursor

Alt-Delete: **backward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word behind of the cursor ; *Toolbar Search Commands*: Delete word behind the cursor

Alt-Down: **fold-expand-more-current** - Expand the current fold point one more level

Alt-End: **fold-expand-all** - Expand all fold points in the current file

Alt-F: **forward-word** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move forward one word

Alt-F11: **prev-points-of-use-match** - Display the previous match in the active points of use tool

Alt-F12: **next-points-of-use-match** - Display the next match in the active points of use tool

Alt-F3: **search** - Bring up the search manager in search mode.

Alt-F4: **close-window** - Close the current window and all documents and panels in it

Alt-F5: **run-to-cursor** - Run to current cursor position

Alt-F6: **run-failed-tests** - Re-run all the previously failed tests. The tests are debugged when debug is True.

Alt-F7: **run-last-tests** - Run again the last group of tests that were run. The tests are debugged when debug is True.

Alt-G: **goto-line** - Position cursor at start of given line number

Alt-Home: **fold-collapse-all** - Collapse all fold points in the current file

Alt-L: **goto-line** - Position cursor at start of given line number

Alt-Left: **visit-history-previous** - Move back in history to previous visited editor position

Alt-Page_Down: **fold-expand-all-current** - Expand the current fold point completely

Alt-Page_Up: **fold-collapse-all-current** - Collapse the current fold point completely

Alt-Period: **goto-selected-symbol-defn** - Goto the definition of the selected source symbol, optionally showing the definition in another split if one is available and other_split is True.

Alt-Q: **fill-paragraph** - Attempt to auto-justify the paragraph around the current start of selection

Alt-Return: **new-line** - Place a new line at the current cursor position

Alt-Right: **visit-history-next** - Move forward in history to next visited editor position

Alt-Slash: **fold-toggle** - Toggle the current fold point

Alt-Slash: **show-autocompleter** - Show the auto-completer for current cursor position

Alt-Tab: **show-autocompleter** - Show the auto-completer for current cursor position

Alt-Up: **fold-collapse-more-current** - Collapse the current fold point one more level

Alt-V: **backward-page** - Move cursor backward one page

Alt-W: **copy** - Action varies according to focus: *Active Editor Commands*: Copy selected text ; *Document Viewer Commands*: Copy any selected text. ; *Exceptions Commands*: Copy the exception traceback to the clipboard ; *Search Manager Instance Commands*: Copy selected text ; *Toolbar Search Commands*: Cut selection

Alt-X: **command-by-name** - Execute given command by name, collecting any args as needed

Alt-g: **goto-line** - Position cursor at start of given line number

Alt-greater: **end-of-document** - Move cursor to end of document

Alt-less: **start-of-document** - Move cursor to start of document

Alt-percent: **query-replace** - Initiate incremental mini-search query/replace from the cursor position.

Alt-percent: **query-replace** - Initiate incremental mini-search query/replace from the cursor position.

Alt-{: **previous-blank-line(threshold=1)** - Move to the previous blank line in the file, if any. If threshold>0 then a line is considered blank if it contains less than that many characters after leading and trailing whitespace are removed.

Alt-}: **next-blank-line(threshold=1)** - Move to the next blank line in the file, if any. If threshold>0 then a line is considered blank if it contains less than that many characters after leading and trailing whitespace are removed.

BackSpace: **backward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character behind the cursor, or the current selection if not empty. ; *Toolbar Search Commands*: Delete character behind the cursor

Ctrl-0: **replace** - Bring up the search manager in replace mode.

Ctrl-1: **activate-file-option-menu** - Activate the file menu for the editor.

Ctrl-2: **activate-symbol-option-menu-1** - Activate the 1st symbol menu for the editor.

Ctrl-3: **activate-symbol-option-menu-2** - Activate the 2nd symbol menu for the editor.

Ctrl-4: **activate-symbol-option-menu-3** - Activate the 3rd symbol menu for the editor.

Ctrl-5: **activate-symbol-option-menu-4** - Activate the 4th symbol menu for the editor.

Ctrl-6: **activate-symbol-option-menu-5** - Activate the 5th symbol menu for the editor.

Ctrl-9: **search** - Bring up the search manager in search mode.

Ctrl-=: **indent-to-match** - Indent the current line or selected region to match indentation of preceding non-blank line. Set toggle=True to indent instead of one level higher if already at the matching position.

Ctrl-@: **set-mark-command** - Set start of text marking for selection at current cursor position. Subsequently, all cursor move operations will automatically extend the text selection until stop-mark-command is issued. Unit defines what is selected: can be one of char, line, or block (rectangle).

Ctrl-A: **beginning-of-line** - Action varies according to focus: *Active Editor Commands*: Move to beginning of current line. When toggle is True, moves to the end of the leading white space if already at the beginning of the line (and vice versa).; *Toolbar Search Commands*: Move to the beginning of the toolbar search entry

Ctrl-Alt-@: **replace-string-regex** - Replace all occurrences of a string from the cursor position to end of file. The search string is treated as a regular expression.

Ctrl-Alt-B: **search-sel-backward** - Search backward using current selection

Ctrl-Alt-Down: **goto-next-bookmark(current_file_only=True)** - Go to the next bookmark, or the first one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-Alt-F: **search-sel-forward** - Search forward using current selection

Ctrl-Alt-F6: **debug-failed-tests** - Re-run all the previously failed tests in the debugger.

Ctrl-Alt-F7: **debug-last-tests** - Debug the last group of tests that were run.

Ctrl-Alt-Left: **goto-previous-bookmark** - Go to the previous bookmark in the bookmark list, or the last one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-Alt-R: **isearch-backward-regex** - Action varies according to focus: *Active Editor Commands*: Initiate incremental regular expression mini-search backward from the cursor position, optionally entering the given search string ; *Document Viewer Commands*: Initiate incremental regular expression mini-search backward from the cursor position, optionally entering the given search string.

Ctrl-Alt-Right: **goto-next-bookmark** - Go to the next bookmark, or the first one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-Alt-S: **isearch-forward-regex** - Action varies according to focus: *Active Editor Commands*: Initiate incremental regular expression mini-search forward from the cursor position, optionally entering the given search string ; *Document Viewer Commands*: Initiate incremental regular expression mini-search forward from the cursor position, optionally entering the given search string.

Ctrl-Alt-Up: **goto-previous-bookmark(current_file_only=True)** - Go to the previous bookmark in the bookmark list, or the last one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-Alt-percent: **query-replace-regex** - Initiate incremental mini-search query/replace from the cursor position. The search string is treated as a regular expression.

Ctrl-Apostrophe: **enclose(start="'", end="'")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-B: **backward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one character ; *Toolbar Search Commands*: Move backward one character

Ctrl-BackSpace: **backward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word behind of the cursor ; *Toolbar Search Commands*: Delete word behind the cursor

Ctrl-Braceleft: **enclose(start="{", end="}")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-Bracketleft: **enclose(start="[", end="]")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-C Bar: **evaluate-sel-in-shell** - Evaluate the current selection from the editor within the Python Shell tool, optionally restarting the shell first. When whole_lines is set, the selection is rounded to whole lines before evaluation. When unspecified (set to None), the setting from the Shell's Option menu is used instead.

Ctrl-C C: **comment-out-region** - Comment out the selected region. The style of commenting can be controlled with the style argument: 'indented' uses the default comment style indented at end of leading white space and 'block' uses a block comment in column zero. If not given, the style configured with the Editor / Block Comment Style preference is used. Each call adds a level of commenting.

Ctrl-C Ctrl-C: **debug-continue** - Continue (or start) running, to next breakpoint

Ctrl-C Ctrl-K: **debug-kill** - Stop debugging

Ctrl-C Ctrl-S: **debug-stop** - Pause free-running execution at current program counter

Ctrl-C M: **isearch-sel**

Ctrl-C R: **isearch-sel-backward** - Initiate incremental mini-search backward from the cursor position, using current selection as the search string. Set persist=False to do the search but end the interactive search session immediately.

Ctrl-C S: **isearch-sel-forward** - Action varies according to focus: *Active Editor Commands*: Initiate incremental mini-search forward from the cursor position, using current selection as the search string. Set persist=False to do the search but end the interactive search session immediately.; *Document Viewer Commands*: Initiate incremental mini-search forward from the cursor position, using current selection as the search string. Set persist=False to do the search but end the interactive search session immediately.

Ctrl-C U: **uncomment-out-region** - Uncomment out the selected region if commented out. If one_level is True then each call removes only one level of commenting.

Ctrl-C greater: **indent-region** - Indent the selected region one level of indentation. Set sel to None to use preference to determine selection behavior, or "never-select" to unselect after indent, "always-select" to always select after indent, or "retain-select" to retain current selection after indent.

Ctrl-C less: **outdent-region** - Outdent the selected region one level of indentation. Set sel to None to use preference to determine selection behavior, or "never-select" to unselect after indent, "always-select" to always select after indent, or "retain-select" to retain current selection after indent.

Ctrl-C numbersign: **comment-toggle** - Toggle commenting out of the selected lines. The style of commenting can be controlled with the style argument: 'indented' uses the default comment style indented at end of leading white space and 'block'

uses a block comment in column zero. If not given, the style configured with the Editor / Block Comment Style preference is used.

Ctrl-D: **forward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character in front of the cursor ; *Toolbar Search Commands*: Delete character in front of the cursor

Ctrl-Delete: **forward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word in front of the cursor ; *Toolbar Search Commands*: Delete word in front of the cursor

Ctrl-Down: **select-less** - Select less code; undoes the last select-more command

Ctrl-E: **end-of-line** - Action varies according to focus: *Active Editor Commands*: Move to end of current line; *Toolbar Search Commands*: Move to the end of the toolbar search entry

Ctrl-End: **end-of-document** - Move cursor to end of document

Ctrl-F: **forward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one character ; *Toolbar Search Commands*: Move forward one character

Ctrl-F12: **command-by-name** - Execute given command by name, collecting any args as needed

Ctrl-F3: **search-sel-forward** - Search forward using current selection

Ctrl-F4: **close** - Close active document. Abandon any changes when ignore_changes is True. Close empty windows when close_window is true and quit if all document windows closed when can_quit is true.

Ctrl-F5: **debug-kill** - Stop debugging

Ctrl-F6: **step-over** - Step over current instruction

Ctrl-F8: **start-select-line** - Turn on auto-select mode line by line

Ctrl-F9: **break-clear-all** - Clear all breakpoints

Ctrl-G: **stop-mark-command** - Stop text marking for selection at current cursor position, leaving the selection set as is. Subsequent cursor move operations will deselect the range and set selection to cursor position. Deselect immediately when deselect is True.

Ctrl-H: **backward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character behind the cursor, or the current selection if not empty. ; *Toolbar Search Commands*: Delete character behind the cursor

Ctrl-Home: **start-of-document** - Move cursor to start of document

Ctrl-Insert: **copy** - Action varies according to focus: *Active Editor Commands*: Copy selected text ; *Document Viewer Commands*: Copy any selected text. ; *Exceptions Commands*: Copy the exception traceback to the clipboard ; *Search*

Manager Instance Commands: Copy selected text ; *Toolbar Search Commands*: Cut selection

Ctrl-J: **fill-paragraph** - Attempt to auto-justify the paragraph around the current start of selection

Ctrl-K: **kill-line** - Kill rest of line from cursor to end of line, and place it into the clipboard with any other contiguously removed lines. End-of-line is removed only if there is nothing between the cursor and the end of the line.

Ctrl-L: **center-cursor** - Scroll so cursor is centered on display

Ctrl-Left: **backward-word** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move backward one word

Ctrl-Less: **enclose(start="<", end=">")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-M: **brace-match** - Match brace at current cursor position, selecting all text between the two and hilighting the braces

Ctrl-Minus: **zoom-out** - Action varies according to focus: *Document Viewer Commands*: Decrease documentation font size; *General Editor Commands*: Zoom out, increasing the text display size temporarily by one font size

Ctrl-N: **next-line** - Move to screen next line, optionally repositioning character within line: 'same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Ctrl-Next: **forward-page** - Move cursor forward one page

Ctrl-O: **next-window** - Switch to the next window alphabetically by title

Ctrl-O: **open-line** - Open the current line by inserting a newline after the caret

Ctrl-P: **previous-line** - Move to previous screen line, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Ctrl-Page_Down: **next-document** - Move to the next document alphabetically in the list of documents open in the current window

Ctrl-Page_Up: **previous-document** - Move to the previous document alphabetically in the list of documents open in the current window

Ctrl-Parenleft: **enclose(start="(", end=")")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-Period: **redo** - Redo last action

Ctrl-Plus: **zoom-in** - Action varies according to focus: *Document Viewer Commands*: Increase documentation font size; *General Editor Commands*: Zoom in, increasing the text display size temporarily by one font size

Ctrl-Pointer_Button1: **goto-clicked-symbol-defn** - Goto the definition of the source symbol that was last clicked on, optionally showing the definition in another split if one is available and other_split is True.

Ctrl-Prior: **backward-page** - Move cursor backward one page

Ctrl-Quotedbl: **enclose(start='"', end='")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-Quoteleft: **begin-visited-document-cycle(move_back=True, back_key="Ctrl-Quoteleft", forward_key="Ctrl-AsciiTilde")** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-R: **isearch-backward** - Action varies according to focus: *Active Editor Commands*: Initiate incremental mini-search backward from the cursor position, optionally entering the given search string ; *Document Viewer Commands*: Initiate incremental mini-search backward from the cursor position, optionally entering the given search string.

Ctrl-Return: **new-line-after** - Place a new line after the current line

Ctrl-Right: **forward-word** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move forward one word

Ctrl-Right: **forward-word(gravity="end")** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move forward one word

Ctrl-S: **isearch-forward** - Action varies according to focus: *Active Editor Commands*: Initiate incremental mini-search forward from the cursor position, optionally entering the given search string ; *Document Viewer Commands*: Initiate incremental mini-search forward from the cursor position, optionally entering the given search string.

Ctrl-Shift-Delete: **delete-lines**

Ctrl-Shift-Down: **scroll-text-down** - Scroll text down a line w/o moving cursor's relative position on screen. Repeat is number of lines or if >0 and <1.0 then

percent of screen. Set move_cursor to False to leave cursor in current position within the source, otherwise it is moved so the cursor remains on same screen line.

Ctrl-Shift-End: **end-of-document-extend** - Move cursor to end of document, adjusting the selection range to new position

Ctrl-Shift-F3: **search-sel-backward** - Search backward using current selection

Ctrl-Shift-F5: **debug-stop** - Pause free-running execution at current program counter

Ctrl-Shift-F6: **debug-all-tests** - Debug all the tests in testing panel.

Ctrl-Shift-F7: **debug-current-tests** - Runs the current test or tests, if possible. The current tests are determined by the current position in the active view.

Ctrl-Shift-F9: Multiple commands (first available is executed):

- **break-disable-all** - Disable all breakpoints
- **break-enable-all** - Enable all breakpoints

Ctrl-Shift-Home: **start-of-document-extend** - Move cursor to start of document, adjusting the selection range to new position

Ctrl-Shift-I: **add-current-file-to-project** - Add the frontmost currently open file to project

Ctrl-Shift-ISO_Left_Tab: **begin-visited-document-cycle(move_back=False)** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-Shift-Left: **backward-word-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one word, adjusting the selection range to new position. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move backward one word, extending the selection

Ctrl-Shift-Next: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Ctrl-Shift-Page_Down: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Ctrl-Shift-Page_Up: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Ctrl-Shift-Prior: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Ctrl-Shift-Right: **forward-word-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one word, adjusting the selection range to

new position. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move forward one word, extending the selection

Ctrl-Shift-Tab: **begin-visited-document-cycle(move_back=False)** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-Shift-Up: **scroll-text-up** - Scroll text up a line w/o moving cursor's relative position on screen. Repeat is number of lines or if >0 and <1.0 then percent of screen. Set move_cursor to False to leave cursor in current position within the source, otherwise it is moved so the cursor remains on same screen line.

Ctrl-Slash: **undo** - Undo last action

Ctrl-Space: **set-mark-command** - Set start of text marking for selection at current cursor position. Subsequently, all cursor move operations will automatically extend the text selection until stop-mark-command is issued. Unit defines what is selected: can be one of char, line, or block (rectangle).

Ctrl-T: **forward-tab** - Action varies according to focus: *Active Editor Commands*: Place a tab character at the current cursor position ; *Search Manager Instance Commands*: Place a forward tab at the current cursor position in search or replace string

Ctrl-Tab: **begin-visited-document-cycle(move_back=True)** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-U: **initiate-repeat** - Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke.

Ctrl-Up: **select-more** - Select more code on either the current line or larger multi-line blocks.

Ctrl-V: **forward-page** - Move cursor forward one page

Ctrl-W: **cut** - Action varies according to focus: *Active Editor Commands*: Cut selected text ; *Search Manager Instance Commands*: Cut selected text ; *Toolbar Search Commands*: Cut selection

Ctrl-X 1: **unsplit** - Unsplit all editors so there's only one. Action specifies how to choose the remaining displayed editor. One of:

```
current -- Show current editor
close   -- Close current editor before unsplitting
recent  -- Change to recent buffer before unsplitting
recent-or-close -- Change to recent buffer before closing
split, or close the current buffer if there is only
one split left.
```

NOTE: The parameters for this command are subject to change in the future.

Ctrl-X 2: **split-vertically** - Split current view vertically. Create new editor in new view when new==1.

Ctrl-X 3: **split-horizontally** - Split current view horizontally.

Ctrl-X 4 A: **add-change-log-entry** - Add a change log entry

Ctrl-X 5 0: **close-window** - Close the current window and all documents and panels in it

Ctrl-X 5 2: **new-document-window** - Create a new document window with same documents and panels as in the current document window (if any; otherwise empty with default panels)

Ctrl-X 5 3: **new-document-window** - Create a new document window with same documents and panels as in the current document window (if any; otherwise empty with default panels)

Ctrl-X 5 O: **next-window** - Switch to the next window alphabetically by title

Ctrl-X B: **switch-document** - Switches to named document. Name may either be the complete name or the last path component of a path name.

Ctrl-X Bracketleft: **start-of-document** - Move cursor to start of document

Ctrl-X Bracketright: **end-of-document** - Move cursor to end of document

Ctrl-X Ctrl-C: **quit** - Quit the application.

Ctrl-X Ctrl-F: **open-from-keyboard** - Open a file from disk using keyboard-driven selection of the file

Ctrl-X Ctrl-O: **open-from-project** - Open document from the project via the Open From Project dialog. The given fragment is used as the initial fragment filter and if it is None, the selected text or the symbol under the cursor is used. If skip_if_unique is true, the file is opened without the dialog being displayed if only one filename matches the fragment.

Ctrl-X Ctrl-S: **save** - Save active document. Also close it if close is True.

Ctrl-X Ctrl-T: **swap-lines(previous=True)** - Swap the line at start of current selection with the line that follows it, or the preceding line if previous is True.

Ctrl-X Ctrl-W: **write-file** - Write current file to a new location, optionally omitting all but the lines in the given range. The editor is changed to point to the new location when follow is True. If follow is 'untitled' then the editor is changed to point to the new location only if starting with an untitled buffer and saving the whole file. Note that the editor contents will be truncated to the given start/end lines when follow is True.

Ctrl-X Ctrl-X: **exchange-point-and-mark** - When currently marking text, this exchanges the current position and mark ends of the current selection

Ctrl-X D: **recent-document** - Switches to previous document most recently visited in the current window or window set if in one-window-per-editor windowing mode.

Ctrl-X E: **execute-kbd-macro** - Execute most recently recorded keyboard macro. If register is None then the user is asked to enter a letter a-z for the register where the macro is filed. Otherwise, register 'a' is used by default.

Ctrl-X G: **find-symbol** - Allow user to visit point of definition of a source symbol in the current editor context by typing a fragment of the name

Ctrl-X I: **insert-file** - Insert a file at current cursor position, prompting user for file selection

Ctrl-X K: **kill-buffer** - Close the current text file

Ctrl-X L C: **use-lexer-cpp** - Force syntax highlighting for C/C++ source

Ctrl-X L H: **use-lexer-html** - Force syntax highlighting for HTML

Ctrl-X L M: **use-lexer-makefile** - Force syntax highlighting for make files

Ctrl-X L N: **use-lexer-none** - Use no syntax highlighting

Ctrl-X L P: **use-lexer-python** - Force syntax highlighting for Python source

Ctrl-X L S: **use-lexer-sql** - Force syntax highlighting for SQL

Ctrl-X L X: **use-lexer-Xml**

Ctrl-X N: **next-document** - Move to the next document alphabetically in the list of documents open in the current window

Ctrl-X O: **move-editor-focus** - Move focus to next or previous editor split, optionally wrapping when the end is reached.

Ctrl-X P: **previous-document** - Move to the previous document alphabetically in the list of documents open in the current window

Ctrl-X R B: **goto-bookmark** - Goto named bookmark

Ctrl-X R M: **set-bookmark** - Set a bookmark at current location on the editor. Mark is the project-wide textual name of the bookmark.

Ctrl-X R Return: **show-bookmarks** - Show a list of all currently defined bookmarks

Ctrl-X R T: **toggle-bookmark** - Set or remove a bookmark at current location on the editor. When set, the name of the bookmark is set to an auto-generated default.

Ctrl-X Space: Multiple commands (first available is executed):

- **break-set** - Set a new regular breakpoint on current line
- **break-clear** - Clear the breakpoint on the current line

Ctrl-X U: **undo** - Undo last action

Ctrl-X parenleft: **start-kbd-macro** - Start definition of a keyboard macro. If register=None then the user is prompted to enter a letter a-z under which to file the macro. Otherwise, register 'a' is used by default.

Ctrl-X parenright: **stop-kbd-macro** - Stop definition of a keyboard macro

Ctrl-Y: Multiple commands (first available is executed):

- **yank-line** - Yank contents of kill buffer created with kill-line into the edit buffer
- **paste** - Action varies according to focus: *Active Editor Commands*: Paste text from clipboard ; *Search Manager Instance Commands*: Paste text from clipboard ; *Toolbar Search Commands*: Paste from clipboard

Ctrl-parenleft: **batch-search** - Search on current selection using the Search in Files tool. The look_in argument gets entered in the look in field if not None or ''. The current selection is put into the search field if it doesn't span multiple lines and either use_selection is true or there's nothing in the search field. The given search text is used instead, if provided

Ctrl-parenright: **batch-replace** - Display search and replace in files tool.

Ctrl-underscore: **undo** - Undo last action

Delete: **forward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character in front of the cursor ; *Toolbar Search Commands*: Delete character in front of the cursor

Down: **next-line** - Move to screen next line, optionally repositioning character within line: 'same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

End: **end-of-line** - Action varies according to focus: *Active Editor Commands*: Move to end of current line; *Toolbar Search Commands*: Move to the end of the toolbar search entry

Esc X: **command-by-name** - Execute given command by name, collecting any args as needed

F1: Multiple commands (first available is executed):

- **show-horizontal-tools** - Show the horizontal tool area
- **minimize-horizontal-tools** - Minimize the horizontal tool area

F11: **frame-up** - Move up the current debug stack

F12: **frame-down** - Move down the current debug stack

F2: Multiple commands (first available is executed):

- **show-vertical-tools** - Show the vertical tool area
- **minimize-vertical-tools** - Minimize the vertical tool area

F3: **search-forward** - Search again using the search manager's current settings in forward direction

F4: **goto-selected-symbol-defn** - Goto the definition of the selected source symbol, optionally showing the definition in another split if one is available and other_split is True.

F5: **debug-continue** - Continue (or start) running, to next breakpoint

F6: **step-over-statement** - Step over current statement

F7: **step-into** - Step into current execution point, or start debugging at first line

F8: **step-out** - Step out of the current function or method

F9: Multiple commands (first available is executed):

- **break-set** - Set a new regular breakpoint on current line
- **break-clear** - Clear the breakpoint on the current line

Home: **beginning-of-line** - Action varies according to focus: *Active Editor Commands*: Move to beginning of current line. When toggle is True, moves to the end of the leading white space if already at the beginning of the line (and vice versa).; *Toolbar Search Commands*: Move to the beginning of the toolbar search entry

Home: **beginning-of-line-text** - Move to end of the leading white space, if any, on the current line. If toggle is True, moves to the beginning of the line if already at the end of the leading white space (and vice versa).

ISO_Left_Tab: **backward-tab** - Outdent line at current position

Insert: **toggle-overtype** - Toggle status of overtyping mode

Left: **backward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one character ; *Toolbar Search Commands*: Move backward one character

Next: **forward-page** - Move cursor forward one page

Page_Down: **forward-page** - Move cursor forward one page

Page_Up: **backward-page** - Move cursor backward one page

Prior: **backward-page** - Move cursor backward one page

Return: **new-line** - Place a new line at the current cursor position

Right: **forward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one character ; *Toolbar Search Commands*: Move forward one character

Shift-Alt-A: **diff-merge-a-b**

Shift-Alt-B: **diff-merge-b-a**

Shift-Alt-Down: **next-line-extend-rect** - Move to next screen line, adjusting the rectangular selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Shift-Alt-Left: **backward-char-extend-rect** - Move cursor backward one character, adjusting the rectangular selection range to new position

Shift-Alt-N: **diff-next**

Shift-Alt-P: **diff-previous**

Shift-Alt-Right: **forward-char-extend-rect** - Move cursor forward one character, adjusting the rectangular selection range to new position

Shift-Alt-Up: **previous-line-extend-rect** - Move to previous screen line, adjusting the rectangular selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Shift-BackSpace: **backward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character behind the cursor, or the current selection if not empty. ; *Toolbar Search Commands*: Delete character behind the cursor

Shift-Ctrl-F8: **start-select-block** - Turn on auto-select block mode

Shift-Delete: **cut** - Action varies according to focus: *Active Editor Commands*: Cut selected text ; *Search Manager Instance Commands*: Cut selected text ; *Toolbar Search Commands*: Cut selection

Shift-Down: **next-line-extend** - Move to next screen line, adjusting the selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, 'fnb' for first non-blank char, or 'xcode' to simulate XCode style Shift-Alt line selection.

Shift-End: **end-of-line-extend** - Action varies according to focus: *Active Editor Commands*: Move to end of current line, adjusting the selection range to new position ; *Toolbar Search Commands*: Move to the end of the toolbar search entry, extending the selection

Shift-F1: **move-focus** - Move the keyboard focus forward within the Window to the next editable area

Shift-F11: **frame-show** - Show the position (thread and stack frame) where the debugger originally stopped

Shift-F2: Multiple commands (first available is executed):

- **enter-fullscreen** - Hide both the vertical and horizontal tool areas and toolbar, saving previous state so it can be restored later with exit_fullscreen
- **exit-fullscreen** - Restore previous non-fullscreen state of all tools and tool bar

Shift-F3: **search-backward** - Search again using the search manager's current settings in backward direction

Shift-F4: **find-points-of-use** - Find points of use for a symbol. The symbol defaults to the active selection.

Shift-F5: **debug-file** - Start debugging the current file (rather than the main entry point)

Shift-F6: **run-all-tests** - Runs all the tests in testing panel.

Shift-F7: **run-current-tests** - Runs the current test or tests, if possible. The current tests are determined by the current position in the active view. The tests are debugged when debug is True.

Shift-F8: **start-select-char** - Turn on auto-select mode character by character

Shift-F9: Multiple commands (first available is executed):

- **break-enable** - Enable the breakpoint on the current line
- **break-disable** - Disable the breakpoint on current line

Shift-Home: **beginning-of-line-extend** - Action varies according to focus: *Active Editor Commands*: Move to beginning of current line, adjusting the selection range to the new position. When toggle is True, moves to the end of the leading white space if already at the beginning of the line (and vice versa).; *Toolbar Search Commands*: Move to the beginning of the toolbar search entry, extending the selection

Shift-Home: **beginning-of-line-text-extend** - Move to end of the leading white space, if any, on the current line, adjusting the selection range to the new position. If toggle is True, moves to the beginning of the line if already at the end of the leading white space (and vice versa).

Shift-Insert: **paste** - Action varies according to focus: *Active Editor Commands*: Paste text from clipboard ; *Search Manager Instance Commands*: Paste text from clipboard ; *Toolbar Search Commands*: Paste from clipboard

Shift-Left: **backward-char-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one character, adjusting the selection range to new position ; *Toolbar Search Commands*: Move backward one character, extending the selection

Shift-Next: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Shift-Page_Down: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Shift-Page_Up: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Shift-Prior: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Shift-Return: **new-line-before** - Place a new line before the current line

Shift-Right: **forward-char-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one character, adjusting the selection range to new position ; *Toolbar Search Commands*: Move forward one character, extending the selection

Shift-Tab: **backward-tab** - Outdent line at current position

Shift-Up: **previous-line-extend** - Move to previous screen line, adjusting the selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, 'fnb' for first non-blank char, or 'xcode' to simulate XCode style Shift-Alt line selection.

Tab: **tab-key** - Implement the tab key, the action of which is configurable by preference

Up: **previous-line** - Move to previous screen line, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Visual-Esc: **exit-visual-mode** - Exit visual mode and return back to default mode

21.3. VI/VIM Personality

This section documents all the default key bindings for the **VI/VIM** keyboard personality, set by the **Personality** preference.

Alt-1: **fold-python-methods** - Fold up all Python methods, expand all classes, and leave other fold points alone

Alt-2: **fold-python-classes** - Fold up all Python classes but leave other fold points alone

Alt-3: **fold-python-classes-and-defs** - Fold up all Python classes, methods, and functions but leave other fold points alone

Alt-BackSpace: **backward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word behind of the cursor ; *Toolbar Search Commands*: Delete word behind the cursor

Alt-Delete: **backward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word behind of the cursor ; *Toolbar Search Commands*: Delete word behind the cursor

Alt-Down: **fold-expand-more-current** - Expand the current fold point one more level

Alt-End: **fold-expand-all** - Expand all fold points in the current file

Alt-F11: **prev-points-of-use-match** - Display the previous match in the active points of use tool

Alt-F12: **next-points-of-use-match** - Display the next match in the active points of use tool

Alt-F3: **search** - Bring up the search manager in search mode.

Alt-F4: **close-window** - Close the current window and all documents and panels in it

Alt-F5: **run-to-cursor** - Run to current cursor position

Alt-F6: **run-failed-tests** - Re-run all the previously failed tests. The tests are debugged when debug is True.

Alt-F7: **run-last-tests** - Run again the last group of tests that were run. The tests are debugged when debug is True.

Alt-Home: **fold-collapse-all** - Collapse all fold points in the current file

Alt-Left: **visit-history-previous** - Move back in history to previous visited editor position

Alt-Page_Down: **fold-expand-all-current** - Expand the current fold point completely

Alt-Page_Up: **fold-collapse-all-current** - Collapse the current fold point completely

Alt-Return: **new-line** - Place a new line at the current cursor position

Alt-Right: **visit-history-next** - Move forward in history to next visited editor position

Alt-Slash: **fold-toggle** - Toggle the current fold point

Alt-Up: **fold-collapse-more-current** - Collapse the current fold point one more level

BackSpace: **backward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character behind the cursor, or the current selection if not empty. ; *Toolbar Search Commands*: Delete character behind the cursor

Browse-!: **filter-next-move** - Filter the lines covered by the next cursor move command through an external command and replace the lines with the result

Browse-": **set-register** - Set the register to use for subsequent cut/copy/paste operations

Browse-#: **isearch-sel-backward(persist=0, whole_word=1)** - Initiate incremental mini-search backward from the cursor position, using current selection as the search string. Set persist=False to do the search but end the interactive search session immediately.

Browse-$: **end-of-line** - Action varies according to focus: *Active Editor Commands*: Move to end of current line; *Toolbar Search Commands*: Move to the end of the toolbar search entry

Browse-%: **goto-percent-line(cursor="fnb")** - Position cursor at start of line at given percent in file. This uses the previously entered numeric modifier or defaults to going to line one. The cursor can be positioned at 'start', 'end', or 'fnb' for first non-blank character, or in VI mode it will do brace matching operation to reflect how VI overrides this command.

Browse-&: **repeat-replace** - Repeat the last query replace or range replace operation on the current line. The first match is replaced without confirmation.

Browse-+: **next-line-in-file(cursor="fnb")** - Move to next line in file, repositioning character within line: 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Browse-,: **repeat-search-char(opposite=1)** - Repeat the last search_char operation, optionally in the opposite direction.

Browse-.: **repeat-command** - Repeat the last editor command

Browse-/: **isearch-forward-regex** - Action varies according to focus: *Active Editor Commands*: Initiate incremental regular expression mini-search forward from the cursor position, optionally entering the given search string ; *Document Viewer Commands*: Initiate incremental regular expression mini-search forward from the cursor position, optionally entering the given search string.

Browse-0: **beginning-of-line(toggle=0)** - Action varies according to focus: *Active Editor Commands*: Move to beginning of current line. When toggle is True, moves to the end of the leading white space if already at the beginning of the line (and vice versa).; *Toolbar Search Commands*: Move to the beginning of the toolbar search entry

Browse-1: **initiate-numeric-modifier(digit=1)** - VI style repeat/numeric modifier for following command

Browse-2: **initiate-numeric-modifier(digit=2)** - VI style repeat/numeric modifier for following command

Browse-3: **initiate-numeric-modifier(digit=3)** - VI style repeat/numeric modifier for following command

Browse-4: **initiate-numeric-modifier(digit=4)** - VI style repeat/numeric modifier for following command

Browse-5: **initiate-numeric-modifier(digit=5)** - VI style repeat/numeric modifier for following command

Browse-6: **initiate-numeric-modifier(digit=6)** - VI style repeat/numeric modifier for following command

Browse-7: **initiate-numeric-modifier(digit=7)** - VI style repeat/numeric modifier for following command

Browse-8: **initiate-numeric-modifier(digit=8)** - VI style repeat/numeric modifier for following command

Browse-9: **initiate-numeric-modifier(digit=9)** - VI style repeat/numeric modifier for following command

Browse-;: **repeat-search-char** - Repeat the last search_char operation, optionally in the opposite direction.

Browse-<: **outdent-next-move** - Outdent lines spanned by next cursor move

Browse-=: **indent-to-match-next-move** - Indent lines spanned by next cursor move to match, based on the preceding line

Browse->: **indent-next-move** - Indent lines spanned by next cursor move

Browse-?: **isearch-backward-regex** - Action varies according to focus: *Active Editor Commands*: Initiate incremental regular expression mini-search backward from the cursor position, optionally entering the given search string ; *Document Viewer Commands*: Initiate incremental regular expression mini-search backward from the cursor position, optionally entering the given search string.

Browse-@: **execute-kbd-macro(register=None)** - Execute most recently recorded keyboard macro. If register is None then the user is asked to enter a letter a-z for the register where the macro is filed. Otherwise, register 'a' is used by default.

Browse-A: **enter-insert-mode(pos="after")** - Enter editor insert mode

Browse-Apostrophe: **vi-goto-bookmark** - Goto bookmark using single character name defined by the next pressed key

Browse-BackSpace: **backward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one character ; *Toolbar Search Commands*: Move backward one character

Browse-Ctrl-B: **backward-page** - Move cursor backward one page

Browse-Ctrl-C: **vi-ctrl-c**

Browse-Ctrl-D: **scroll-text-down(repeat=0.5)** - Scroll text down a line w/o moving cursor's relative position on screen. Repeat is number of lines or if >0 and <1.0 then percent of screen. Set move_cursor to False to leave cursor in current position within the source, otherwise it is moved so the cursor remains on same screen line.

Browse-Ctrl-E: **scroll-text-down(move_cursor=False)** - Scroll text down a line w/o moving cursor's relative position on screen. Repeat is number of lines or if >0 and <1.0 then percent of screen. Set move_cursor to False to leave cursor in current position within the source, otherwise it is moved so the cursor remains on same screen line.

Browse-Ctrl-F: **forward-page** - Move cursor forward one page

Browse-Ctrl-I: **visit-history-next** - Move forward in history to next visited editor position

Browse-Ctrl-J: **next-line** - Move to screen next line, optionally repositioning character within line: 'same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Browse-Ctrl-M: **next-line-in-file(cursor="fnb")** - Move to next line in file, repositioning character within line: 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Browse-Ctrl-N: **next-line** - Move to screen next line, optionally repositioning character within line: 'same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Browse-Ctrl-O: **visit-history-previous** - Move back in history to previous visited editor position

Browse-Ctrl-P: **previous-line** - Move to previous screen line, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Browse-Ctrl-Q: **start-select-block** - Turn on auto-select block mode

Browse-Ctrl-R: **redo** - Redo last action

Browse-Ctrl-Shift-O: **open-from-project** - Open document from the project via the Open From Project dialog. The given fragment is used as the initial fragment filter and if it is None, the selected text or the symbol under the cursor is used. If skip_if_unique is true, the file is opened without the dialog being displayed if only one filename matches the fragment.

Browse-Ctrl-Shift-T: **find-symbol** - Allow user to visit point of definition of a source symbol in the current editor context by typing a fragment of the name

Browse-Ctrl-U: **scroll-text-up(repeat=0.5)** - Scroll text up a line w/o moving cursor's relative position on screen. Repeat is number of lines or if >0 and <1.0 then percent of screen. Set move_cursor to False to leave cursor in current position within the source, otherwise it is moved so the cursor remains on same screen line.

Browse-Ctrl-V: **vi-ctrl-v**

Browse-Ctrl-W Browse-+: **grow-split-vertically** - Increase height of this split

Browse-Ctrl-W Browse-Ctrl-W: **move-editor-focus** - Move focus to next or previous editor split, optionally wrapping when the end is reached.

Browse-Ctrl-W Browse-Ctrl-^: **vi-split-edit-alternate**

Browse-Ctrl-W Browse-Down: **move-editor-focus(wrap=False)** - Move focus to next or previous editor split, optionally wrapping when the end is reached.

Browse-Ctrl-W Browse-Minus: **shrink-split-vertically** - Decrease height of this split

Browse-Ctrl-W Browse-Up: **move-editor-focus(dir=-1, wrap=False)** - Move focus to next or previous editor split, optionally wrapping when the end is reached.

Browse-Ctrl-W Browse-W: **move-editor-focus(dir=-1)** - Move focus to next or previous editor split, optionally wrapping when the end is reached.

Browse-Ctrl-W Browse-b: **move-editor-focus-last** - Move focus to last editor split

Browse-Ctrl-W Browse-c: **unsplit(action="recent-or-close")** - Unsplit all editors so there's only one. Action specifies how to choose the remaining displayed editor. One of:

```
current -- Show current editor
close   -- Close current editor before unsplitting
recent  -- Change to recent buffer before unsplitting
recent-or-close -- Change to recent buffer before closing
split, or close the current buffer if there is only
one split left.
```

NOTE: The parameters for this command are subject to change in the future.

Browse-Ctrl-W Browse-j: **move-editor-focus(wrap=False)** - Move focus to next or previous editor split, optionally wrapping when the end is reached.

Browse-Ctrl-W Browse-k: **move-editor-focus(dir=-1, wrap=False)** - Move focus to next or previous editor split, optionally wrapping when the end is reached.

Browse-Ctrl-W Browse-n: **split-vertically(new=1)** - Split current view vertically. Create new editor in new view when new==1.

Browse-Ctrl-W Browse-o: **unsplit** - Unsplit all editors so there's only one. Action specifies how to choose the remaining displayed editor. One of:

```
current -- Show current editor
close   -- Close current editor before unsplitting
recent  -- Change to recent buffer before unsplitting
recent-or-close -- Change to recent buffer before closing
split, or close the current buffer if there is only
one split left.
```

NOTE: The parameters for this command are subject to change in the future.

Browse-Ctrl-W Browse-p: **move-editor-focus-previous** - Move focus to previous editor split

Browse-Ctrl-W Browse-q: Multiple commands (first available is executed):

- **unsplit(action="close")** - Unsplit all editors so there's only one. Action specifies how to choose the remaining displayed editor. One of:

  ```
  current -- Show current editor
  close   -- Close current editor before unsplitting
  recent  -- Change to recent buffer before unsplitting
  recent-or-close -- Change to recent buffer before closing
  split, or close the current buffer if there is only
  one split left.
  ```

  NOTE: The parameters for this command are subject to change in the future.
- **close(close_window=1)** - Close active document. Abandon any changes when ignore_changes is True. Close empty windows when close_window is true and quit if all document windows closed when can_quit is true.

Browse-Ctrl-W Browse-s: **split-vertically** - Split current view vertically. Create new editor in new view when new==1.

Browse-Ctrl-W Browse-t: **move-editor-focus-first** - Move focus to first editor split

Browse-Ctrl-W Browse-v: **split-horizontally** - Split current view horizontally.

Browse-Ctrl-X: **vi-ctrl-x**

Browse-Ctrl-Y: **scroll-text-up(move_cursor=False)** - Scroll text up a line w/o moving cursor's relative position on screen. Repeat is number of lines or if >0 and <1.0 then percent of screen. Set move_cursor to False to leave cursor in current position within the source, otherwise it is moved so the cursor remains on same screen line.

Browse-Ctrl-^: **nth-document** - Move to the nth document alphabetically in the list of documents open in the current window

Browse-Ctrl-h: **backward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one character ; *Toolbar Search Commands*: Move backward one character

Browse-Esc: **clear-move-command** - Clear any pending move command action, as for VI mode

Browse-F: **search-char(dir=1, single_line=1)** - Search for the given character. Searches to right if dir > 0 and to left if dir < 0. Optionally place cursor pos characters to left or right of the target (e.g., use -1 to place one to left). If repeat > 1, the Nth match is found. Set single_line=1 to search only within the current line.

Browse-G Browse-Shift-I: **enter-insert-mode(pos="sol")** - Enter editor insert mode

Browse-Grave: **vi-goto-bookmark** - Goto bookmark using single character name defined by the next pressed key

Browse-I: **enter-insert-mode(pos="before")** - Enter editor insert mode

Browse-Insert: **enter-insert-mode(pos="before")** - Enter editor insert mode

Browse-Minus: **previous-line-in-file(cursor="fnb")** - Move to previous line in file, repositioning character within line: 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Browse-O: **enter-insert-mode(pos="new-below")** - Enter editor insert mode

Browse-Return: **next-line(cursor="start")** - Move to screen next line, optionally repositioning character within line: 'same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Browse-Shift-A: **enter-insert-mode(pos="eol")** - Enter editor insert mode

Browse-Shift-B: **backward-word(delimiters=" tnr")** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move backward one word

Browse-Shift-C: **delete-to-end-of-line-insert** - Delete everything between the cursor and end of line and enter insert move (when working in a modal editor key binding)

Browse-Shift-D: **delete-to-end-of-line(post_offset=-1)** - Delete everything between the cursor and end of line

Browse-Shift-E: **forward-word(delimiters=" tnr", gravity="endm1")** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one

word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move forward one word

Browse-Shift-F: **search-char(dir=-1, single_line=1)** - Search for the given character. Searches to right if dir > 0 and to left if dir < 0. Optionally place cursor pos characters to left or right of the target (e.g., use -1 to place one to left). If repeat > 1, the Nth match is found. Set single_line=1 to search only within the current line.

Browse-Shift-G: **goto-nth-line-default-end(cursor="fnb")** - Same as goto_nth_line but defaults to end of file if no lineno is given

Browse-Shift-H: **cursor-move-to-top** - Move cursor to top of display (without scrolling), optionally at an offset of given number of lines below top

Browse-Shift-I: **enter-insert-mode(pos="fnb")** - Enter editor insert mode

Browse-Shift-J: **join-lines** - Join together specified number of lines after current line (replace newlines with the given delimiter (single space by default)

Browse-Shift-L: **cursor-move-to-bottom** - Move cursor to bottom of display (without scrolling), optionally at an offset of given number of lines before bottom

Browse-Shift-M: **cursor-move-to-center** - Move cursor to center of display (without scrolling)

Browse-Shift-N: **isearch-repeat(reverse=1)** - Repeat the most recent isearch, using same string and regex/text. Reverse direction when reverse is True.

Browse-Shift-O: **enter-insert-mode(pos="new-above")** - Enter editor insert mode

Browse-Shift-P: **paste-register(pos=-1)** - Paste text from register as before or after the current position. If the register contains only lines, then the lines are pasted before or after current line (rather than at cursor). If the register contains fragments of lines, the text is pasted over the current selection or either before or after the cursor. Set pos = 1 to paste after, or -1 to paste before. Set indent=1 to indent the pasted text to match current line. Set cursor=-1 to place cursor before lines or cursor=1 to place it after lines after paste completes.

Browse-Shift-R: **enter-replace-mode** - Enter editor replace mode

Browse-Shift-S: **delete-line-insert** - Delete the current line or lines when the selection spans multiple lines or given repeat is > 1. Enters insert mode (when working with modal key bindings).

Browse-Shift-T: **search-char(dir=-1, pos=1, single_line=1)** - Search for the given character. Searches to right if dir > 0 and to left if dir < 0. Optionally place cursor pos characters to left or right of the target (e.g., use -1 to place one to left).

If repeat > 1, the Nth match is found. Set single_line=1 to search only within the current line.

Browse-Shift-V: **start-select-line** - Turn on auto-select mode line by line

Browse-Shift-W: **forward-word(delimiters=" tnr")** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move forward one word

Browse-Shift-Y: **move-to-register(unit="line")** - Cut or copy a specified number of characters or lines, or the current selection. Set cut=1 to remove the range of text from the editor after moving to register (otherwise it is just copied). Unit should be one of 'char' or 'line' or 'sel' for current selection.

Browse-Shift-Z Browse-Shift-Q: **close(ignore_changes=1, close_window=1)** - Close active document. Abandon any changes when ignore_changes is True. Close empty windows when close_window is true and quit if all document windows closed when can_quit is true.

Browse-Shift-Z Browse-Shift-Z: **write-file-and-close(filename=None)** - Write current document to given location and close it. Saves to current file name if the given filename is None.

Browse-Shift-x: **backward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character behind the cursor, or the current selection if not empty. ; *Toolbar Search Commands*: Delete character behind the cursor

Browse-Space: **forward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one character ; *Toolbar Search Commands*: Move forward one character

Browse-T: **search-char(dir=1, pos=1, single_line=1)** - Search for the given character. Searches to right if dir > 0 and to left if dir < 0. Optionally place cursor pos characters to left or right of the target (e.g., use -1 to place one to left). If repeat > 1, the Nth match is found. Set single_line=1 to search only within the current line.

Browse-Underscore: **beginning-of-line-text** - Move to end of the leading white space, if any, on the current line. If toggle is True, moves to the beginning of the line if already at the end of the leading white space (and vice versa).

Browse-[Browse-p: **paste-register(pos=-1, indent=1)** - Paste text from register as before or after the current position. If the register contains only lines, then the lines are pasted before or after current line (rather than at cursor). If the register contains fragments of lines, the text is pasted over the current selection or either before or after the cursor. Set pos = 1 to paste after, or -1 to paste before. Set

indent=1 to indent the pasted text to match current line. Set cursor=-1 to place cursor before lines or cursor=1 to place it after lines after paste completes.

Browse-*: **isearch-sel-forward(persist=0, whole_word=1)** - Action varies according to focus: *Active Editor Commands*: Initiate incremental mini-search forward from the cursor position, using current selection as the search string. Set persist=False to do the search but end the interactive search session immediately.; *Document Viewer Commands*: Initiate incremental mini-search forward from the cursor position, using current selection as the search string. Set persist=False to do the search but end the interactive search session immediately.

Browse-] Browse-p: **paste-register(indent=1)** - Paste text from register as before or after the current position. If the register contains only lines, then the lines are pasted before or after current line (rather than at cursor). If the register contains fragments of lines, the text is pasted over the current selection or either before or after the cursor. Set pos = 1 to paste after, or -1 to paste before. Set indent=1 to indent the pasted text to match current line. Set cursor=-1 to place cursor before lines or cursor=1 to place it after lines after paste completes.

Browse-^: **beginning-of-line-text(toggle=0)** - Move to end of the leading white space, if any, on the current line. If toggle is True, moves to the beginning of the line if already at the end of the leading white space (and vice versa).

Browse-b: **backward-word** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move backward one word

Browse-c: **delete-next-move-insert** - Delete the text covered by the next cursor move command and then enter insert mode (when working in a modal editor key binding)

Browse-colon: **vi-command-by-name** - Execute a VI command (implements ":" commands from VI)

Browse-d: **delete-next-move** - Delete the text covered by the next cursor move command.

Browse-e: **forward-word(gravity="endm1")** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move forward one word

Browse-g Browse-$: **end-of-screen-line** - Move to end of current wrapped line

Browse-g Browse-0: **beginning-of-screen-line** - Move to beginning of current wrapped line

Browse-g Browse-Shift-D: **goto-selected-symbol-defn** - Goto the definition of the selected source symbol, optionally showing the definition in another split if one is available and other_split is True.

Browse-g Browse-Shift-E: **backward-word(delimiters=" tnr", gravity="endm1")** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move backward one word

Browse-g Browse-Shift-J: **join-lines(delim="")** - Join together specified number of lines after current line (replace newlines with the given delimiter (single space by default)

Browse-g Browse-Shift-P: **paste-register(pos=-1, cursor=1)** - Paste text from register as before or after the current position. If the register contains only lines, then the lines are pasted before or after current line (rather than at cursor). If the register contains fragments of lines, the text is pasted over the current selection or either before or after the cursor. Set pos = 1 to paste after, or -1 to paste before. Set indent=1 to indent the pasted text to match current line. Set cursor=-1 to place cursor before lines or cursor=1 to place it after lines after paste completes.

Browse-g Browse-Shift-T: **previous-document** - Move to the previous document alphabetically in the list of documents open in the current window

Browse-g Browse-Shift-U: **case-upper-next-move** - Change case of text spanned by next cursor movement to upper case

Browse-g Browse-T: **next-document** - Move to the next document alphabetically in the list of documents open in the current window

Browse-g Browse-^: **beginning-of-screen-line-text** - Move to first non-blank character at beginning of current wrapped line

Browse-g Browse-d: **goto-selected-symbol-defn** - Goto the definition of the selected source symbol, optionally showing the definition in another split if one is available and other_split is True.

Browse-g Browse-e: **backward-word(gravity="endm1")** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move backward one word

Browse-g Browse-g: **goto-nth-line(cursor="fnb")** - Position cursor at start of given line number (1=first, -1 = last). This differs from goto-line in that it never prompts for a line number but instead uses the previously entered numeric modifier

or defaults to going to line one. The cursor can be positioned at 'start', 'end', or 'fnb' for first non-blank character.

Browse-g Browse-j: **next-line** - Move to screen next line, optionally repositioning character within line: 'same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Browse-g Browse-k: **previous-line** - Move to previous screen line, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Browse-g Browse-m: **middle-of-screen-line** - Move to middle of current wrapped line

Browse-g Browse-p: **paste-register(cursor=1)** - Paste text from register as before or after the current position. If the register contains only lines, then the lines are pasted before or after current line (rather than at cursor). If the register contains fragments of lines, the text is pasted over the current selection or either before or after the cursor. Set pos = 1 to paste after, or -1 to paste before. Set indent=1 to indent the pasted text to match current line. Set cursor=-1 to place cursor before lines or cursor=1 to place it after lines after paste completes.

Browse-g Browse-q Browse-q: **fill-paragraph** - Attempt to auto-justify the paragraph around the current start of selection

Browse-g Browse-r: **replace-char(line_mode="extend")** - Replace num characters with given character. Set line_mode to multiline to allow replacing across lines, extend to replace on current line and then extend the line length, and restrict to replace only if enough characters exist on current line after cursor position.

Browse-g Browse-u: **case-lower-next-move** - Change case of text spanned by next cursor movement to lower case

Browse-g Browse-v: **previous-select** - Turn on auto-select using previous mode and selection

Browse-g Browse-~: **case-swap-next-move** - Change case of text spanned by next cursor movement so each letter is the opposite of its current case

Browse-h: **backward-char(wrap=0)** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one character ; *Toolbar Search Commands*: Move backward one character

Browse-j: **next-line** - Move to screen next line, optionally repositioning character within line: 'same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Browse-k: **previous-line** - Move to previous screen line, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Browse-l: **forward-char(wrap=0)** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one character ; *Toolbar Search Commands*: Move forward one character

Browse-m: **vi-set-bookmark** - Set a bookmark at current location on the editor using the next key press as the name of the bookmark.

Browse-n: **isearch-repeat** - Repeat the most recent isearch, using same string and regex/text. Reverse direction when reverse is True.

Browse-p: **paste-register** - Paste text from register as before or after the current position. If the register contains only lines, then the lines are pasted before or after current line (rather than at cursor). If the register contains fragments of lines, the text is pasted over the current selection or either before or after the cursor. Set pos = 1 to paste after, or -1 to paste before. Set indent=1 to indent the pasted text to match current line. Set cursor=-1 to place cursor before lines or cursor=1 to place it after lines after paste completes.

Browse-q: Multiple commands (first available is executed):

- **start-kbd-macro(register=None)** - Start definition of a keyboard macro. If register=None then the user is prompted to enter a letter a-z under which to file the macro. Otherwise, register 'a' is used by default.
- **stop-kbd-macro** - Stop definition of a keyboard macro

Browse-r: **replace-char(line_mode="restrict")** - Replace num characters with given character. Set line_mode to multiline to allow replacing across lines, extend to replace on current line and then extend the line length, and restrict to replace only if enough characters exist on current line after cursor position.

Browse-s: **forward-delete-char-insert** - Delete one char in front of the cursor and enter insert mode (when working in modal key bindings)

Browse-u: **undo** - Undo last action

Browse-v: **start-select-char** - Turn on auto-select mode character by character

Browse-w: **forward-word** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move forward one word

Browse-x: **forward-delete-char-within-line** - Delete one character in front of the cursor unless at end of line, in which case delete backward. Do nothing if the line is empty. This is VI style 'x' in browser mode.

Browse-y: **move-to-register-next-move** - Move the text spanned by the next cursor motion to a register

Browse-z Browse-.: **center-cursor** - Scroll so cursor is centered on display

Browse-z Browse-Minus: **cursor-to-bottom** - Scroll so cursor is centered at bottom of display

Browse-z Browse-Plus: **cursor-to-top** - Scroll so cursor is centered at top of display

Browse-z Browse-Return: **cursor-to-top** - Scroll so cursor is centered at top of display

Browse-z Browse-Shift-H: **scroll-text-right(repeat=0.5)** - Scroll text right a column w/o moving cursor's relative position on screen. Repeat is number of columns or if >0 and <1.0 then percent of screen.

Browse-z Browse-Shift-L: **scroll-text-left(repeat=0.5)** - Scroll text left a column w/o moving cursor's relative position on screen. Repeat is number of columns or if >0 and <1.0 then percent of screen.

Browse-z Browse-Shift-M: **fold-collapse-all** - Collapse all fold points in the current file

Browse-z Browse-Shift-O: **fold-expand-all-current** - Expand the current fold point completely

Browse-z Browse-Shift-R: **fold-expand-all** - Expand all fold points in the current file

Browse-z Browse-b: **cursor-to-bottom** - Scroll so cursor is centered at bottom of display

Browse-z Browse-c: **fold-collapse-current** - Collapse the current fold point

Browse-z Browse-h: **scroll-text-right** - Scroll text right a column w/o moving cursor's relative position on screen. Repeat is number of columns or if >0 and <1.0 then percent of screen.

Browse-z Browse-l: **scroll-text-left** - Scroll text left a column w/o moving cursor's relative position on screen. Repeat is number of columns or if >0 and <1.0 then percent of screen.

Browse-z Browse-m: **vi-fold-less** - Approximation of zm key binding in vim

Browse-z Browse-o: **fold-expand-current** - Expand the current fold point

Browse-z Browse-r: **vi-fold-more** - Approximation of zr key binding in vim

Browse-z Browse-t: **cursor-to-top** - Scroll so cursor is centered at top of display

Browse-z Browse-z: **center-cursor** - Scroll so cursor is centered on display

Browse-{: **backward-paragraph** - Move cursor backward one paragraph (to next all-whitespace line).

Browse-|: **goto-column** - Move cursor to given column

Browse-}: **forward-paragraph** - Move cursor forward one paragraph (to next all-whitespace line).

Browse-~: **case-swap** - Change case of the current selection, or character ahead of the cursor if there is no selection, so each letter is the opposite of its current case

Ctrl-1: **activate-file-option-menu** - Activate the file menu for the editor.

Ctrl-2: **activate-symbol-option-menu-1** - Activate the 1st symbol menu for the editor.

Ctrl-3: **activate-symbol-option-menu-2** - Activate the 2nd symbol menu for the editor.

Ctrl-4: **activate-symbol-option-menu-3** - Activate the 3rd symbol menu for the editor.

Ctrl-5: **activate-symbol-option-menu-4** - Activate the 4th symbol menu for the editor.

Ctrl-6: **activate-symbol-option-menu-5** - Activate the 5th symbol menu for the editor.

Ctrl-=: **indent-to-match** - Indent the current line or selected region to match indentation of preceding non-blank line. Set toggle=True to indent instead of one level higher if already at the matching position.

Ctrl-Alt-Down: **goto-next-bookmark(current_file_only=True)** - Go to the next bookmark, or the first one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-Alt-F6: **debug-failed-tests** - Re-run all the previously failed tests in the debugger.

Ctrl-Alt-F7: **debug-last-tests** - Debug the last group of tests that were run.

Ctrl-Alt-Left: **goto-previous-bookmark** - Go to the previous bookmark in the bookmark list, or the last one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-Alt-Right: **goto-next-bookmark** - Go to the next bookmark, or the first one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-Alt-Up: **goto-previous-bookmark(current_file_only=True)** - Go to the previous bookmark in the bookmark list, or the last one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-Apostrophe: **enclose(start="", end="")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-BackSpace: **backward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word behind of the cursor ; *Toolbar Search Commands*: Delete word behind the cursor

Ctrl-Braceleft: **enclose(start="{", end="}")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-Bracketleft: **enclose(start="[", end="]")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-Delete: **forward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word in front of the cursor ; *Toolbar Search Commands*: Delete word in front of the cursor

Ctrl-Down: **select-less** - Select less code; undoes the last select-more command

Ctrl-End: **end-of-document** - Move cursor to end of document

Ctrl-F12: **command-by-name** - Execute given command by name, collecting any args as needed

Ctrl-F3: **search-sel-forward** - Search forward using current selection

Ctrl-F4: **close** - Close active document. Abandon any changes when ignore_changes is True. Close empty windows when close_window is true and quit if all document windows closed when can_quit is true.

Ctrl-F5: **debug-kill** - Stop debugging

Ctrl-F6: **step-over** - Step over current instruction

Ctrl-F8: **start-select-line** - Turn on auto-select mode line by line

Ctrl-F9: **break-clear-all** - Clear all breakpoints

Ctrl-Home: **start-of-document** - Move cursor to start of document

Ctrl-Insert: **copy** - Action varies according to focus: *Active Editor Commands*: Copy selected text ; *Document Viewer Commands*: Copy any selected text. ; *Exceptions Commands*: Copy the exception traceback to the clipboard ; *Search Manager Instance Commands*: Copy selected text ; *Toolbar Search Commands*: Cut selection

Ctrl-Left: **backward-word** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move backward one word

Ctrl-Less: **enclose(start="<", end=">")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-Minus: **zoom-out** - Action varies according to focus: *Document Viewer Commands*: Decrease documentation font size; *General Editor Commands*: Zoom out, increasing the text display size temporarily by one font size

Ctrl-Next: **forward-page** - Move cursor forward one page

Ctrl-Page_Down: **next-document** - Move to the next document alphabetically in the list of documents open in the current window

Ctrl-Page_Up: **previous-document** - Move to the previous document alphabetically in the list of documents open in the current window

Ctrl-Parenleft: **enclose(start="(", end=")")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-Plus: **zoom-in** - Action varies according to focus: *Document Viewer Commands*: Increase documentation font size; *General Editor Commands*: Zoom in, increasing the text display size temporarily by one font size

Ctrl-Pointer_Button1: **goto-clicked-symbol-defn** - Goto the definition of the source symbol that was last clicked on, optionally showing the definition in another split if one is available and other_split is True.

Ctrl-Prior: **backward-page** - Move cursor backward one page

Ctrl-Quotedbl: **enclose(start='"', end='"')** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-Quoteleft: **begin-visited-document-cycle(move_back=True, back_key="Ctrl-Quoteleft", forward_key="Ctrl-AsciiTilde")** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-Return: **new-line-after** - Place a new line after the current line

Ctrl-Right: **forward-word** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move forward one word

Ctrl-S: **save** - Save active document. Also close it if close is True.

Ctrl-Shift-Delete: **delete-lines**

Ctrl-Shift-Down: **scroll-text-down** - Scroll text down a line w/o moving cursor's relative position on screen. Repeat is number of lines or if >0 and <1.0 then percent of screen. Set move_cursor to False to leave cursor in current position within the source, otherwise it is moved so the cursor remains on same screen line.

Ctrl-Shift-End: **end-of-document-extend** - Move cursor to end of document, adjusting the selection range to new position

Ctrl-Shift-F: **batch-search** - Search on current selection using the Search in Files tool. The look_in argument gets entered in the look in field if not None or ''. The current selection is put into the search field if it doesn't span multiple lines and either use_selection is true or there's nothing in the search field. The given search text is used instead, if provided

Ctrl-Shift-F3: **search-sel-backward** - Search backward using current selection

Ctrl-Shift-F5: **debug-stop** - Pause free-running execution at current program counter

Ctrl-Shift-F6: **debug-all-tests** - Debug all the tests in testing panel.

Ctrl-Shift-F7: **debug-current-tests** - Runs the current test or tests, if possible. The current tests are determined by the current position in the active view.

Ctrl-Shift-F9: Multiple commands (first available is executed):

- **break-disable-all** - Disable all breakpoints
- **break-enable-all** - Enable all breakpoints

Ctrl-Shift-G: **batch-replace** - Display search and replace in files tool.

Ctrl-Shift-Home: **start-of-document-extend** - Move cursor to start of document, adjusting the selection range to new position

Ctrl-Shift-I: **add-current-file-to-project** - Add the frontmost currently open file to project

Ctrl-Shift-ISO_Left_Tab: **begin-visited-document-cycle(move_back=False)** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-Shift-Left: **backward-word-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one word, adjusting the selection range to new position. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move backward one word, extending the selection

Ctrl-Shift-Next: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Ctrl-Shift-Page_Down: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Ctrl-Shift-Page_Up: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Ctrl-Shift-Prior: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Ctrl-Shift-Right: **forward-word-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one word, adjusting the selection range to new position. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move forward one word, extending the selection

Ctrl-Shift-Tab: **begin-visited-document-cycle(move_back=False)** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-Shift-Up: **scroll-text-up** - Scroll text up a line w/o moving cursor's relative position on screen. Repeat is number of lines or if >0 and <1.0 then percent of screen. Set move_cursor to False to leave cursor in current position within the source, otherwise it is moved so the cursor remains on same screen line.

Ctrl-T: **forward-tab** - Action varies according to focus: *Active Editor Commands*: Place a tab character at the current cursor position ; *Search Manager Instance Commands*: Place a forward tab at the current cursor position in search or replace string

Ctrl-Tab: **begin-visited-document-cycle(move_back=True)** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-Up: **select-more** - Select more code on either the current line or larger multi-line blocks.

Delete: **forward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character in front of the cursor ; *Toolbar Search Commands*: Delete character in front of the cursor

Down: **next-line** - Move to screen next line, optionally repositioning character within line: 'same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

End: **end-of-line** - Action varies according to focus: *Active Editor Commands*: Move to end of current line; *Toolbar Search Commands*: Move to the end of the toolbar search entry

F1: Multiple commands (first available is executed):

- **show-horizontal-tools** - Show the horizontal tool area
- **minimize-horizontal-tools** - Minimize the horizontal tool area

F11: **frame-up** - Move up the current debug stack

F12: **frame-down** - Move down the current debug stack

F2: Multiple commands (first available is executed):

- **show-vertical-tools** - Show the vertical tool area
- **minimize-vertical-tools** - Minimize the vertical tool area

F3: **search-forward** - Search again using the search manager's current settings in forward direction

F4: **goto-selected-symbol-defn** - Goto the definition of the selected source symbol, optionally showing the definition in another split if one is available and other_split is True.

F5: **debug-continue** - Continue (or start) running, to next breakpoint

F6: **step-over-statement** - Step over current statement

F7: **step-into** - Step into current execution point, or start debugging at first line

F8: **step-out** - Step out of the current function or method

F9: Multiple commands (first available is executed):

- **break-set** - Set a new regular breakpoint on current line
- **break-clear** - Clear the breakpoint on the current line

Home: **beginning-of-line-text** - Move to end of the leading white space, if any, on the current line. If toggle is True, moves to the beginning of the line if already at the end of the leading white space (and vice versa).

ISO_Left_Tab: **backward-tab** - Outdent line at current position

Insert: **toggle-overtype** - Toggle status of overtyping mode

Insert-Ctrl-C: **vi-ctrl-c**

Insert-Ctrl-D: **outdent-region** - Outdent the selected region one level of indentation. Set sel to None to use preference to determine selection behavior, or "never-select" to unselect after indent, "always-select" to always select after indent, or "retain-select" to retain current selection after indent.

Insert-Ctrl-H: **backward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character behind the cursor, or the current selection if not empty. ; *Toolbar Search Commands*: Delete character behind the cursor

Insert-Ctrl-J: **new-line** - Place a new line at the current cursor position

Insert-Ctrl-M: **new-line** - Place a new line at the current cursor position

Insert-Ctrl-N: **next-line** - Move to screen next line, optionally repositioning character within line: 'same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Insert-Ctrl-O: **enter-browse-mode(provisional=True)** - Enter editor browse mode

Insert-Ctrl-P: **previous-line** - Move to previous screen line, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Insert-Ctrl-Q: **start-select-block** - Turn on auto-select block mode

Insert-Ctrl-T: **indent-region** - Indent the selected region one level of indentation. Set sel to None to use preference to determine selection behavior, or "never-select" to unselect after indent, "always-select" to always select after indent, or "retain-select" to retain current selection after indent.

Insert-Ctrl-U: **delete-to-start-of-line** - Delete everything between the cursor and start of line

Insert-Ctrl-V: **vi-ctrl-v**

Insert-Ctrl-W: **backward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word behind of the cursor ; *Toolbar Search Commands*: Delete word behind the cursor

Insert-Ctrl-X: **vi-ctrl-x**

Insert-Ctrl-[: **enter-browse-mode** - Enter editor browse mode

Insert-Esc: **enter-browse-mode** - Enter editor browse mode

Left: **backward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one character ; *Toolbar Search Commands*: Move backward one character

Left: **backward-char(wrap=0)** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one character ; *Toolbar Search Commands*: Move backward one character

Next: **forward-page** - Move cursor forward one page

Page_Down: **forward-page** - Move cursor forward one page

Page_Up: **backward-page** - Move cursor backward one page

Prior: **backward-page** - Move cursor backward one page

Replace-Ctrl-C: **enter-browse-mode** - Enter editor browse mode

Replace-Ctrl-D: **outdent-region** - Outdent the selected region one level of indentation. Set sel to None to use preference to determine selection behavior, or "never-select" to unselect after indent, "always-select" to always select after indent, or "retain-select" to retain current selection after indent.

Replace-Ctrl-H: **backward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one character ; *Toolbar Search Commands*: Move backward one character

Replace-Ctrl-J: **new-line** - Place a new line at the current cursor position

Replace-Ctrl-M: **new-line** - Place a new line at the current cursor position

Replace-Ctrl-T: **indent-region** - Indent the selected region one level of indentation. Set sel to None to use preference to determine selection behavior, or "never-select" to unselect after indent, "always-select" to always select after indent, or "retain-select" to retain current selection after indent.

Replace-Ctrl-U: **delete-to-start-of-line** - Delete everything between the cursor and start of line

Replace-Ctrl-W: **backward-word** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move backward one word

Replace-Ctrl-[: **enter-browse-mode** - Enter editor browse mode

Replace-Esc: **enter-browse-mode** - Enter editor browse mode

Return: **new-line** - Place a new line at the current cursor position

Right: **forward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one character ; *Toolbar Search Commands*: Move forward one character

Right: **forward-char(wrap=0)** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one character ; *Toolbar Search Commands*: Move forward one character

Shift-Alt-A: **diff-merge-a-b**

Shift-Alt-B: **diff-merge-b-a**

Shift-Alt-Down: **next-line-extend-rect** - Move to next screen line, adjusting the rectangular selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Shift-Alt-Left: **backward-char-extend-rect** - Move cursor backward one character, adjusting the rectangular selection range to new position

Shift-Alt-N: **diff-next**

Shift-Alt-P: **diff-previous**

Shift-Alt-Right: **forward-char-extend-rect** - Move cursor forward one character, adjusting the rectangular selection range to new position

Shift-Alt-Up: **previous-line-extend-rect** - Move to previous screen line, adjusting the rectangular selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Shift-BackSpace: **backward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character behind the cursor, or the current selection if not empty. ; *Toolbar Search Commands*: Delete character behind the cursor

Shift-Ctrl-F8: **start-select-block** - Turn on auto-select block mode

Shift-Delete: **cut** - Action varies according to focus: *Active Editor Commands*: Cut selected text ; *Search Manager Instance Commands*: Cut selected text ; *Toolbar Search Commands*: Cut selection

Shift-Down: **forward-page** - Move cursor forward one page

Shift-Down: **next-line-extend** - Move to next screen line, adjusting the selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, 'fnb' for first non-blank char, or 'xcode' to simulate XCode style Shift-Alt line selection.

Shift-End: **end-of-line-extend** - Action varies according to focus: *Active Editor Commands*: Move to end of current line, adjusting the selection range to new position ; *Toolbar Search Commands*: Move to the end of the toolbar search entry, extending the selection

Shift-F1: **move-focus** - Move the keyboard focus forward within the Window to the next editable area

Shift-F11: **frame-show** - Show the position (thread and stack frame) where the debugger originally stopped

Shift-F2: Multiple commands (first available is executed):

- **enter-fullscreen** - Hide both the vertical and horizontal tool areas and toolbar, saving previous state so it can be restored later with exit_fullscreen
- **exit-fullscreen** - Restore previous non-fullscreen state of all tools and tool bar

Shift-F3: **search-backward** - Search again using the search manager's current settings in backward direction

Shift-F4: **find-points-of-use** - Find points of use for a symbol. The symbol defaults to the active selection.

Shift-F5: **debug-file** - Start debugging the current file (rather than the main entry point)

Shift-F6: **run-all-tests** - Runs all the tests in testing panel.

Shift-F7: **run-current-tests** - Runs the current test or tests, if possible. The current tests are determined by the current position in the active view. The tests are debugged when debug is True.

Shift-F8: **start-select-char** - Turn on auto-select mode character by character

Shift-F9: Multiple commands (first available is executed):

- **break-enable** - Enable the breakpoint on the current line
- **break-disable** - Disable the breakpoint on current line

Shift-Home: **beginning-of-line-text-extend** - Move to end of the leading white space, if any, on the current line, adjusting the selection range to the new position. If toggle is True, moves to the beginning of the line if already at the end of the leading white space (and vice versa).

Shift-Insert: **paste** - Action varies according to focus: *Active Editor Commands*: Paste text from clipboard ; *Search Manager Instance Commands*: Paste text from clipboard ; *Toolbar Search Commands*: Paste from clipboard

Shift-Left: **backward-char-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one character, adjusting the selection range to new position ; *Toolbar Search Commands*: Move backward one character, extending the selection

Shift-Left: **backward-word** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move backward one word

Shift-Next: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Shift-Page_Down: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Shift-Page_Up: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Shift-Prior: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Shift-Return: **new-line-before** - Place a new line before the current line

Shift-Right: **forward-char-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one character, adjusting the selection range to new position ; *Toolbar Search Commands*: Move forward one character, extending the selection

Shift-Right: **forward-word** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move forward one word

Shift-Tab: **backward-tab** - Outdent line at current position

Shift-Up: **backward-page** - Move cursor backward one page

Shift-Up: **previous-line-extend** - Move to previous screen line, adjusting the selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, 'fnb' for first non-blank char, or 'xcode' to simulate XCode style Shift-Alt line selection.

Tab: **tab-key** - Implement the tab key, the action of which is configurable by preference

Up: **previous-line** - Move to previous screen line, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Visual-!: **filter-selection** - Filter the current selection through an external command and replace the lines with the result

Visual-1: **initiate-numeric-modifier(digit=1)** - VI style repeat/numeric modifier for following command

Visual-2: **initiate-numeric-modifier(digit=2)** - VI style repeat/numeric modifier for following command

Visual-3: **initiate-numeric-modifier(digit=3)** - VI style repeat/numeric modifier for following command

Visual-4: **initiate-numeric-modifier(digit=4)** - VI style repeat/numeric modifier for following command

Visual-5: **initiate-numeric-modifier(digit=5)** - VI style repeat/numeric modifier for following command

Visual-6: **initiate-numeric-modifier(digit=6)** - VI style repeat/numeric modifier for following command

Visual-7: **initiate-numeric-modifier(digit=7)** - VI style repeat/numeric modifier for following command

Visual-8: **initiate-numeric-modifier(digit=8)** - VI style repeat/numeric modifier for following command

Visual-9: **initiate-numeric-modifier(digit=9)** - VI style repeat/numeric modifier for following command

Visual-<: **outdent-lines** - Outdent selected number of lines from cursor position. Set lines to None to indent all the lines in current selection. Set levels to outdent more than one level at a time.

Visual->: **indent-lines** - Indent selected number of lines from cursor position. Set lines to None to indent all the lines in current selection. Set levels to indent more than one level at a time.

Visual-Ctrl-V: **enter-browse-mode** - Enter editor browse mode

Visual-Ctrl-[: **exit-visual-mode** - Exit visual mode and return back to default mode

Visual-Esc: **exit-visual-mode** - Exit visual mode and return back to default mode

Visual-Esc: **exit-visual-mode** - Exit visual mode and return back to default mode

Visual-Shift-A: **enter-insert-mode(pos="after")** - Enter editor insert mode

Visual-Shift-I: **enter-insert-mode(pos="before")** - Enter editor insert mode

Visual-Shift-J: **join-selection** - Join together all lines in given selection (replace newlines with the given delimiter (single space by default)

Visual-Shift-O: **exchange-point-and-mark** - When currently marking text, this exchanges the current position and mark ends of the current selection

Visual-Shift-R: **enter-insert-mode(pos="delete-lines")** - Enter editor insert mode

Visual-Shift-V: **enter-browse-mode** - Enter editor browse mode

Visual-Shift-Y: **move-to-register(unit="line")** - Cut or copy a specified number of characters or lines, or the current selection. Set cut=1 to remove the range of text from the editor after moving to register (otherwise it is just copied). Unit should be one of 'char' or 'line' or 'sel' for current selection.

Visual-c: **enter-insert-mode(pos="delete-sel")** - Enter editor insert mode

Visual-colon: **vi-command-by-name** - Execute a VI command (implements ":" commands from VI)

Visual-d: **move-to-register(unit="sel", cut=1)** - Cut or copy a specified number of characters or lines, or the current selection. Set cut=1 to remove the range of text from the editor after moving to register (otherwise it is just copied). Unit should be one of 'char' or 'line' or 'sel' for current selection.

Visual-g Visual-Shift-J: **join-selection(delim="")** - Join together all lines in given selection (replace newlines with the given delimiter (single space by default)

Visual-g Visual-q: **fill-paragraph** - Attempt to auto-justify the paragraph around the current start of selection

Visual-o: **exchange-point-and-mark** - When currently marking text, this exchanges the current position and mark ends of the current selection

Visual-r: **replace-char** - Replace num characters with given character. Set line_mode to multiline to allow replacing across lines, extend to replace on current line and then extend the line length, and restrict to replace only if enough characters exist on current line after cursor position.

Visual-s: **enter-insert-mode(pos="delete-sel")** - Enter editor insert mode

Visual-v: **enter-browse-mode** - Enter editor browse mode

Visual-x: **move-to-register(unit="sel", cut=1)** - Cut or copy a specified number of characters or lines, or the current selection. Set cut=1 to remove the range of text from the editor after moving to register (otherwise it is just copied). Unit should be one of 'char' or 'line' or 'sel' for current selection.

Visual-y: **move-to-register(unit="sel")** - Cut or copy a specified number of characters or lines, or the current selection. Set cut=1 to remove the range of text from the editor after moving to register (otherwise it is just copied). Unit should be one of 'char' or 'line' or 'sel' for current selection.

21.4. Visual Studio Personality

This section documents all the default key bindings for the **Visual Studio** keyboard personality, set by the **Personality** preference.

Alt-1: **fold-python-methods** - Fold up all Python methods, expand all classes, and leave other fold points alone

Alt-2: **fold-python-classes** - Fold up all Python classes but leave other fold points alone

Alt-3: **fold-python-classes-and-defs** - Fold up all Python classes, methods, and functions but leave other fold points alone

Alt-BackSpace: **backward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word behind of the cursor ; *Toolbar Search Commands*: Delete word behind the cursor

Alt-Delete: **backward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word behind of the cursor ; *Toolbar Search Commands*: Delete word behind the cursor

Alt-Down: **fold-expand-more-current** - Expand the current fold point one more level

Alt-End: **fold-expand-all** - Expand all fold points in the current file

Alt-F11: **prev-points-of-use-match** - Display the previous match in the active points of use tool

Alt-F12: **next-points-of-use-match** - Display the next match in the active points of use tool

Alt-F3: **search** - Bring up the search manager in search mode.

Alt-F4: **close-window** - Close the current window and all documents and panels in it

Alt-F5: **run-to-cursor** - Run to current cursor position

Alt-F6: **run-failed-tests** - Re-run all the previously failed tests. The tests are debugged when debug is True.

Alt-F7: **run-last-tests** - Run again the last group of tests that were run. The tests are debugged when debug is True.

Alt-F7: **view-project-properties** - View or change project-wide properties

Alt-Home: **fold-collapse-all** - Collapse all fold points in the current file

Alt-Left: **visit-history-previous** - Move back in history to previous visited editor position

Alt-Page_Down: **fold-expand-all-current** - Expand the current fold point completely

Alt-Page_Up: **fold-collapse-all-current** - Collapse the current fold point completely

Alt-Return: **new-line** - Place a new line at the current cursor position

Alt-Right: **visit-history-next** - Move forward in history to next visited editor position

Alt-Slash: **fold-toggle** - Toggle the current fold point

Alt-Up: **fold-collapse-more-current** - Collapse the current fold point one more level

Alt-comma: **query-replace** - Initiate incremental mini-search query/replace from the cursor position.

Alt-period: **replace-string** - Replace all occurrences of a string from the cursor position to end of file.

BackSpace: **backward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character behind the cursor, or the current selection if not empty. ; *Toolbar Search Commands*: Delete character behind the cursor

Ctrl-0: **next-document** - Move to the next document alphabetically in the list of documents open in the current window

Ctrl-1: **activate-file-option-menu** - Activate the file menu for the editor.

Ctrl-2: **activate-symbol-option-menu-1** - Activate the 1st symbol menu for the editor.

Ctrl-3: **activate-symbol-option-menu-2** - Activate the 2nd symbol menu for the editor.

Ctrl-4: **activate-symbol-option-menu-3** - Activate the 3rd symbol menu for the editor.

Ctrl-5: **activate-symbol-option-menu-4** - Activate the 4th symbol menu for the editor.

Ctrl-6: **activate-symbol-option-menu-5** - Activate the 5th symbol menu for the editor.

Ctrl-7 C: **use-lexer-cpp** - Force syntax highlighting for C/C++ source

Ctrl-7 H: **use-lexer-html** - Force syntax highlighting for HTML

Ctrl-7 M: **use-lexer-makefile** - Force syntax highlighting for make files

Ctrl-7 N: **use-lexer-none** - Use no syntax highlighting

Ctrl-7 P: **use-lexer-python** - Force syntax highlighting for Python source

Ctrl-7 S: **use-lexer-sql** - Force syntax highlighting for SQL

Ctrl-7 X: **use-lexer-xml** - Force syntax highlighting for XML files

Ctrl-8: **recent-document** - Switches to previous document most recently visited in the current window or window set if in one-window-per-editor windowing mode.

Ctrl-9: **previous-document** - Move to the previous document alphabetically in the list of documents open in the current window

Ctrl-=: **indent-to-match** - Indent the current line or selected region to match indentation of preceding non-blank line. Set toggle=True to indent instead of one level higher if already at the matching position.

Ctrl-A: **select-all** - Select all text in the editor

Ctrl-Alt-B: **search-sel-backward** - Search backward using current selection

Ctrl-Alt-Comma: **query-replace-regex** - Initiate incremental mini-search query/replace from the cursor position. The search string is treated as a regular expression.

Ctrl-Alt-D: **selection-add-next-occurence(skip_current=True)** - Add another selection containing the text of the current selection. If skip_current is true, the

current selection will be deselected. If nothing is currently selected, select the current word. Searches backwards if reverse is true.

Ctrl-Alt-Down: **goto-next-bookmark(current_file_only=True)** - Go to the next bookmark, or the first one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-Alt-F: **search-sel-forward** - Search forward using current selection

Ctrl-Alt-F6: **debug-failed-tests** - Re-run all the previously failed tests in the debugger.

Ctrl-Alt-F7: **debug-last-tests** - Debug the last group of tests that were run.

Ctrl-Alt-G: **goto-bookmark** - Goto named bookmark

Ctrl-Alt-K: **show-bookmarks** - Show a list of all currently defined bookmarks

Ctrl-Alt-Left: **goto-previous-bookmark** - Go to the previous bookmark in the bookmark list, or the last one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-Alt-M: **set-bookmark** - Set a bookmark at current location on the editor. Mark is the project-wide textual name of the bookmark.

Ctrl-Alt-Right: **goto-next-bookmark** - Go to the next bookmark, or the first one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-Alt-T: **toggle-bookmark** - Set or remove a bookmark at current location on the editor. When set, the name of the bookmark is set to an auto-generated default.

Ctrl-Alt-Up: **goto-previous-bookmark(current_file_only=True)** - Go to the previous bookmark in the bookmark list, or the last one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-Alt-period: **replace-string-regex** - Replace all occurrences of a string from the cursor position to end of file. The search string is treated as a regular expression.

Ctrl-Apostrophe: **enclose(start="'", end="'")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-B: **isearch-sel-forward** - Action varies according to focus: *Active Editor Commands*: Initiate incremental mini-search forward from the cursor position, using current selection as the search string. Set persist=False to do the search but end the interactive search session immediately.; *Document Viewer Commands*: Initiate incremental mini-search forward from the cursor position, using current selection as the search string. Set persist=False to do the search but end the interactive search session immediately.

Ctrl-BackSpace: **backward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word behind of the cursor ; *Toolbar Search Commands*: Delete word behind the cursor

Ctrl-Braceleft: **enclose(start="{", end="}")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-Bracketleft: **enclose(start="[", end="]")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-C: **copy** - Action varies according to focus: *Active Editor Commands*: Copy selected text ; *Document Viewer Commands*: Copy any selected text. ; *Exceptions Commands*: Copy the exception traceback to the clipboard ; *Search Manager Instance Commands*: Copy selected text ; *Toolbar Search Commands*: Cut selection

Ctrl-Comma: **next-window** - Switch to the next window alphabetically by title

Ctrl-D: **toolbar-search-focus** - Move focus to toolbar search entry.

Ctrl-Delete: **forward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word in front of the cursor ; *Toolbar Search Commands*: Delete word in front of the cursor

Ctrl-Down: **select-less** - Select less code; undoes the last select-more command

Ctrl-E: **brace-match** - Match brace at current cursor position, selecting all text between the two and hilighting the braces

Ctrl-End: **end-of-document** - Move cursor to end of document

Ctrl-F: **search** - Bring up the search manager in search mode.

Ctrl-F10: **debug-to-cursor**

Ctrl-F12: **command-by-name** - Execute given command by name, collecting any args as needed

Ctrl-F3: **search-sel-forward** - Search forward using current selection

Ctrl-F4: **close** - Close active document. Abandon any changes when ignore_changes is True. Close empty windows when close_window is true and quit if all document windows closed when can_quit is true.

Ctrl-F5: **debug-file** - Start debugging the current file (rather than the main entry point)

Ctrl-F5: **debug-kill** - Stop debugging

Ctrl-F6: **step-over** - Step over current instruction

Ctrl-F8: **start-select-line** - Turn on auto-select mode line by line

Ctrl-F9: **break-clear-all** - Clear all breakpoints

Ctrl-G: **goto-line** - Position cursor at start of given line number

Ctrl-H: **replace** - Bring up the search manager in replace mode.

Ctrl-Home: **start-of-document** - Move cursor to start of document

Ctrl-I: **isearch-forward** - Action varies according to focus: *Active Editor Commands*: Initiate incremental mini-search forward from the cursor position, optionally entering the given search string ; *Document Viewer Commands*: Initiate incremental mini-search forward from the cursor position, optionally entering the given search string.

Ctrl-Insert: **copy** - Action varies according to focus: *Active Editor Commands*: Copy selected text ; *Document Viewer Commands*: Copy any selected text. ; *Exceptions Commands*: Copy the exception traceback to the clipboard ; *Search Manager Instance Commands*: Copy selected text ; *Toolbar Search Commands*: Cut selection

Ctrl-J: **show-autocompleter** - Show the auto-completer for current cursor position

Ctrl-K Ctrl-C: **comment-out-region** - Comment out the selected region. The style of commenting can be controlled with the style argument: 'indented' uses the default comment style indented at end of leading white space and 'block' uses a block comment in column zero. If not given, the style configured with the Editor / Block Comment Style preference is used. Each call adds a level of commenting.

Ctrl-K Ctrl-F: **fill-paragraph** - Attempt to auto-justify the paragraph around the current start of selection

Ctrl-K Ctrl-K: **toggle-bookmark** - Set or remove a bookmark at current location on the editor. When set, the name of the bookmark is set to an auto-generated default.

Ctrl-K Ctrl-N: **goto-next-bookmark** - Go to the next bookmark, or the first one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-K Ctrl-O: **open-from-keyboard** - Open a file from disk using keyboard-driven selection of the file

Ctrl-K Ctrl-P: **goto-previous-bookmark** - Go to the previous bookmark in the bookmark list, or the last one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-K Ctrl-S: **switch-document** - Switches to named document. Name may either be the complete name or the last path component of a path name.

Ctrl-K Ctrl-T: **comment-toggle** - Toggle commenting out of the selected lines. The style of commenting can be controlled with the style argument: 'indented' uses the default comment style indented at end of leading white space and 'block' uses a

block comment in column zero. If not given, the style configured with the Editor / Block Comment Style preference is used.

Ctrl-K Ctrl-U: **uncomment-out-region** - Uncomment out the selected region if commented out. If one_level is True then each call removes only one level of commenting.

Ctrl-L: **cut-line** - Cut the current line(s) to clipboard.

Ctrl-Left: **backward-word** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move backward one word

Ctrl-Less: **enclose(start="<", end=">")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-M: **execute-kbd-macro** - Execute most recently recorded keyboard macro. If register is None then the user is asked to enter a letter a-z for the register where the macro is filed. Otherwise, register 'a' is used by default.

Ctrl-Minus: **visit-history-previous** - Move back in history to previous visited editor position

Ctrl-Minus: **zoom-out** - Action varies according to focus: *Document Viewer Commands*: Decrease documentation font size; *General Editor Commands*: Zoom out, increasing the text display size temporarily by one font size

Ctrl-N: **new-file** - Create a new file

Ctrl-Next: **forward-page** - Move cursor forward one page

Ctrl-O: **open-gui** - Open a file from disk, prompting with file selection dialog if necessary

Ctrl-P: **print-view** - Print active editor document

Ctrl-Page_Down: **next-document** - Move to the next document alphabetically in the list of documents open in the current window

Ctrl-Page_Up: **previous-document** - Move to the previous document alphabetically in the list of documents open in the current window

Ctrl-Parenleft: **enclose(start="(", end=")")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-Plus: **zoom-in** - Action varies according to focus: *Document Viewer Commands*: Increase documentation font size; *General Editor Commands*: Zoom in, increasing the text display size temporarily by one font size

Ctrl-Pointer_Button1: **goto-clicked-symbol-defn** - Goto the definition of the source symbol that was last clicked on, optionally showing the definition in another split if one is available and other_split is True.

Ctrl-Prior: **backward-page** - Move cursor backward one page

Ctrl-Q: **quit** - Quit the application.

Ctrl-Quotedbl: **enclose(start='"', end='")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-Quoteleft: **begin-visited-document-cycle(move_back=True, back_key="Ctrl-Quoteleft", forward_key="Ctrl-AsciiTilde")** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-R: **replace** - Bring up the search manager in replace mode.

Ctrl-Return: **new-line-after** - Place a new line after the current line

Ctrl-Right: **forward-word** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move forward one word

Ctrl-S: **save** - Save active document. Also close it if close is True.

Ctrl-Shift-B: **isearch-sel-backward** - Initiate incremental mini-search backward from the cursor position, using current selection as the search string. Set persist=False to do the search but end the interactive search session immediately.

Ctrl-Shift-D: **selection-add-next-occurence** - Add another selection containing the text of the current selection. If skip_current is true, the current selection will be deselected. If nothing is currently selected, select the current word. Searches backwards if reverse is true.

Ctrl-Shift-Delete: **delete-lines**

Ctrl-Shift-Down: **scroll-text-down** - Scroll text down a line w/o moving cursor's relative position on screen. Repeat is number of lines or if >0 and <1.0 then percent of screen. Set move_cursor to False to leave cursor in current position within the source, otherwise it is moved so the cursor remains on same screen line.

Ctrl-Shift-End: **end-of-document-extend** - Move cursor to end of document, adjusting the selection range to new position

Ctrl-Shift-F: **batch-search** - Search on current selection using the Search in Files tool. The look_in argument gets entered in the look in field if not None or ''. The current selection is put into the search field if it doesn't span multiple lines and

either use_selection is true or there's nothing in the search field. The given search text is used instead, if provided

Ctrl-Shift-F3: **search-sel-backward** - Search backward using current selection

Ctrl-Shift-F5: **debug-stop** - Pause free-running execution at current program counter

Ctrl-Shift-F6: **debug-all-tests** - Debug all the tests in testing panel.

Ctrl-Shift-F7: **debug-current-tests** - Runs the current test or tests, if possible. The current tests are determined by the current position in the active view.

Ctrl-Shift-F9: Multiple commands (first available is executed):

- **break-disable-all** - Disable all breakpoints
- **break-enable-all** - Enable all breakpoints

Ctrl-Shift-G: **search-backward** - Search again using the search manager's current settings in backward direction

Ctrl-Shift-H: **batch-replace** - Display search and replace in files tool.

Ctrl-Shift-Home: **start-of-document-extend** - Move cursor to start of document, adjusting the selection range to new position

Ctrl-Shift-I: **add-current-file-to-project** - Add the frontmost currently open file to project

Ctrl-Shift-ISO_Left_Tab: **begin-visited-document-cycle(move_back=False)** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-Shift-Left: **backward-word-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one word, adjusting the selection range to new position. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move backward one word, extending the selection

Ctrl-Shift-Next: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Ctrl-Shift-O: **open-from-project** - Open document from the project via the Open From Project dialog. The given fragment is used as the initial fragment filter and if it is None, the selected text or the symbol under the cursor is used. If skip_if_unique is true, the file is opened without the dialog being displayed if only one filename matches the fragment.

Ctrl-Shift-Page_Down: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Ctrl-Shift-Page_Up: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Ctrl-Shift-Prior: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Ctrl-Shift-R: **batch-replace** - Display search and replace in files tool.

Ctrl-Shift-Right: **forward-word-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one word, adjusting the selection range to new position. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move forward one word, extending the selection

Ctrl-Shift-S: **save-all** - Save all unsaved items, prompting for names for any new items that don't have a filename already.

Ctrl-Shift-T: **find-symbol** - Allow user to visit point of definition of a source symbol in the current editor context by typing a fragment of the name

Ctrl-Shift-Tab: **begin-visited-document-cycle(move_back=False)** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-Shift-U: **case-upper** - Change case of the current selection, or character ahead of the cursor if there is no selection, to upper case

Ctrl-Shift-U: **isearch-backward** - Action varies according to focus: *Active Editor Commands*: Initiate incremental mini-search backward from the cursor position, optionally entering the given search string ; *Document Viewer Commands*: Initiate incremental mini-search backward from the cursor position, optionally entering the given search string.

Ctrl-Shift-Up: **scroll-text-up** - Scroll text up a line w/o moving cursor's relative position on screen. Repeat is number of lines or if >0 and <1.0 then percent of screen. Set move_cursor to False to leave cursor in current position within the source, otherwise it is moved so the cursor remains on same screen line.

Ctrl-Slash: **command-by-name** - Execute given command by name, collecting any args as needed

Ctrl-T: **forward-tab** - Action varies according to focus: *Active Editor Commands*: Place a tab character at the current cursor position ; *Search Manager Instance Commands*: Place a forward tab at the current cursor position in search or replace string

Ctrl-Tab: **begin-visited-document-cycle(move_back=True)** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-U: **case-lower** - Change case of the current selection, or character ahead of the cursor if there is no selection, to lower case

Ctrl-Underscore: **visit-history-next** - Move forward in history to next visited editor position

Ctrl-Up: **select-more** - Select more code on either the current line or larger multi-line blocks.

Ctrl-V: **paste** - Action varies according to focus: *Active Editor Commands*: Paste text from clipboard ; *Search Manager Instance Commands*: Paste text from clipboard ; *Toolbar Search Commands*: Paste from clipboard

Ctrl-W: **close** - Close active document. Abandon any changes when ignore_changes is True. Close empty windows when close_window is true and quit if all document windows closed when can_quit is true.

Ctrl-X: **cut** - Action varies according to focus: *Active Editor Commands*: Cut selected text ; *Search Manager Instance Commands*: Cut selected text ; *Toolbar Search Commands*: Cut selection

Ctrl-Y: **redo** - Redo last action

Ctrl-Z: **undo** - Undo last action

Ctrl-]: **brace-match** - Match brace at current cursor position, selecting all text between the two and hilighting the braces

Ctrl-greater: **indent-region** - Indent the selected region one level of indentation. Set sel to None to use preference to determine selection behavior, or "never-select" to unselect after indent, "always-select" to always select after indent, or "retain-select" to retain current selection after indent.

Ctrl-less: **outdent-region** - Outdent the selected region one level of indentation. Set sel to None to use preference to determine selection behavior, or "never-select" to unselect after indent, "always-select" to always select after indent, or "retain-select" to retain current selection after indent.

Ctrl-parenleft: **start-kbd-macro** - Start definition of a keyboard macro. If register=None then the user is prompted to enter a letter a-z under which to file the macro. Otherwise, register 'a' is used by default.

Ctrl-parenright: **stop-kbd-macro** - Stop definition of a keyboard macro

Ctrl-space: **show-autocompleter** - Show the auto-completer for current cursor position

Delete: **forward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character in front of the cursor ; *Toolbar Search Commands*: Delete character in front of the cursor

Down: **next-line** - Move to screen next line, optionally repositioning character within line: 'same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

End: **end-of-line** - Action varies according to focus: *Active Editor Commands*: Move to end of current line; *Toolbar Search Commands*: Move to the end of the toolbar search entry

F1: Multiple commands (first available is executed):

- **show-horizontal-tools** - Show the horizontal tool area
- **minimize-horizontal-tools** - Minimize the horizontal tool area

F10: **step-over-statement** - Step over current statement

F11: **frame-up** - Move up the current debug stack

F11: **step-into** - Step into current execution point, or start debugging at first line

F12: **frame-down** - Move down the current debug stack

F2: Multiple commands (first available is executed):

- **show-vertical-tools** - Show the vertical tool area
- **minimize-vertical-tools** - Minimize the vertical tool area

F3: **search-forward** - Search again using the search manager's current settings in forward direction

F4: **goto-selected-symbol-defn** - Goto the definition of the selected source symbol, optionally showing the definition in another split if one is available and other_split is True.

F5: **debug-continue** - Continue (or start) running, to next breakpoint

F6: **step-over-statement** - Step over current statement

F7: **step-into** - Step into current execution point, or start debugging at first line

F8: **step-out** - Step out of the current function or method

F9: Multiple commands (first available is executed):

- **break-set** - Set a new regular breakpoint on current line
- **break-clear** - Clear the breakpoint on the current line

Home: **beginning-of-line-text** - Move to end of the leading white space, if any, on the current line. If toggle is True, moves to the beginning of the line if already at the end of the leading white space (and vice versa).

ISO_Left_Tab: **backward-tab** - Outdent line at current position

Insert: **toggle-overtype** - Toggle status of overtyping mode

Left: **backward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one character ; *Toolbar Search Commands*: Move backward one character

Next: **forward-page** - Move cursor forward one page

Page_Down: **forward-page** - Move cursor forward one page

Page_Up: **backward-page** - Move cursor backward one page

Prior: **backward-page** - Move cursor backward one page

Return: **new-line** - Place a new line at the current cursor position

Right: **forward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one character ; *Toolbar Search Commands*: Move forward one character

Shift-Alt-A: **diff-merge-a-b**

Shift-Alt-B: **diff-merge-b-a**

Shift-Alt-Down: **next-line-extend-rect** - Move to next screen line, adjusting the rectangular selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Shift-Alt-Left: **backward-char-extend-rect** - Move cursor backward one character, adjusting the rectangular selection range to new position

Shift-Alt-N: **diff-next**

Shift-Alt-P: **diff-previous**

Shift-Alt-Right: **forward-char-extend-rect** - Move cursor forward one character, adjusting the rectangular selection range to new position

Shift-Alt-Up: **previous-line-extend-rect** - Move to previous screen line, adjusting the rectangular selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Shift-BackSpace: **backward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character behind the cursor, or the current selection if not empty. ; *Toolbar Search Commands*: Delete character behind the cursor

Shift-Ctrl-F8: **start-select-block** - Turn on auto-select block mode

Shift-Delete: **cut** - Action varies according to focus: *Active Editor Commands*: Cut selected text ; *Search Manager Instance Commands*: Cut selected text ; *Toolbar Search Commands*: Cut selection

Shift-Delete: **cut-selection-or-line** - Cut the current selection or current line if there is no selection. The text is placed on the clipboard.

Shift-Down: **next-line-extend** - Move to next screen line, adjusting the selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, 'fnb' for first non-blank char, or 'xcode' to simulate XCode style Shift-Alt line selection.

Shift-End: **end-of-line-extend** - Action varies according to focus: *Active Editor Commands*: Move to end of current line, adjusting the selection range to new position ; *Toolbar Search Commands*: Move to the end of the toolbar search entry, extending the selection

Shift-F1: **move-focus** - Move the keyboard focus forward within the Window to the next editable area

Shift-F11: **frame-show** - Show the position (thread and stack frame) where the debugger originally stopped

Shift-F11: **step-out** - Step out of the current function or method

Shift-F2: Multiple commands (first available is executed):

- **enter-fullscreen** - Hide both the vertical and horizontal tool areas and toolbar, saving previous state so it can be restored later with exit_fullscreen
- **exit-fullscreen** - Restore previous non-fullscreen state of all tools and tool bar

Shift-F3: **search-backward** - Search again using the search manager's current settings in backward direction

Shift-F4: **find-points-of-use** - Find points of use for a symbol. The symbol defaults to the active selection.

Shift-F5: **debug-file** - Start debugging the current file (rather than the main entry point)

Shift-F5: **debug-kill** - Stop debugging

Shift-F6: **run-all-tests** - Runs all the tests in testing panel.

Shift-F7: **run-current-tests** - Runs the current test or tests, if possible. The current tests are determined by the current position in the active view. The tests are debugged when debug is True.

Shift-F8: **start-select-char** - Turn on auto-select mode character by character

Shift-F9: Multiple commands (first available is executed):

- **break-enable** - Enable the breakpoint on the current line
- **break-disable** - Disable the breakpoint on current line

Shift-Home: **beginning-of-line-text-extend** - Move to end of the leading white space, if any, on the current line, adjusting the selection range to the new position. If toggle is True, moves to the beginning of the line if already at the end of the leading white space (and vice versa).

Shift-Insert: **paste** - Action varies according to focus: *Active Editor Commands*: Paste text from clipboard ; *Search Manager Instance Commands*: Paste text from clipboard ; *Toolbar Search Commands*: Paste from clipboard

Shift-Left: **backward-char-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one character, adjusting the selection range to new position ; *Toolbar Search Commands*: Move backward one character, extending the selection

Shift-Next: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Shift-Page_Down: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Shift-Page_Up: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Shift-Prior: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Shift-Return: **new-line-before** - Place a new line before the current line

Shift-Right: **forward-char-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one character, adjusting the selection range to new position ; *Toolbar Search Commands*: Move forward one character, extending the selection

Shift-Tab: **backward-tab** - Outdent line at current position

Shift-Up: **previous-line-extend** - Move to previous screen line, adjusting the selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, 'fnb' for first non-blank char, or 'xcode' to simulate XCode style Shift-Alt line selection.

Tab: **tab-key** - Implement the tab key, the action of which is configurable by preference

Up: **previous-line** - Move to previous screen line, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Visual-Esc: **exit-visual-mode** - Exit visual mode and return back to default mode

21.5. OS X Personality

This section documents all the default key bindings for the **OS X** keyboard personality, set by the **Personality** preference.

Alt-Down: **next-line(cursor="end")** - Move to screen next line, optionally repositioning character within line: 'same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Alt-Up: **previous-line(cursor="start")** - Move to previous screen line, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Backspace: **backward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character behind the cursor, or the current selection if not empty. ; *Toolbar Search Commands*: Delete character behind the cursor

Command-0: **next-document** - Move to the next document alphabetically in the list of documents open in the current window

Command-1: **activate-file-option-menu** - Activate the file menu for the editor.

Command-2: **activate-symbol-option-menu-1** - Activate the 1st symbol menu for the editor.

Command-3: **activate-symbol-option-menu-2** - Activate the 2nd symbol menu for the editor.

Command-4: **activate-symbol-option-menu-3** - Activate the 3rd symbol menu for the editor.

Command-5: **activate-symbol-option-menu-4** - Activate the 4th symbol menu for the editor.

Command-6: **activate-symbol-option-menu-5** - Activate the 5th symbol menu for the editor.

Command-7 C: **use-lexer-cpp** - Force syntax highlighting for C/C++ source

Command-7 H: **use-lexer-html** - Force syntax highlighting for HTML

Command-7 M: **use-lexer-makefile** - Force syntax highlighting for make files

Command-7 N: **use-lexer-none** - Use no syntax highlighting

Command-7 P: **use-lexer-python** - Force syntax highlighting for Python source

Command-7 S: **use-lexer-sql** - Force syntax highlighting for SQL

Command-7 X: **use-lexer-xml** - Force syntax highlighting for XML files

Command-8: **recent-document** - Switches to previous document most recently visited in the current window or window set if in one-window-per-editor windowing mode.

Command-9: **previous-document** - Move to the previous document alphabetically in the list of documents open in the current window

Command-A: **select-all** - Select all text in the editor

Command-Alt-Minus: **fold-python-methods** - Fold up all Python methods, expand all classes, and leave other fold points alone

Command-Apostrophe: **comment-out-region** - Comment out the selected region. The style of commenting can be controlled with the style argument: 'indented' uses the default comment style indented at end of leading white space and 'block' uses a block comment in column zero. If not given, the style configured with the Editor / Block Comment Style preference is used. Each call adds a level of commenting.

Command-Asterisk: **fold-expand-all-current** - Expand the current fold point completely

Command-B: **brace-match** - Match brace at current cursor position, selecting all text between the two and hilighting the braces

Command-Backslash: **indent-to-match** - Indent the current line or selected region to match indentation of preceding non-blank line. Set toggle=True to indent instead of one level higher if already at the matching position.

Command-Bracketleft: **outdent-region** - Outdent the selected region one level of indentation. Set sel to None to use preference to determine selection behavior, or "never-select" to unselect after indent, "always-select" to always select after indent, or "retain-select" to retain current selection after indent.

Command-Bracketright: **indent-region** - Indent the selected region one level of indentation. Set sel to None to use preference to determine selection behavior, or "never-select" to unselect after indent, "always-select" to always select after indent, or "retain-select" to retain current selection after indent.

Command-C: **copy** - Action varies according to focus: *Active Editor Commands*: Copy selected text ; *Document Viewer Commands*: Copy any selected text. ; *Exceptions Commands*: Copy the exception traceback to the clipboard ; *Search Manager Instance Commands*: Copy selected text ; *Toolbar Search Commands*: Cut selection

Command-Comma: **show-preferences-gui** - Edit the preferences file using the preferences GUI, optionally opening to the section that contains the given preference by name

Command-Ctrl-Asterisk: **fold-expand-all** - Expand all fold points in the current file

Command-Ctrl-Minus: **fold-collapse-all** - Collapse all fold points in the current file

Command-Ctrl-R: **replace-and-search** - Replace current selection and search again.

Command-Ctrl-Slash: **fold-python-classes** - Fold up all Python classes but leave other fold points alone

Command-D: **set-bookmark** - Set a bookmark at current location on the editor. Mark is the project-wide textual name of the bookmark.

Command-Down: **end-of-document** - Move cursor to end of document

Command-E: **search-sel-forward** - Search forward using current selection

Command-Equal: **fold-python-classes-and-defs** - Fold up all Python classes, methods, and functions but leave other fold points alone

Command-F: **search** - Bring up the search manager in search mode.

Command-F12: **command-by-name** - Execute given command by name, collecting any args as needed

Command-F3: **search-sel-forward** - Search forward using current selection

Command-F4: **close** - Close active document. Abandon any changes when ignore_changes is True. Close empty windows when close_window is true and quit if all document windows closed when can_quit is true.

Command-F5: **debug-kill** - Stop debugging

Command-F8: **start-select-line** - Turn on auto-select mode line by line

Command-F9: **break-clear-all** - Clear all breakpoints

Command-G: **search-forward** - Search again using the search manager's current settings in forward direction

Command-H: **toggle-bookmark** - Set or remove a bookmark at current location on the editor. When set, the name of the bookmark is set to an auto-generated default.

Command-I: **view-file-properties** - View project properties for a particular file (current file if none is given)

Command-J: **fill-paragraph** - Attempt to auto-justify the paragraph around the current start of selection

Command-L: **goto-line** - Position cursor at start of given line number

Command-Left: **beginning-of-line** - Action varies according to focus: *Active Editor Commands*: Move to beginning of current line. When toggle is True, moves to the end of the leading white space if already at the beginning of the line (and vice versa).; *Toolbar Search Commands*: Move to the beginning of the toolbar search entry

Command-M: **execute-kbd-macro** - Execute most recently recorded keyboard macro. If register is None then the user is asked to enter a letter a-z for the register where the macro is filed. Otherwise, register 'a' is used by default.

Command-Minus: **fold-collapse-all-current** - Collapse the current fold point completely

Command-N: **new-file** - Create a new file

Command-O: **open-gui** - Open a file from disk, prompting with file selection dialog if necessary

Command-Option-F6: **debug-failed-tests** - Re-run all the previously failed tests in the debugger.

Command-Option-F7: **debug-last-tests** - Debug the last group of tests that were run.

Command-P: **print-view** - Print active editor document

Command-Plus: **fold-expand-more-current** - Expand the current fold point one more level

Command-Pointer_Button1: **goto-clicked-symbol-defn** - Goto the definition of the source symbol that was last clicked on, optionally showing the definition in another split if one is available and other_split is True.

Command-Q: **quit** - Quit the application.

Command-Question: **show-document** - Show the given documentation section

Command-Quotedbl: **uncomment-out-region** - Uncomment out the selected region if commented out. If one_level is True then each call removes only one level of commenting.

Command-R: **replace** - Bring up the search manager in replace mode.

Command-Return: **new-line** - Place a new line at the current cursor position

Command-Right: **end-of-line** - Action varies according to focus: *Active Editor Commands*: Move to end of current line; *Toolbar Search Commands*: Move to the end of the toolbar search entry

Command-S: **save** - Save active document. Also close it if close is True.

Command-Semicolon: **comment-toggle** - Toggle commenting out of the selected lines. The style of commenting can be controlled with the style argument: 'indented' uses the default comment style indented at end of leading white space and 'block'

uses a block comment in column zero. If not given, the style configured with the Editor / Block Comment Style preference is used.

Command-Shift-D: **goto-bookmark** - Goto named bookmark

Command-Shift-Down: **end-of-document-extend** - Move cursor to end of document, adjusting the selection range to new position

Command-Shift-F: **batch-search** - Search on current selection using the Search in Files tool. The look_in argument gets entered in the look in field if not None or ''. The current selection is put into the search field if it doesn't span multiple lines and either use_selection is true or there's nothing in the search field. The given search text is used instead, if provided

Command-Shift-F3: **search-sel-backward** - Search backward using current selection

Command-Shift-F5: **debug-stop** - Pause free-running execution at current program counter

Command-Shift-F6: **debug-all-tests** - Debug all the tests in testing panel.

Command-Shift-F7: **debug-current-tests** - Runs the current test or tests, if possible. The current tests are determined by the current position in the active view.

Command-Shift-G: **search-backward** - Search again using the search manager's current settings in backward direction

Command-Shift-I: **add-current-file-to-project** - Add the frontmost currently open file to project

Command-Shift-K: **show-bookmarks** - Show a list of all currently defined bookmarks

Command-Shift-Left: **beginning-of-line-extend** - Action varies according to focus: *Active Editor Commands*: Move to beginning of current line, adjusting the selection range to the new position. When toggle is True, moves to the end of the leading white space if already at the beginning of the line (and vice versa).; *Toolbar Search Commands*: Move to the beginning of the toolbar search entry, extending the selection

Command-Shift-O: **open-from-project** - Open document from the project via the Open From Project dialog. The given fragment is used as the initial fragment filter and if it is None, the selected text or the symbol under the cursor is used. If skip_if_unique is true, the file is opened without the dialog being displayed if only one filename matches the fragment.

Command-Shift-R: **batch-replace** - Display search and replace in files tool.

Command-Shift-Right: **end-of-line-extend** - Action varies according to focus: *Active Editor Commands*: Move to end of current line, adjusting the selection range

to new position ; *Toolbar Search Commands*: Move to the end of the toolbar search entry, extending the selection

Command-Shift-S: **save-as** - Save active document to a new file

Command-Shift-T: **find-symbol** - Allow user to visit point of definition of a source symbol in the current editor context by typing a fragment of the name

Command-Shift-U: **isearch-backward** - Action varies according to focus: *Active Editor Commands*: Initiate incremental mini-search backward from the cursor position, optionally entering the given search string ; *Document Viewer Commands*: Initiate incremental mini-search backward from the cursor position, optionally entering the given search string.

Command-Shift-Up: **start-of-document-extend** - Move cursor to start of document, adjusting the selection range to new position

Command-Shift-W: **close** - Close active document. Abandon any changes when ignore_changes is True. Close empty windows when close_window is true and quit if all document windows closed when can_quit is true.

Command-Shift-Y: **selection-add-next-occurence(skip_current=True)** - Add another selection containing the text of the current selection. If skip_current is true, the current selection will be deselected. If nothing is currently selected, select the current word. Searches backwards if reverse is true.

Command-Shift-Z: **redo** - Redo last action

Command-Slash: **fold-toggle** - Toggle the current fold point

Command-T: **search** - Bring up the search manager in search mode.

Command-U: **isearch-forward** - Action varies according to focus: *Active Editor Commands*: Initiate incremental mini-search forward from the cursor position, optionally entering the given search string ; *Document Viewer Commands*: Initiate incremental mini-search forward from the cursor position, optionally entering the given search string.

Command-Underscore: **fold-collapse-more-current** - Collapse the current fold point one more level

Command-Up: **start-of-document** - Move cursor to start of document

Command-V: **paste** - Action varies according to focus: *Active Editor Commands*: Paste text from clipboard ; *Search Manager Instance Commands*: Paste text from clipboard ; *Toolbar Search Commands*: Paste from clipboard

Command-W: **close** - Close active document. Abandon any changes when ignore_changes is True. Close empty windows when close_window is true and quit if all document windows closed when can_quit is true.

Command-X: **cut** - Action varies according to focus: *Active Editor Commands*: Cut selected text ; *Search Manager Instance Commands*: Cut selected text ; *Toolbar Search Commands*: Cut selection

Command-Y: **selection-add-next-occurence** - Add another selection containing the text of the current selection. If skip_current is true, the current selection will be deselected. If nothing is currently selected, select the current word. Searches backwards if reverse is true.

Command-Z: **undo** - Undo last action

Command-parenleft: **start-kbd-macro** - Start definition of a keyboard macro. If register=None then the user is prompted to enter a letter a-z under which to file the macro. Otherwise, register 'a' is used by default.

Command-parenright: **stop-kbd-macro** - Stop definition of a keyboard macro

Command-period: **debug-kill** - Stop debugging

Ctrl-=: **indent-to-match** - Indent the current line or selected region to match indentation of preceding non-blank line. Set toggle=True to indent instead of one level higher if already at the matching position.

Ctrl-Comma: **visit-history-previous** - Move back in history to previous visited editor position

Ctrl-Down: **forward-page** - Move cursor forward one page

Ctrl-F12: **command-by-name** - Execute given command by name, collecting any args as needed

Ctrl-ISO_Left_Tab: **begin-visited-document-cycle(move_back=False)** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-Left: **backward-word(delimiters="_`~!@#$%^&*()+-={}[]\|;:'",.<>/? trn")** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move backward one word

Ctrl-Minus: **zoom-out** - Action varies according to focus: *Document Viewer Commands*: Decrease documentation font size; *General Editor Commands*: Zoom out, increasing the text display size temporarily by one font size

Ctrl-Option-Delete: **backward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word behind of the cursor ; *Toolbar Search Commands*: Delete word behind the cursor

Ctrl-Option-Down: **next-line-extend-rect** - Move to next screen line, adjusting the rectangular selection range to new position, optionally repositioning character

within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Ctrl-Option-Left: **backward-char-extend-rect** - Move cursor backward one character, adjusting the rectangular selection range to new position

Ctrl-Option-Right: **forward-char-extend-rect** - Move cursor forward one character, adjusting the rectangular selection range to new position

Ctrl-Option-Up: **previous-line-extend-rect** - Move to previous screen line, adjusting the rectangular selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Ctrl-Period: **visit-history-next** - Move forward in history to next visited editor position

Ctrl-Plus: **zoom-in** - Action varies according to focus: *Document Viewer Commands*: Increase documentation font size; *General Editor Commands*: Zoom in, increasing the text display size temporarily by one font size

Ctrl-R: **query-replace** - Initiate incremental mini-search query/replace from the cursor position.

Ctrl-Return: **new-line** - Place a new line at the current cursor position

Ctrl-Right: **forward-word(delimiters="_`~!@#$%^&*()+-={}[]\|;:'",.<>/? trn")** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move forward one word

Ctrl-Shift-Left:
backward-word-extend(delimiters="_`~!@#$%^&*()+-={}[]\|;:'",.<>/? trn") - Action varies according to focus: *Active Editor Commands*: Move cursor backward one word, adjusting the selection range to new position. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move backward one word, extending the selection

Ctrl-Shift-Right:
forward-word-extend(delimiters="_`~!@#$%^&*()+-={}[]\|;:'",.<>/? trn") - Action varies according to focus: *Active Editor Commands*: Move cursor forward one word, adjusting the selection range to new position. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move forward one word, extending the selection

Ctrl-T: **forward-tab** - Action varies according to focus: *Active Editor Commands*: Place a tab character at the current cursor position ; *Search Manager Instance Commands*: Place a forward tab at the current cursor position in search or replace string

Ctrl-Tab: **begin-visited-document-cycle(move_back=True)** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-Up: **backward-page** - Move cursor backward one page

Ctrl-a: **beginning-of-line** - Action varies according to focus: *Active Editor Commands*: Move to beginning of current line. When toggle is True, moves to the end of the leading white space if already at the beginning of the line (and vice versa).; *Toolbar Search Commands*: Move to the beginning of the toolbar search entry

Ctrl-b: **backward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one character ; *Toolbar Search Commands*: Move backward one character

Ctrl-d: **forward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character in front of the cursor ; *Toolbar Search Commands*: Delete character in front of the cursor

Ctrl-e: **end-of-line** - Action varies according to focus: *Active Editor Commands*: Move to end of current line; *Toolbar Search Commands*: Move to the end of the toolbar search entry

Ctrl-f: **forward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one character ; *Toolbar Search Commands*: Move forward one character

Ctrl-h: **backward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character behind the cursor, or the current selection if not empty. ; *Toolbar Search Commands*: Delete character behind the cursor

Ctrl-k: **kill-line** - Kill rest of line from cursor to end of line, and place it into the clipboard with any other contiguously removed lines. End-of-line is removed only if there is nothing between the cursor and the end of the line.

Ctrl-n: **next-line** - Move to screen next line, optionally repositioning character within line: 'same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Ctrl-p: **previous-line** - Move to previous screen line, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Ctrl-space: **show-autocompleter** - Show the auto-completer for current cursor position

Ctrl-v: **forward-page** - Move cursor forward one page

Ctrl-y: **paste** - Action varies according to focus: *Active Editor Commands*: Paste text from clipboard ; *Search Manager Instance Commands*: Paste text from clipboard ; *Toolbar Search Commands*: Paste from clipboard

Delete: **forward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character in front of the cursor ; *Toolbar Search Commands*: Delete character in front of the cursor

Down: **next-line** - Move to screen next line, optionally repositioning character within line: 'same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

End: **scroll-to-end** - Scroll to the end of the text in the editor. Set move_caret to control whether the caret is moved.

F1: Multiple commands (first available is executed):

- **show-horizontal-tools** - Show the horizontal tool area
- **minimize-horizontal-tools** - Minimize the horizontal tool area

F11: **frame-up** - Move up the current debug stack

F12: **frame-down** - Move down the current debug stack

F2: Multiple commands (first available is executed):

- **show-vertical-tools** - Show the vertical tool area
- **minimize-vertical-tools** - Minimize the vertical tool area

F3: **search-forward** - Search again using the search manager's current settings in forward direction

F4: **goto-selected-symbol-defn** - Goto the definition of the selected source symbol, optionally showing the definition in another split if one is available and other_split is True.

F5: **debug-continue** - Continue (or start) running, to next breakpoint

F6: **step-over-statement** - Step over current statement

F7: **step-into** - Step into current execution point, or start debugging at first line

F8: **step-out** - Step out of the current function or method

F9: Multiple commands (first available is executed):

- **break-set** - Set a new regular breakpoint on current line
- **break-clear** - Clear the breakpoint on the current line

Home: **scroll-to-start** - Scroll to the top of the text in the editor. Set move_caret to control whether the the caret is moved.

ISO_Left_Tab: **backward-tab** - Outdent line at current position

Left: **backward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one character ; *Toolbar Search Commands*: Move backward one character

Option-Backspace: **backward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word behind of the cursor ; *Toolbar Search Commands*: Delete word behind the cursor

Option-Delete: **forward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word in front of the cursor ; *Toolbar Search Commands*: Delete word in front of the cursor

Option-F3: **search** - Bring up the search manager in search mode.

Option-F4: **close-window** - Close the current window and all documents and panels in it

Option-F6: **run-failed-tests** - Re-run all the previously failed tests. The tests are debugged when debug is True.

Option-F7: **run-last-tests** - Run again the last group of tests that were run. The tests are debugged when debug is True.

Option-Left: **backward-word** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move backward one word

Option-Page_Down: **forward-page** - Move cursor forward one page

Option-Page_Up: **backward-page** - Move cursor backward one page

Option-Return: **new-line** - Place a new line at the current cursor position

Option-Right: **forward-word** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move forward one word

Option-Shift-Left: **backward-word-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one word, adjusting the selection range to new position. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move backward one word, extending the selection

Option-Shift-Right: **forward-word-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one word, adjusting the selection range to new position. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move forward one word, extending the selection

Option-Up: **select-more** - Select more code on either the current line or larger multi-line blocks.

Page_Down: **forward-page** - Move cursor forward one page

Page_Up: **backward-page** - Move cursor backward one page

Return: **new-line** - Place a new line at the current cursor position

Right: **forward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one character ; *Toolbar Search Commands*: Move forward one character

Shift-Alt-Down: **next-line-extend(cursor="xcode")** - Move to next screen line, adjusting the selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, 'fnb' for first non-blank char, or 'xcode' to simulate XCode style Shift-Alt line selection.

Shift-Alt-Up: **previous-line-extend(cursor="xcode")** - Move to previous screen line, adjusting the selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, 'fnb' for first non-blank char, or 'xcode' to simulate XCode style Shift-Alt line selection.

Shift-Backspace: **backward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character behind the cursor, or the current selection if not empty. ; *Toolbar Search Commands*: Delete character behind the cursor

Shift-Command-F8: **start-select-block** - Turn on auto-select block mode

Shift-Delete: **forward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character in front of the cursor ; *Toolbar Search Commands*: Delete character in front of the cursor

Shift-Down: **next-line-extend** - Move to next screen line, adjusting the selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, 'fnb' for first non-blank char, or 'xcode' to simulate XCode style Shift-Alt line selection.

Shift-End: **end-of-document-extend** - Move cursor to end of document, adjusting the selection range to new position

Shift-F1: **move-focus** - Move the keyboard focus forward within the Window to the next editable area

Shift-F2: Multiple commands (first available is executed):

- **enter-fullscreen** - Hide both the vertical and horizontal tool areas and toolbar, saving previous state so it can be restored later with exit_fullscreen
- **exit-fullscreen** - Restore previous non-fullscreen state of all tools and tool bar

Shift-F3: **search-backward** - Search again using the search manager's current settings in backward direction

Shift-F4: **new-document-window** - Create a new document window with same documents and panels as in the current document window (if any; otherwise empty with default panels)

Shift-F5: **debug-file** - Start debugging the current file (rather than the main entry point)

Shift-F6: **run-all-tests** - Runs all the tests in testing panel.

Shift-F7: **run-current-tests** - Runs the current test or tests, if possible. The current tests are determined by the current position in the active view. The tests are debugged when debug is True.

Shift-F8: **start-select-char** - Turn on auto-select mode character by character

Shift-F9: Multiple commands (first available is executed):

- **break-enable** - Enable the breakpoint on the current line
- **break-disable** - Disable the breakpoint on current line

Shift-Home: **start-of-document-extend** - Move cursor to start of document, adjusting the selection range to new position

Shift-Left: **backward-char-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one character, adjusting the selection range to new position ; *Toolbar Search Commands*: Move backward one character, extending the selection

Shift-Page_Down: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Shift-Page_Up: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Shift-Return: **new-line** - Place a new line at the current cursor position

Shift-Right: **forward-char-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one character, adjusting the selection range to new position ; *Toolbar Search Commands*: Move forward one character, extending the selection

Shift-Tab: **backward-tab** - Outdent line at current position

Shift-Up: **previous-line-extend** - Move to previous screen line, adjusting the selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, 'fnb' for first non-blank char, or 'xcode' to simulate XCode style Shift-Alt line selection.

Tab: **tab-key** - Implement the tab key, the action of which is configurable by preference

Up: **previous-line** - Move to previous screen line, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Visual-Esc: **exit-visual-mode** - Exit visual mode and return back to default mode

21.6. Eclipse Personality

This section documents all the default key bindings for the **Eclipse** keyboard personality, set by the **Personality** preference.

Alt-1: **fold-python-methods** - Fold up all Python methods, expand all classes, and leave other fold points alone

Alt-1: **fold-python-methods** - Fold up all Python methods, expand all classes, and leave other fold points alone

Alt-2: **fold-python-classes** - Fold up all Python classes but leave other fold points alone

Alt-2: **fold-python-classes** - Fold up all Python classes but leave other fold points alone

Alt-3: **fold-python-classes-and-defs** - Fold up all Python classes, methods, and functions but leave other fold points alone

Alt-3: **fold-python-classes-and-defs** - Fold up all Python classes, methods, and functions but leave other fold points alone

Alt-BackSpace: **backward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word behind of the cursor ; *Toolbar Search Commands*: Delete word behind the cursor

Alt-BackSpace: **backward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word behind of the cursor ; *Toolbar Search Commands*: Delete word behind the cursor

Alt-Delete: **backward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word behind of the cursor ; *Toolbar Search Commands*: Delete word behind the cursor

Alt-Delete: **backward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word behind of the cursor ; *Toolbar Search Commands*: Delete word behind the cursor

Alt-Down: **fold-expand-more-current** - Expand the current fold point one more level

Alt-Down: **fold-expand-more-current** - Expand the current fold point one more level

Alt-Down: **move-line-down(indent=True)** - Move the current line or lines up down line, optionally indenting to match the new position

Alt-End: **fold-expand-all** - Expand all fold points in the current file

Alt-End: **fold-expand-all** - Expand all fold points in the current file

Alt-Enter: **view-file-properties** - View project properties for a particular file (current file if none is given)

Alt-F11: **prev-points-of-use-match** - Display the previous match in the active points of use tool

Alt-F11: **prev-points-of-use-match** - Display the previous match in the active points of use tool

Alt-F12: **next-points-of-use-match** - Display the next match in the active points of use tool

Alt-F12: **next-points-of-use-match** - Display the next match in the active points of use tool

Alt-F3: **search** - Bring up the search manager in search mode.

Alt-F3: **search** - Bring up the search manager in search mode.

Alt-F4: **close-window** - Close the current window and all documents and panels in it

Alt-F4: **close-window** - Close the current window and all documents and panels in it

Alt-F5: **run-to-cursor** - Run to current cursor position

Alt-F5: **run-to-cursor** - Run to current cursor position

Alt-F6: **run-failed-tests** - Re-run all the previously failed tests. The tests are debugged when debug is True.

Alt-F6: **run-failed-tests** - Re-run all the previously failed tests. The tests are debugged when debug is True.

Alt-F7: **run-last-tests** - Run again the last group of tests that were run. The tests are debugged when debug is True.

Alt-F7: **run-last-tests** - Run again the last group of tests that were run. The tests are debugged when debug is True.

Alt-Home: **fold-collapse-all** - Collapse all fold points in the current file

Alt-Home: **fold-collapse-all** - Collapse all fold points in the current file

Alt-Left: **visit-history-previous** - Move back in history to previous visited editor position

Alt-Left: **visit-history-previous** - Move back in history to previous visited editor position

Alt-Left: **visit-history-previous** - Move back in history to previous visited editor position

Alt-Page_Down: **fold-expand-all-current** - Expand the current fold point completely

Alt-Page_Down: **fold-expand-all-current** - Expand the current fold point completely

Alt-Page_Up: **fold-collapse-all-current** - Collapse the current fold point completely

Alt-Page_Up: **fold-collapse-all-current** - Collapse the current fold point completely

Alt-Return: **new-line** - Place a new line at the current cursor position

Alt-Return: **new-line** - Place a new line at the current cursor position

Alt-Right: **visit-history-next** - Move forward in history to next visited editor position

Alt-Right: **visit-history-next** - Move forward in history to next visited editor position

Alt-Right: **visit-history-next** - Move forward in history to next visited editor position

Alt-Shift-Down: **select-less** - Select less code; undoes the last select-more command

Alt-Shift-L: **introduce-variable** - Introduce named variable set to the current selected expression or to the range in the active editor specified by pos_range. The new_name argument is used as the default variable name if it is specified.

Alt-Shift-Left: **previous-statement** - Select the previous statement. Will ignore indented statements under the current statements unless ignore_indented is False. Specify a count of more than 1 to go back multiple statements.

Alt-Shift-M: **extract-def** - Extract selected lines to a new function or method. The new_name argument is used as the default for the name field if specified.

Alt-Shift-O: **show_preferences_gui(prefname="edit.highlight-occurrences")**

Alt-Shift-R: **rename-symbol** - Rename currently selected symbol. The new_name argument is used as the default for the name field if specified.

Alt-Shift-Right: **next-statement** - Select the next statement. Will ignore indented statements under the current statements unless ignore_indented is False. Specify a count of more than 1 to go forward multiple statements.

Alt-Shift-T: **show-panel(panel_type="refactoring")** - Show most recently visited panel instance of given type. If no such panel exists, add one to the primary window and show it. Returns the panel view object or None if not shown. Focus is shifted to panel if grab_focus is specified and is true; if grab_focus is not specified, it defaults to the value of flash.

The valid panel types are:

project (*) browser (**) batch-search (*) interactive-search source-assistant (**) debug-data debug-stack debug-io debug-exceptions debug-breakpoints (**) debug-probe (**) debug-watch (**) debug-modules (**) python-shell messages (*) help indent (**) bookmarks (**) testing (**) open-files (*) os-command (**) snippets (**) diff (**) uses (**) refactoring (**) versioncontrol.svn (**) versioncontrol.hg (**) versioncontrol.git (**) versioncontrol.bzr (**) versioncontrol.cvs (**) versioncontrol.perforce (**)

(*) Wing Personal and Pro only (**) Wing Pro only

Alt-Shift-U: **show_preferences_gui(prefname="edit.highlight-occurrences")**

Alt-Shift-Up: **select-more** - Select more code on either the current line or larger multi-line blocks.

Alt-Shift-V: **move-symbol** - Move the currently selected symbol to another module, class, or function. The new_filename and new_scope_name arguments are used as default values in the filename and scope name fields if specified.

Alt-Slash: **fold-toggle** - Toggle the current fold point

Alt-Slash: **fold-toggle** - Toggle the current fold point

Alt-Up: **fold-collapse-more-current** - Collapse the current fold point one more level

Alt-Up: **fold-collapse-more-current** - Collapse the current fold point one more level

Alt-Up: **move-line-up(indent=True)** - Move the current line or lines up one line, optionally indenting to match the new position

Alt-comma: **query-replace** - Initiate incremental mini-search query/replace from the cursor position.

Alt-period: **replace-string** - Replace all occurrences of a string from the cursor position to end of file.

BackSpace: **backward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character behind the cursor, or the current selection if not empty. ; *Toolbar Search Commands*: Delete character behind the cursor

BackSpace: **backward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character behind the cursor, or the current selection if not empty. ; *Toolbar Search Commands*: Delete character behind the cursor

Ctrl-/: **comment-block-toggle** - Toggle block comment (with ## at start) on the selected lines in editor. This is a different style of block commenting than Wing implements by default (the default in Wing is intended to work better with some of the other editor functionality)

Ctrl-0: **next-document** - Move to the next document alphabetically in the list of documents open in the current window

Ctrl-1: **activate-file-option-menu** - Activate the file menu for the editor.

Ctrl-1: **activate-file-option-menu** - Activate the file menu for the editor.

Ctrl-2: **activate-symbol-option-menu-1** - Activate the 1st symbol menu for the editor.

Ctrl-2: **activate-symbol-option-menu-1** - Activate the 1st symbol menu for the editor.

Ctrl-3: **activate-symbol-option-menu-2** - Activate the 2nd symbol menu for the editor.

Ctrl-3: **activate-symbol-option-menu-2** - Activate the 2nd symbol menu for the editor.

Ctrl-4: **activate-symbol-option-menu-3** - Activate the 3rd symbol menu for the editor.

Ctrl-4: **activate-symbol-option-menu-3** - Activate the 3rd symbol menu for the editor.

Ctrl-5: **activate-symbol-option-menu-4** - Activate the 4th symbol menu for the editor.

Ctrl-5: **activate-symbol-option-menu-4** - Activate the 4th symbol menu for the editor.

Ctrl-6: **activate-symbol-option-menu-5** - Activate the 5th symbol menu for the editor.

Ctrl-6: **activate-symbol-option-menu-5** - Activate the 5th symbol menu for the editor.

Ctrl-7 C: **use-lexer-cpp** - Force syntax highlighting for C/C++ source

Ctrl-7 H: **use-lexer-html** - Force syntax highlighting for HTML

Ctrl-7 M: **use-lexer-makefile** - Force syntax highlighting for make files

Ctrl-7 N: **use-lexer-none** - Use no syntax highlighting

Ctrl-7 P: **use-lexer-python** - Force syntax highlighting for Python source

Ctrl-7 S: **use-lexer-sql** - Force syntax highlighting for SQL

Ctrl-7 X: **use-lexer-xml** - Force syntax highlighting for XML files

Ctrl-8: **recent-document** - Switches to previous document most recently visited in the current window or window set if in one-window-per-editor windowing mode.

Ctrl-9: **previous-document** - Move to the previous document alphabetically in the list of documents open in the current window

Ctrl-=: **indent-to-match** - Indent the current line or selected region to match indentation of preceding non-blank line. Set toggle=True to indent instead of one level higher if already at the matching position.

Ctrl-=: **indent-to-match** - Indent the current line or selected region to match indentation of preceding non-blank line. Set toggle=True to indent instead of one level higher if already at the matching position.

Ctrl-A: **select-all** - Select all text in the editor

Ctrl-Alt-B: **search-sel-backward** - Search backward using current selection

Ctrl-Alt-Comma: **query-replace-regex** - Initiate incremental mini-search query/replace from the cursor position. The search string is treated as a regular expression.

Ctrl-Alt-D: **evaluate-sel-in-debug-probe** - Evaluate the current selection from the editor within the Debug Probe tool. When whole_lines is set, the selection is rounded to whole lines before evaluation. When unspecified (set to None), the setting from the Shell's Option menu is used instead.

Ctrl-Alt-D: **selection-add-next-occurence(skip_current=True)** - Add another selection containing the text of the current selection. If skip_current is true, the current selection will be deselected. If nothing is currently selected, select the current word. Searches backwards if reverse is true.

Ctrl-Alt-Down: **duplicate-line** - Duplicate the current line or lines. Places the duplicate on the line following the selection if pos is 'below' or before the selection if it is 'above'.

Ctrl-Alt-Down: **goto-next-bookmark(current_file_only=True)** - Go to the next bookmark, or the first one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-Alt-Down: **goto-next-bookmark(current_file_only=True)** - Go to the next bookmark, or the first one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-Alt-E: **evaluate-sel-in-shell** - Evaluate the current selection from the editor within the Python Shell tool, optionally restarting the shell first. When whole_lines is set, the selection is rounded to whole lines before evaluation. When unspecified (set to None), the setting from the Shell's Option menu is used instead.

Ctrl-Alt-F: **search-sel-forward** - Search forward using current selection

Ctrl-Alt-F6: **debug-failed-tests** - Re-run all the previously failed tests in the debugger.

Ctrl-Alt-F6: **debug-failed-tests** - Re-run all the previously failed tests in the debugger.

Ctrl-Alt-F7: **debug-last-tests** - Debug the last group of tests that were run.

Ctrl-Alt-F7: **debug-last-tests** - Debug the last group of tests that were run.

Ctrl-Alt-G: **goto-bookmark** - Goto named bookmark

Ctrl-Alt-K: **show-bookmarks** - Show a list of all currently defined bookmarks

Ctrl-Alt-Left: **goto-previous-bookmark** - Go to the previous bookmark in the bookmark list, or the last one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-Alt-Left: **goto-previous-bookmark** - Go to the previous bookmark in the bookmark list, or the last one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-Alt-M: **set-bookmark** - Set a bookmark at current location on the editor. Mark is the project-wide textual name of the bookmark.

Ctrl-Alt-Right: **goto-next-bookmark** - Go to the next bookmark, or the first one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-Alt-Right: **goto-next-bookmark** - Go to the next bookmark, or the first one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-Alt-T: **toggle-bookmark** - Set or remove a bookmark at current location on the editor. When set, the name of the bookmark is set to an auto-generated default.

Ctrl-Alt-Up: **duplicate-line-above** - Duplicate the current line or lines above the selection.

Ctrl-Alt-Up: **goto-previous-bookmark(current_file_only=True)** - Go to the previous bookmark in the bookmark list, or the last one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-Alt-Up: **goto-previous-bookmark(current_file_only=True)** - Go to the previous bookmark in the bookmark list, or the last one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-Alt-V: **evaluate-file-in-shell** - Run the contents of the editor within the Python Shell

Ctrl-Alt-period: **replace-string-regex** - Replace all occurrences of a string from the cursor position to end of file. The search string is treated as a regular expression.

Ctrl-Apostrophe: **enclose(start="'", end="'")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-Apostrophe: **enclose(start="'", end="'")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-Asterisk: **fold-expand-all** - Expand all fold points in the current file

Ctrl-B: **isearch-sel-forward** - Action varies according to focus: *Active Editor Commands*: Initiate incremental mini-search forward from the cursor position, using current selection as the search string. Set persist=False to do the search but end the interactive search session immediately.; *Document Viewer Commands*: Initiate incremental mini-search forward from the cursor position, using current selection as the search string. Set persist=False to do the search but end the interactive search session immediately.

Ctrl-BackSpace: **backward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word behind of the cursor ; *Toolbar Search Commands*: Delete word behind the cursor

Ctrl-BackSpace: **backward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word behind of the cursor ; *Toolbar Search Commands*: Delete word behind the cursor

Ctrl-Braceleft: **enclose(start="{", end="}")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-Braceleft: **enclose(start="{", end="}")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-Bracketleft: **enclose(start="[", end="]")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-Bracketleft: **enclose(start="[", end="]")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-C: **copy** - Action varies according to focus: *Active Editor Commands*: Copy selected text ; *Document Viewer Commands*: Copy any selected text. ; *Exceptions Commands*: Copy the exception traceback to the clipboard ; *Search Manager Instance Commands*: Copy selected text ; *Toolbar Search Commands*: Cut selection

Ctrl-Comma: **next-window** - Switch to the next window alphabetically by title

Ctrl-D: **delete-line** - Delete the current line or lines when the selection spans multiple lines or given repeat is > 1

Ctrl-D: **toolbar-search-focus** - Move focus to toolbar search entry.

Ctrl-Delete: **forward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word in front of the cursor ; *Toolbar Search Commands*: Delete word in front of the cursor

Ctrl-Delete: **forward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word in front of the cursor ; *Toolbar Search Commands*: Delete word in front of the cursor

Ctrl-Down: **select-less** - Select less code; undoes the last select-more command

Ctrl-Down: **select-less** - Select less code; undoes the last select-more command

Ctrl-E: **brace-match** - Match brace at current cursor position, selecting all text between the two and hilighting the braces

Ctrl-E: **show-panel(panel_type="open-files")** - Show most recently visited panel instance of given type. If no such panel exists, add one to the primary window and show it. Returns the panel view object or None if not shown. Focus is shifted to panel if grab_focus is specified and is true; if grab_focus is not specified, it defaults to the value of flash.

The valid panel types are:

project (*) browser (**) batch-search (*) interactive-search source-assistant (**) debug-data debug-stack debug-io debug-exceptions debug-breakpoints (**) debug-probe (**) debug-watch (**) debug-modules (**) python-shell messages (*) help indent (**) bookmarks (**) testing (**) open-files (*) os-command (**) snippets (**) diff (**) uses (**) refactoring (**) versioncontrol.svn (**) versioncontrol.hg (**) versioncontrol.git (**) versioncontrol.bzr (**) versioncontrol.cvs (**) versioncontrol.perforce (**)

(*) Wing Personal and Pro only (**) Wing Pro only

Ctrl-End: **end-of-document** - Move cursor to end of document

Ctrl-End: **end-of-document** - Move cursor to end of document

Ctrl-F: **search** - Bring up the search manager in search mode.

Ctrl-F12: **command-by-name** - Execute given command by name, collecting any args as needed

Ctrl-F12: **command-by-name** - Execute given command by name, collecting any args as needed

Ctrl-F3: **search-sel-forward** - Search forward using current selection

Ctrl-F3: **search-sel-forward** - Search forward using current selection

Ctrl-F4: **close** - Close active document. Abandon any changes when ignore_changes is True. Close empty windows when close_window is true and quit if all document windows closed when can_quit is true.

Ctrl-F4: **close** - Close active document. Abandon any changes when ignore_changes is True. Close empty windows when close_window is true and quit if all document windows closed when can_quit is true.

Ctrl-F5: **debug-kill** - Stop debugging

Ctrl-F5: **debug-kill** - Stop debugging

Ctrl-F5: **run-to-cursor** - Run to current cursor position

Ctrl-F6: **next-document** - Move to the next document alphabetically in the list of documents open in the current window

Ctrl-F6: **step-over** - Step over current instruction

Ctrl-F6: **step-over** - Step over current instruction

Ctrl-F8: **start-select-line** - Turn on auto-select mode line by line

Ctrl-F8: **start-select-line** - Turn on auto-select mode line by line

Ctrl-F9: **break-clear-all** - Clear all breakpoints

Ctrl-F9: **break-clear-all** - Clear all breakpoints

Ctrl-G: **goto-selected-symbol-defn** - Goto the definition of the selected source symbol, optionally showing the definition in another split if one is available and other_split is True.

Ctrl-G: **search-forward** - Search again using the search manager's current settings in forward direction

Ctrl-H: **batch-search** - Search on current selection using the Search in Files tool. The look_in argument gets entered in the look in field if not None or ''. The current selection is put into the search field if it doesn't span multiple lines and either

use_selection is true or there's nothing in the search field. The given search text is used instead, if provided

Ctrl-H: **replace** - Bring up the search manager in replace mode.

Ctrl-Home: **start-of-document** - Move cursor to start of document

Ctrl-Home: **start-of-document** - Move cursor to start of document

Ctrl-I: **replace-and-search** - Replace current selection and search again.

Ctrl-Insert: **copy** - Action varies according to focus: *Active Editor Commands*: Copy selected text ; *Document Viewer Commands*: Copy any selected text. ; *Exceptions Commands*: Copy the exception traceback to the clipboard ; *Search Manager Instance Commands*: Copy selected text ; *Toolbar Search Commands*: Cut selection

Ctrl-Insert: **copy** - Action varies according to focus: *Active Editor Commands*: Copy selected text ; *Document Viewer Commands*: Copy any selected text. ; *Exceptions Commands*: Copy the exception traceback to the clipboard ; *Search Manager Instance Commands*: Copy selected text ; *Toolbar Search Commands*: Cut selection

Ctrl-J: **fill-paragraph** - Attempt to auto-justify the paragraph around the current start of selection

Ctrl-J: **isearch-forward** - Action varies according to focus: *Active Editor Commands*: Initiate incremental mini-search forward from the cursor position, optionally entering the given search string ; *Document Viewer Commands*: Initiate incremental mini-search forward from the cursor position, optionally entering the given search string.

Ctrl-K: **open-from-keyboard** - Open a file from disk using keyboard-driven selection of the file

Ctrl-K: **search-forward** - Search again using the search manager's current settings in forward direction

Ctrl-L: **goto-line** - Position cursor at start of given line number

Ctrl-L: **goto-line** - Position cursor at start of given line number

Ctrl-Left: **backward-word** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move backward one word

Ctrl-Left: **backward-word** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may

be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move backward one word

Ctrl-Less: **enclose(start="<", end=">")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-Less: **enclose(start="<", end=">")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-M: Multiple commands (first available is executed):

- **enter-fullscreen** - Hide both the vertical and horizontal tool areas and toolbar, saving previous state so it can be restored later with exit_fullscreen
- **exit-fullscreen** - Restore previous non-fullscreen state of all tools and tool bar

Ctrl-M: **execute-kbd-macro** - Execute most recently recorded keyboard macro. If register is None then the user is asked to enter a letter a-z for the register where the macro is filed. Otherwise, register 'a' is used by default.

Ctrl-Minus: **fold-collapse-current** - Collapse the current fold point

Ctrl-Minus: **zoom-out** - Action varies according to focus: *Document Viewer Commands*: Decrease documentation font size; *General Editor Commands*: Zoom out, increasing the text display size temporarily by one font size

Ctrl-Minus: **zoom-out** - Action varies according to focus: *Document Viewer Commands*: Decrease documentation font size; *General Editor Commands*: Zoom out, increasing the text display size temporarily by one font size

Ctrl-N: **new-file** - Create a new file

Ctrl-Next: **forward-page** - Move cursor forward one page

Ctrl-Next: **forward-page** - Move cursor forward one page

Ctrl-O: **find-symbol** - Allow user to visit point of definition of a source symbol in the current editor context by typing a fragment of the name

Ctrl-O: **open-gui** - Open a file from disk, prompting with file selection dialog if necessary

Ctrl-P: **print-view** - Print active editor document

Ctrl-Page_Down: **next-document** - Move to the next document alphabetically in the list of documents open in the current window

Ctrl-Page_Down: **next-document** - Move to the next document alphabetically in the list of documents open in the current window

Ctrl-Page_Up: **previous-document** - Move to the previous document alphabetically in the list of documents open in the current window

Ctrl-Page_Up: **previous-document** - Move to the previous document alphabetically in the list of documents open in the current window

Ctrl-Parenleft: **enclose(start="(", end=")")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-Parenleft: **enclose(start="(", end=")")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-Period: **comment-toggle** - Toggle commenting out of the selected lines. The style of commenting can be controlled with the style argument: 'indented' uses the default comment style indented at end of leading white space and 'block' uses a block comment in column zero. If not given, the style configured with the Editor / Block Comment Style preference is used.

Ctrl-Plus: **fold-expand-current** - Expand the current fold point

Ctrl-Plus: **zoom-in** - Action varies according to focus: *Document Viewer Commands*: Increase documentation font size; *General Editor Commands*: Zoom in, increasing the text display size temporarily by one font size

Ctrl-Plus: **zoom-in** - Action varies according to focus: *Document Viewer Commands*: Increase documentation font size; *General Editor Commands*: Zoom in, increasing the text display size temporarily by one font size

Ctrl-Pointer_Button1: **goto-clicked-symbol-defn** - Goto the definition of the source symbol that was last clicked on, optionally showing the definition in another split if one is available and other_split is True.

Ctrl-Pointer_Button1: **goto-clicked-symbol-defn** - Goto the definition of the source symbol that was last clicked on, optionally showing the definition in another split if one is available and other_split is True.

Ctrl-Prior: **backward-page** - Move cursor backward one page

Ctrl-Prior: **backward-page** - Move cursor backward one page

Ctrl-Q: **quit** - Quit the application.

Ctrl-Q: **visit-history-previous** - Move back in history to previous visited editor position

Ctrl-Quotedbl: **enclose(start='"', end='"')** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-Quotedbl: **enclose(start='"', end='")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-Quoteleft: **begin-visited-document-cycle(move_back=True, back_key="Ctrl-Quoteleft", forward_key="Ctrl-AsciiTilde")** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-Quoteleft: **begin-visited-document-cycle(move_back=True, back_key="Ctrl-Quoteleft", forward_key="Ctrl-AsciiTilde")** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-R: **replace** - Bring up the search manager in replace mode.

Ctrl-R: **run-to-cursor** - Run to current cursor position

Ctrl-Return: **new-line-after** - Place a new line after the current line

Ctrl-Return: **new-line-after** - Place a new line after the current line

Ctrl-Right: **forward-word** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move forward one word

Ctrl-Right: **forward-word** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move forward one word

Ctrl-S: **save** - Save active document. Also close it if close is True.

Ctrl-Shift-B: Multiple commands (first available is executed):

- **break-set** - Set a new regular breakpoint on current line
- **break-clear** - Clear the breakpoint on the current line

Ctrl-Shift-B: **isearch-sel-backward** - Initiate incremental mini-search backward from the cursor position, using current selection as the search string. Set persist=False to do the search but end the interactive search session immediately.

Ctrl-Shift-C: **comment-block-toggle** - Toggle block comment (with ## at start) on the selected lines in editor. This is a different style of block commenting than Wing implements by default (the default in Wing is intended to work better with some of the other editor functionality)

Ctrl-Shift-C: **delete-line** - Delete the current line or lines when the selection spans multiple lines or given repeat is > 1

Ctrl-Shift-D: **selection-add-next-occurence** - Add another selection containing the text of the current selection. If skip_current is true, the current selection will be deselected. If nothing is currently selected, select the current word. Searches backwards if reverse is true.

Ctrl-Shift-Delete: **delete-lines**

Ctrl-Shift-Delete: **delete-lines**

Ctrl-Shift-Down: **move-line-down** - Move the current line or lines up down line, optionally indenting to match the new position

Ctrl-Shift-Down: **next-scope** - Select the next scope. Specify a count of more than 1 to go forward multiple scopes. If sibling_only is true, move only to other scopes of the same parent.

Ctrl-Shift-Down: **scroll-text-down** - Scroll text down a line w/o moving cursor's relative position on screen. Repeat is number of lines or if >0 and <1.0 then percent of screen. Set move_cursor to False to leave cursor in current position within the source, otherwise it is moved so the cursor remains on same screen line.

Ctrl-Shift-Down: **scroll-text-down** - Scroll text down a line w/o moving cursor's relative position on screen. Repeat is number of lines or if >0 and <1.0 then percent of screen. Set move_cursor to False to leave cursor in current position within the source, otherwise it is moved so the cursor remains on same screen line.

Ctrl-Shift-E: **focus-current-editor** - Move focus back to the current editor, out of any tool, if there is an active editor.

Ctrl-Shift-End: **end-of-document-extend** - Move cursor to end of document, adjusting the selection range to new position

Ctrl-Shift-End: **end-of-document-extend** - Move cursor to end of document, adjusting the selection range to new position

Ctrl-Shift-Enter: **new-line-before** - Place a new line before the current line

Ctrl-Shift-F: **batch-search** - Search on current selection using the Search in Files tool. The look_in argument gets entered in the look in field if not None or ''. The current selection is put into the search field if it doesn't span multiple lines and either use_selection is true or there's nothing in the search field. The given search text is used instead, if provided

Ctrl-Shift-F: **fill-paragraph** - Attempt to auto-justify the paragraph around the current start of selection

Ctrl-Shift-F2: **close-all** - Close all documents in the current window, or in all windows if in one-window-per-editor windowing policy. Leave currently visible documents (or active window in one-window-per-editor-mode) if omit_current is

True. Abandons changes rather than saving them when ignore_changes is True. Close empty window and quit if all document windows closed when close_window is True.

Ctrl-Shift-F3: **search-sel-backward** - Search backward using current selection

Ctrl-Shift-F3: **search-sel-backward** - Search backward using current selection

Ctrl-Shift-F4: **close-all** - Close all documents in the current window, or in all windows if in one-window-per-editor windowing policy. Leave currently visible documents (or active window in one-window-per-editor-mode) if omit_current is True. Abandons changes rather than saving them when ignore_changes is True. Close empty window and quit if all document windows closed when close_window is True.

Ctrl-Shift-F5: **debug-stop** - Pause free-running execution at current program counter

Ctrl-Shift-F5: **debug-stop** - Pause free-running execution at current program counter

Ctrl-Shift-F6: **debug-all-tests** - Debug all the tests in testing panel.

Ctrl-Shift-F6: **debug-all-tests** - Debug all the tests in testing panel.

Ctrl-Shift-F7: **debug-current-tests** - Runs the current test or tests, if possible. The current tests are determined by the current position in the active view.

Ctrl-Shift-F7: **debug-current-tests** - Runs the current test or tests, if possible. The current tests are determined by the current position in the active view.

Ctrl-Shift-F9: Multiple commands (first available is executed):

- **break-disable-all** - Disable all breakpoints
- **break-enable-all** - Enable all breakpoints

Ctrl-Shift-F9: Multiple commands (first available is executed):

- **break-disable-all** - Disable all breakpoints
- **break-enable-all** - Enable all breakpoints

Ctrl-Shift-G: **find-points-of-use** - Find points of use for a symbol. The symbol defaults to the active selection.

Ctrl-Shift-G: **search-backward** - Search again using the search manager's current settings in backward direction

Ctrl-Shift-H: **batch-replace** - Display search and replace in files tool.

Ctrl-Shift-Home: **start-of-document-extend** - Move cursor to start of document, adjusting the selection range to new position

Ctrl-Shift-Home: **start-of-document-extend** - Move cursor to start of document, adjusting the selection range to new position

Ctrl-Shift-I: **add-current-file-to-project** - Add the frontmost currently open file to project

Ctrl-Shift-I: **add-current-file-to-project** - Add the frontmost currently open file to project

Ctrl-Shift-I: **debug-stop** - Pause free-running execution at current program counter

Ctrl-Shift-ISO_Left_Tab: **begin-visited-document-cycle(move_back=False)** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-Shift-ISO_Left_Tab: **begin-visited-document-cycle(move_back=False)** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-Shift-Insert: **toggle-overtype** - Toggle status of overtyping mode

Ctrl-Shift-J: **isearch-backward** - Action varies according to focus: *Active Editor Commands*: Initiate incremental mini-search backward from the cursor position, optionally entering the given search string ; *Document Viewer Commands*: Initiate incremental mini-search backward from the cursor position, optionally entering the given search string.

Ctrl-Shift-K: **search-backward** - Search again using the search manager's current settings in backward direction

Ctrl-Shift-L: **swap-lines** - Swap the line at start of current selection with the line that follows it, or the preceding line if previous is True.

Ctrl-Shift-Left: **backward-word-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one word, adjusting the selection range to new position. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move backward one word, extending the selection

Ctrl-Shift-Left: **backward-word-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one word, adjusting the selection range to new position. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move backward one word, extending the selection

Ctrl-Shift-Next: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Ctrl-Shift-Next: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Ctrl-Shift-O: **open-from-project** - Open document from the project via the Open From Project dialog. The given fragment is used as the initial fragment filter and if it is None, the selected text or the symbol under the cursor is used. If skip_if_unique is true, the file is opened without the dialog being displayed if only one filename matches the fragment.

Ctrl-Shift-P: **brace-match** - Match brace at current cursor position, selecting all text between the two and hilighting the braces

Ctrl-Shift-Page_Down: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Ctrl-Shift-Page_Down: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Ctrl-Shift-Page_Up: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Ctrl-Shift-Page_Up: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Ctrl-Shift-Prior: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Ctrl-Shift-Prior: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Ctrl-Shift-R: **batch-replace** - Display search and replace in files tool.

Ctrl-Shift-R: **open-from-project** - Open document from the project via the Open From Project dialog. The given fragment is used as the initial fragment filter and if it is None, the selected text or the symbol under the cursor is used. If skip_if_unique is true, the file is opened without the dialog being displayed if only one filename matches the fragment.

Ctrl-Shift-Right: **forward-word-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one word, adjusting the selection range to new position. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move forward one word, extending the selection

Ctrl-Shift-Right: **forward-word-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one word, adjusting the selection range to new position. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move forward one word, extending the selection

Ctrl-Shift-S: **save-all** - Save all unsaved items, prompting for names for any new items that don't have a filename already.

Ctrl-Shift-S: **save-as** - Save active document to a new file

Ctrl-Shift-Space: **show-panel(panel_type="source-assistant")** - Show most recently visited panel instance of given type. If no such panel exists, add one to the primary window and show it. Returns the panel view object or None if not shown. Focus is shifted to panel if grab_focus is specified and is true; if grab_focus is not specified, it defaults to the value of flash.

The valid panel types are:

project (*) browser (**) batch-search (*) interactive-search source-assistant (**) debug-data debug-stack debug-io debug-exceptions debug-breakpoints (**) debug-probe (**) debug-watch (**) debug-modules (**) python-shell messages (*) help indent (**) bookmarks (**) testing (**) open-files (*) os-command (**) snippets (**) diff (**) uses (**) refactoring (**) versioncontrol.svn (**) versioncontrol.hg (**) versioncontrol.git (**) versioncontrol.bzr (**) versioncontrol.cvs (**) versioncontrol.perforce (**)

(*) Wing Personal and Pro only (**) Wing Pro only

Ctrl-Shift-T: **find-symbol** - Allow user to visit point of definition of a source symbol in the current editor context by typing a fragment of the name

Ctrl-Shift-T: **find-symbol** - Allow user to visit point of definition of a source symbol in the current editor context by typing a fragment of the name

Ctrl-Shift-Tab: **begin-visited-document-cycle(move_back=False)** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-Shift-Tab: **begin-visited-document-cycle(move_back=False)** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-Shift-U: **batch-search(look_in="Current File")** - Search on current selection using the Search in Files tool. The look_in argument gets entered in the look in field if not None or ''. The current selection is put into the search field if it doesn't span multiple lines and either use_selection is true or there's nothing in the search field. The given search text is used instead, if provided

Ctrl-Shift-U: **isearch-backward** - Action varies according to focus: *Active Editor Commands*: Initiate incremental mini-search backward from the cursor position, optionally entering the given search string ; *Document Viewer Commands*: Initiate incremental mini-search backward from the cursor position, optionally entering the given search string.

Ctrl-Shift-Up: **move-line-up** - Move the current line or lines up one line, optionally indenting to match the new position

Ctrl-Shift-Up: **previous-scope** - Select the previous scope. Specify a count of more than 1 to go backward multiple scopes. If sibling_only is true, move only to other scopes of the same parent.

Ctrl-Shift-Up: **scroll-text-up** - Scroll text up a line w/o moving cursor's relative position on screen. Repeat is number of lines or if >0 and <1.0 then percent of screen. Set move_cursor to False to leave cursor in current position within the source, otherwise it is moved so the cursor remains on same screen line.

Ctrl-Shift-Up: **scroll-text-up** - Scroll text up a line w/o moving cursor's relative position on screen. Repeat is number of lines or if >0 and <1.0 then percent of screen. Set move_cursor to False to leave cursor in current position within the source, otherwise it is moved so the cursor remains on same screen line.

Ctrl-Shift-V: **duplicate-line** - Duplicate the current line or lines. Places the duplicate on the line following the selection if pos is 'below' or before the selection if it is 'above'.

Ctrl-Shift-W: **close-all** - Close all documents in the current window, or in all windows if in one-window-per-editor windowing policy. Leave currently visible documents (or active window in one-window-per-editor-mode) if omit_current is True. Abandons changes rather than saving them when ignore_changes is True. Close empty window and quit if all document windows closed when close_window is True.

Ctrl-Shift-X: **lower-case** - Change current selection to all lower case

Ctrl-Shift-Y: **duplicate-line-above** - Duplicate the current line or lines above the selection.

Ctrl-Shift-Y: **upper-case** - Change current selection to all upper case

Ctrl-Slash: **comment-out-region** - Comment out the selected region. The style of commenting can be controlled with the style argument: 'indented' uses the default comment style indented at end of leading white space and 'block' uses a block comment in column zero. If not given, the style configured with the Editor / Block Comment Style preference is used. Each call adds a level of commenting.

Ctrl-Slash: **fold-toggle** - Toggle the current fold point

Ctrl-T: **forward-tab** - Action varies according to focus: *Active Editor Commands*: Place a tab character at the current cursor position ; *Search Manager Instance Commands*: Place a forward tab at the current cursor position in search or replace string

Ctrl-T: **forward-tab** - Action varies according to focus: *Active Editor Commands*: Place a tab character at the current cursor position ; *Search Manager Instance Commands*: Place a forward tab at the current cursor position in search or replace string

Ctrl-Tab: **begin-visited-document-cycle(move_back=True)** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-Tab: **begin-visited-document-cycle(move_back=True)** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-U: **execute-file** - Execute the file at the given location or use the active view if loc is None.

Ctrl-U: **isearch-forward** - Action varies according to focus: *Active Editor Commands*: Initiate incremental mini-search forward from the cursor position, optionally entering the given search string ; *Document Viewer Commands*: Initiate incremental mini-search forward from the cursor position, optionally entering the given search string.

Ctrl-Up: **select-more** - Select more code on either the current line or larger multi-line blocks.

Ctrl-Up: **select-more** - Select more code on either the current line or larger multi-line blocks.

Ctrl-V: **paste** - Action varies according to focus: *Active Editor Commands*: Paste text from clipboard ; *Search Manager Instance Commands*: Paste text from clipboard ; *Toolbar Search Commands*: Paste from clipboard

Ctrl-W: **close** - Close active document. Abandon any changes when ignore_changes is True. Close empty windows when close_window is true and quit if all document windows closed when can_quit is true.

Ctrl-X: **cut** - Action varies according to focus: *Active Editor Commands*: Cut selected text ; *Search Manager Instance Commands*: Cut selected text ; *Toolbar Search Commands*: Cut selection

Ctrl-Y: **redo** - Redo last action

Ctrl-Z: **undo** - Undo last action

Ctrl-: **uncomment-out-region** - Uncomment out the selected region if commented out. If one_level is True then each call removes only one level of commenting.

Ctrl-]: **brace-match** - Match brace at current cursor position, selecting all text between the two and hilighting the braces

Ctrl-greater: **indent-region** - Indent the selected region one level of indentation. Set sel to None to use preference to determine selection behavior, or "never-select" to unselect after indent, "always-select" to always select after indent, or "retain-select" to retain current selection after indent.

Ctrl-less: **outdent-region** - Outdent the selected region one level of indentation. Set sel to None to use preference to determine selection behavior, or "never-select" to unselect after indent, "always-select" to always select after indent, or "retain-select" to retain current selection after indent.

Ctrl-parenleft: **start-kbd-macro** - Start definition of a keyboard macro. If register=None then the user is prompted to enter a letter a-z under which to file the macro. Otherwise, register 'a' is used by default.

Ctrl-parenright: **stop-kbd-macro** - Stop definition of a keyboard macro

Ctrl-question: **uncomment-out-region** - Uncomment out the selected region if commented out. If one_level is True then each call removes only one level of commenting.

Ctrl-space: **show-autocompleter** - Show the auto-completer for current cursor position

Ctrl-|: **indent-lines(lines=1)** - Indent selected number of lines from cursor position. Set lines to None to indent all the lines in current selection. Set levels to indent more than one level at a time.

Delete: **forward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character in front of the cursor ; *Toolbar Search Commands*: Delete character in front of the cursor

Delete: **forward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character in front of the cursor ; *Toolbar Search Commands*: Delete character in front of the cursor

Down: **next-line** - Move to screen next line, optionally repositioning character within line: 'same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Down: **next-line** - Move to screen next line, optionally repositioning character within line: 'same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

End: **end-of-line** - Action varies according to focus: *Active Editor Commands*: Move to end of current line; *Toolbar Search Commands*: Move to the end of the toolbar search entry

End: **end-of-line** - Action varies according to focus: *Active Editor Commands*: Move to end of current line; *Toolbar Search Commands*: Move to the end of the toolbar search entry

F1: Multiple commands (first available is executed):

- **show-horizontal-tools** - Show the horizontal tool area
- **minimize-horizontal-tools** - Minimize the horizontal tool area

F1: Multiple commands (first available is executed):

- **show-horizontal-tools** - Show the horizontal tool area
- **minimize-horizontal-tools** - Minimize the horizontal tool area

F11: **debug-continue** - Continue (or start) running, to next breakpoint

F11: **frame-up** - Move up the current debug stack

F11: **frame-up** - Move up the current debug stack

F12: **focus-current-editor** - Move focus back to the current editor, out of any tool, if there is an active editor.

F12: **frame-down** - Move down the current debug stack

F12: **frame-down** - Move down the current debug stack

F2: Multiple commands (first available is executed):

- **show-vertical-tools** - Show the vertical tool area
- **minimize-vertical-tools** - Minimize the vertical tool area

F2: Multiple commands (first available is executed):

- **show-vertical-tools** - Show the vertical tool area
- **minimize-vertical-tools** - Minimize the vertical tool area

F3: **goto-selected-symbol-defn** - Goto the definition of the selected source symbol, optionally showing the definition in another split if one is available and other_split is True.

F3: **search-forward** - Search again using the search manager's current settings in forward direction

F3: **search-forward** - Search again using the search manager's current settings in forward direction

F4: **goto-selected-symbol-defn** - Goto the definition of the selected source symbol, optionally showing the definition in another split if one is available and other_split is True.

F4: **goto-selected-symbol-defn** - Goto the definition of the selected source symbol, optionally showing the definition in another split if one is available and other_split is True.

F4: **show-panel(panel_type="browser")** - Show most recently visited panel instance of given type. If no such panel exists, add one to the primary window and show it. Returns the panel view object or None if not shown. Focus is shifted to

panel if grab_focus is specified and is true; if grab_focus is not specified, it defaults to the value of flash.

The valid panel types are:

project (*) browser (**) batch-search (*) interactive-search source-assistant (**) debug-data debug-stack debug-io debug-exceptions debug-breakpoints (**) debug-probe (**) debug-watch (**) debug-modules (**) python-shell messages (*) help indent (**) bookmarks (**) testing (**) open-files (*) os-command (**) snippets (**) diff (**) uses (**) refactoring (**) versioncontrol.svn (**) versioncontrol.hg (**) versioncontrol.git (**) versioncontrol.bzr (**) versioncontrol.cvs (**) versioncontrol.perforce (**)

(*) Wing Personal and Pro only (**) Wing Pro only

F5: **debug-continue** - Continue (or start) running, to next breakpoint

F5: **debug-continue** - Continue (or start) running, to next breakpoint

F5: **step-into** - Step into current execution point, or start debugging at first line

F6: **step-over-statement** - Step over current statement

F6: **step-over-statement** - Step over current statement

F6: **step-over-statement** - Step over current statement

F7: **step-into** - Step into current execution point, or start debugging at first line

F7: **step-into** - Step into current execution point, or start debugging at first line

F7: **step-out** - Step out of the current function or method

F8: **debug-continue** - Continue (or start) running, to next breakpoint

F8: **step-out** - Step out of the current function or method

F8: **step-out** - Step out of the current function or method

F9: Multiple commands (first available is executed):

- **break-set** - Set a new regular breakpoint on current line
- **break-clear** - Clear the breakpoint on the current line

F9: Multiple commands (first available is executed):

- **break-set** - Set a new regular breakpoint on current line
- **break-clear** - Clear the breakpoint on the current line

Home: **beginning-of-line-text** - Move to end of the leading white space, if any, on the current line. If toggle is True, moves to the beginning of the line if already at the end of the leading white space (and vice versa).

Home: **beginning-of-line-text** - Move to end of the leading white space, if any, on the current line. If toggle is True, moves to the beginning of the line if already at the end of the leading white space (and vice versa).

ISO_Left_Tab: **backward-tab** - Outdent line at current position

ISO_Left_Tab: **backward-tab** - Outdent line at current position

Insert: **toggle-overtype** - Toggle status of overtyping mode

Insert: **toggle-overtype** - Toggle status of overtyping mode

Left: **backward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one character ; *Toolbar Search Commands*: Move backward one character

Left: **backward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one character ; *Toolbar Search Commands*: Move backward one character

Next: **forward-page** - Move cursor forward one page

Next: **forward-page** - Move cursor forward one page

Page_Down: **forward-page** - Move cursor forward one page

Page_Down: **forward-page** - Move cursor forward one page

Page_Up: **backward-page** - Move cursor backward one page

Page_Up: **backward-page** - Move cursor backward one page

Prior: **backward-page** - Move cursor backward one page

Prior: **backward-page** - Move cursor backward one page

Return: **new-line** - Place a new line at the current cursor position

Return: **new-line** - Place a new line at the current cursor position

Right: **forward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one character ; *Toolbar Search Commands*: Move forward one character

Right: **forward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one character ; *Toolbar Search Commands*: Move forward one character

Shift-Alt-A: **diff-merge-a-b**

Shift-Alt-A: **diff-merge-a-b**

Shift-Alt-B: **diff-merge-b-a**

Shift-Alt-B: **diff-merge-b-a**

Shift-Alt-Down: **next-line-extend-rect** - Move to next screen line, adjusting the rectangular selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Shift-Alt-Down: **next-line-extend-rect** - Move to next screen line, adjusting the rectangular selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Shift-Alt-Left: **backward-char-extend-rect** - Move cursor backward one character, adjusting the rectangular selection range to new position

Shift-Alt-Left: **backward-char-extend-rect** - Move cursor backward one character, adjusting the rectangular selection range to new position

Shift-Alt-N: **diff-next**

Shift-Alt-N: **diff-next**

Shift-Alt-P: **diff-previous**

Shift-Alt-P: **diff-previous**

Shift-Alt-Right: **forward-char-extend-rect** - Move cursor forward one character, adjusting the rectangular selection range to new position

Shift-Alt-Right: **forward-char-extend-rect** - Move cursor forward one character, adjusting the rectangular selection range to new position

Shift-Alt-Up: **previous-line-extend-rect** - Move to previous screen line, adjusting the rectangular selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Shift-Alt-Up: **previous-line-extend-rect** - Move to previous screen line, adjusting the rectangular selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Shift-BackSpace: **backward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character behind the cursor, or the current selection if not empty. ; *Toolbar Search Commands*: Delete character behind the cursor

Shift-BackSpace: **backward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character behind the cursor, or the current selection if not empty. ; *Toolbar Search Commands*: Delete character behind the cursor

Shift-Ctrl-F8: **start-select-block** - Turn on auto-select block mode

Shift-Ctrl-F8: **start-select-block** - Turn on auto-select block mode

Shift-Delete: **cut** - Action varies according to focus: *Active Editor Commands*: Cut selected text ; *Search Manager Instance Commands*: Cut selected text ; *Toolbar Search Commands*: Cut selection

Shift-Delete: **cut** - Action varies according to focus: *Active Editor Commands*: Cut selected text ; *Search Manager Instance Commands*: Cut selected text ; *Toolbar Search Commands*: Cut selection

Shift-Down: **next-line-extend** - Move to next screen line, adjusting the selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, 'fnb' for first non-blank char, or 'xcode' to simulate XCode style Shift-Alt line selection.

Shift-Down: **next-line-extend** - Move to next screen line, adjusting the selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, 'fnb' for first non-blank char, or 'xcode' to simulate XCode style Shift-Alt line selection.

Shift-End: **end-of-line-extend** - Action varies according to focus: *Active Editor Commands*: Move to end of current line, adjusting the selection range to new position ; *Toolbar Search Commands*: Move to the end of the toolbar search entry, extending the selection

Shift-End: **end-of-line-extend** - Action varies according to focus: *Active Editor Commands*: Move to end of current line, adjusting the selection range to new position ; *Toolbar Search Commands*: Move to the end of the toolbar search entry, extending the selection

Shift-Enter: **new-line-after** - Place a new line after the current line

Shift-F1: **move-focus** - Move the keyboard focus forward within the Window to the next editable area

Shift-F1: **move-focus** - Move the keyboard focus forward within the Window to the next editable area

Shift-F11: **frame-show** - Show the position (thread and stack frame) where the debugger originally stopped

Shift-F11: **frame-show** - Show the position (thread and stack frame) where the debugger originally stopped

Shift-F2: Multiple commands (first available is executed):

- **enter-fullscreen** - Hide both the vertical and horizontal tool areas and toolbar, saving previous state so it can be restored later with exit_fullscreen

- **exit-fullscreen** - Restore previous non-fullscreen state of all tools and tool bar

Shift-F2: Multiple commands (first available is executed):

- **enter-fullscreen** - Hide both the vertical and horizontal tool areas and toolbar, saving previous state so it can be restored later with exit_fullscreen
- **exit-fullscreen** - Restore previous non-fullscreen state of all tools and tool bar

Shift-F3: **search-backward** - Search again using the search manager's current settings in backward direction

Shift-F3: **search-backward** - Search again using the search manager's current settings in backward direction

Shift-F4: **find-points-of-use** - Find points of use for a symbol. The symbol defaults to the active selection.

Shift-F4: **find-points-of-use** - Find points of use for a symbol. The symbol defaults to the active selection.

Shift-F5: **debug-file** - Start debugging the current file (rather than the main entry point)

Shift-F5: **debug-file** - Start debugging the current file (rather than the main entry point)

Shift-F6: **run-all-tests** - Runs all the tests in testing panel.

Shift-F6: **run-all-tests** - Runs all the tests in testing panel.

Shift-F7: **run-current-tests** - Runs the current test or tests, if possible. The current tests are determined by the current position in the active view. The tests are debugged when debug is True.

Shift-F7: **run-current-tests** - Runs the current test or tests, if possible. The current tests are determined by the current position in the active view. The tests are debugged when debug is True.

Shift-F8: **start-select-char** - Turn on auto-select mode character by character

Shift-F8: **start-select-char** - Turn on auto-select mode character by character

Shift-F9: Multiple commands (first available is executed):

- **break-enable** - Enable the breakpoint on the current line
- **break-disable** - Disable the breakpoint on current line

Shift-F9: Multiple commands (first available is executed):

- **break-enable** - Enable the breakpoint on the current line
- **break-disable** - Disable the breakpoint on current line

Shift-Home: **beginning-of-line-text-extend** - Move to end of the leading white space, if any, on the current line, adjusting the selection range to the new position. If toggle is True, moves to the beginning of the line if already at the end of the leading white space (and vice versa).

Shift-Home: **beginning-of-line-text-extend** - Move to end of the leading white space, if any, on the current line, adjusting the selection range to the new position. If toggle is True, moves to the beginning of the line if already at the end of the leading white space (and vice versa).

Shift-Insert: **paste** - Action varies according to focus: *Active Editor Commands*: Paste text from clipboard ; *Search Manager Instance Commands*: Paste text from clipboard ; *Toolbar Search Commands*: Paste from clipboard

Shift-Insert: **paste** - Action varies according to focus: *Active Editor Commands*: Paste text from clipboard ; *Search Manager Instance Commands*: Paste text from clipboard ; *Toolbar Search Commands*: Paste from clipboard

Shift-Left: **backward-char-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one character, adjusting the selection range to new position ; *Toolbar Search Commands*: Move backward one character, extending the selection

Shift-Left: **backward-char-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one character, adjusting the selection range to new position ; *Toolbar Search Commands*: Move backward one character, extending the selection

Shift-Next: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Shift-Next: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Shift-Page_Down: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Shift-Page_Down: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Shift-Page_Up: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Shift-Page_Up: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Shift-Prior: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Shift-Prior: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Shift-Return: **new-line-before** - Place a new line before the current line

Shift-Return: **new-line-before** - Place a new line before the current line

Shift-Right: **forward-char-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one character, adjusting the selection range to new position ; *Toolbar Search Commands*: Move forward one character, extending the selection

Shift-Right: **forward-char-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one character, adjusting the selection range to new position ; *Toolbar Search Commands*: Move forward one character, extending the selection

Shift-Tab: **backward-tab** - Outdent line at current position

Shift-Tab: **backward-tab** - Outdent line at current position

Shift-Tab: **outdent-region** - Outdent the selected region one level of indentation. Set sel to None to use preference to determine selection behavior, or "never-select" to unselect after indent, "always-select" to always select after indent, or "retain-select" to retain current selection after indent.

Shift-Up: **previous-line-extend** - Move to previous screen line, adjusting the selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, 'fnb' for first non-blank char, or 'xcode' to simulate XCode style Shift-Alt line selection.

Shift-Up: **previous-line-extend** - Move to previous screen line, adjusting the selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, 'fnb' for first non-blank char, or 'xcode' to simulate XCode style Shift-Alt line selection.

Tab: **tab-key** - Implement the tab key, the action of which is configurable by preference

Tab: **tab-key** - Implement the tab key, the action of which is configurable by preference

Up: **previous-line** - Move to previous screen line, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Up: **previous-line** - Move to previous screen line, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Visual-Esc: **exit-visual-mode** - Exit visual mode and return back to default mode

Visual-Esc: **exit-visual-mode** - Exit visual mode and return back to default mode

21.7. Brief Personality

This section documents all the default key bindings for the **Brief** keyboard personality, set by the **Personality** preference.

Alt-0: **set-bookmark(mark="0")** - Set a bookmark at current location on the editor. Mark is the project-wide textual name of the bookmark.

Alt-1: **fold-python-methods** - Fold up all Python methods, expand all classes, and leave other fold points alone

Alt-1: **set-bookmark(mark="1")** - Set a bookmark at current location on the editor. Mark is the project-wide textual name of the bookmark.

Alt-2: **fold-python-classes** - Fold up all Python classes but leave other fold points alone

Alt-2: **set-bookmark(mark="2")** - Set a bookmark at current location on the editor. Mark is the project-wide textual name of the bookmark.

Alt-3: **fold-python-classes-and-defs** - Fold up all Python classes, methods, and functions but leave other fold points alone

Alt-3: **set-bookmark(mark="3")** - Set a bookmark at current location on the editor. Mark is the project-wide textual name of the bookmark.

Alt-4: **set-bookmark(mark="4")** - Set a bookmark at current location on the editor. Mark is the project-wide textual name of the bookmark.

Alt-5: **set-bookmark(mark="5")** - Set a bookmark at current location on the editor. Mark is the project-wide textual name of the bookmark.

Alt-6: **set-bookmark(mark="6")** - Set a bookmark at current location on the editor. Mark is the project-wide textual name of the bookmark.

Alt-7: **set-bookmark(mark="7")** - Set a bookmark at current location on the editor. Mark is the project-wide textual name of the bookmark.

Alt-8: **set-bookmark(mark="8")** - Set a bookmark at current location on the editor. Mark is the project-wide textual name of the bookmark.

Alt-9: **set-bookmark(mark="9")** - Set a bookmark at current location on the editor. Mark is the project-wide textual name of the bookmark.

Alt-A: **toggle-mark-command(select_right=2)** - Change between text-marking and non-text-marking mode. Style is "char" for stream select, "block" for rectangular select, and "line" for line select. Set select_right=1 to select the character to right of the cursor when marking is toggled on.

Alt-BackSpace: **backward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word behind of the cursor ; *Toolbar Search Commands*: Delete word behind the cursor

Alt-C: **toggle-mark-command(style="block")** - Change between text-marking and non-text-marking mode. Style is "char" for stream select, "block" for rectangular select, and "line" for line select. Set select_right=1 to select the character to right of the cursor when marking is toggled on.

Alt-D: **delete-selected-lines** - Delete the line or range of lines that contain the current selection. This duplicates what the editor command delete-line does.

Alt-D: **kill-line** - Kill rest of line from cursor to end of line, and place it into the clipboard with any other contiguously removed lines. End-of-line is removed only if there is nothing between the cursor and the end of the line.

Alt-Delete: **backward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word behind of the cursor ; *Toolbar Search Commands*: Delete word behind the cursor

Alt-Down: **fold-expand-more-current** - Expand the current fold point one more level

Alt-E: **open-gui** - Open a file from disk, prompting with file selection dialog if necessary

Alt-End: **fold-expand-all** - Expand all fold points in the current file

Alt-F11: **prev-points-of-use-match** - Display the previous match in the active points of use tool

Alt-F12: **next-points-of-use-match** - Display the next match in the active points of use tool

Alt-F3: **search** - Bring up the search manager in search mode.

Alt-F4: **close-window** - Close the current window and all documents and panels in it

Alt-F5: **run-to-cursor** - Run to current cursor position

Alt-F5: **search-sel-backward** - Search backward using current selection

Alt-F6: **run-failed-tests** - Re-run all the previously failed tests. The tests are debugged when debug is True.

Alt-F7: **run-last-tests** - Run again the last group of tests that were run. The tests are debugged when debug is True.

Alt-G: **goto-line** - Position cursor at start of given line number

Alt-H: **goto-selected-symbol-defn** - Goto the definition of the selected source symbol, optionally showing the definition in another split if one is available and other_split is True.

Alt-Home: **fold-collapse-all** - Collapse all fold points in the current file

Alt-I: **toggle-overtype** - Toggle status of overtyping mode

Alt-J: **show-bookmarks** - Show a list of all currently defined bookmarks

Alt-K: **kill-line** - Kill rest of line from cursor to end of line, and place it into the clipboard with any other contiguously removed lines. End-of-line is removed only if there is nothing between the cursor and the end of the line.

Alt-L: **toggle-mark-command(style="line")** - Change between text-marking and non-text-marking mode. Style is "char" for stream select, "block" for rectangular select, and "line" for line select. Set select_right=1 to select the character to right of the cursor when marking is toggled on.

Alt-Left: **visit-history-previous** - Move back in history to previous visited editor position

Alt-M: **toggle-mark-command(select_right=1)** - Change between text-marking and non-text-marking mode. Style is "char" for stream select, "block" for rectangular select, and "line" for line select. Set select_right=1 to select the character to right of the cursor when marking is toggled on.

Alt-Minus: **previous-document** - Move to the previous document alphabetically in the list of documents open in the current window

Alt-N: **next-document** - Move to the next document alphabetically in the list of documents open in the current window

Alt-Page_Down: **fold-expand-all-current** - Expand the current fold point completely

Alt-Page_Up: **fold-collapse-all-current** - Collapse the current fold point completely

Alt-R: **insert-file** - Insert a file at current cursor position, prompting user for file selection

Alt-Return: **new-line** - Place a new line at the current cursor position

Alt-Right: **visit-history-next** - Move forward in history to next visited editor position

Alt-S: **search** - Bring up the search manager in search mode.

Alt-Slash: **fold-toggle** - Toggle the current fold point

Alt-T: **replace** - Bring up the search manager in replace mode.

Alt-U: **undo** - Undo last action

Alt-Up: **fold-collapse-more-current** - Collapse the current fold point one more level

Alt-W: **save** - Save active document. Also close it if close is True.

Alt-X: **quit** - Quit the application.

BackSpace: **backward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character behind the cursor, or the current selection if not empty. ; *Toolbar Search Commands*: Delete character behind the cursor

Ctrl-1: **activate-file-option-menu** - Activate the file menu for the editor.

Ctrl-2: **activate-symbol-option-menu-1** - Activate the 1st symbol menu for the editor.

Ctrl-3: **activate-symbol-option-menu-2** - Activate the 2nd symbol menu for the editor.

Ctrl-4: **activate-symbol-option-menu-3** - Activate the 3rd symbol menu for the editor.

Ctrl-5: **activate-symbol-option-menu-4** - Activate the 4th symbol menu for the editor.

Ctrl-6: **activate-symbol-option-menu-5** - Activate the 5th symbol menu for the editor.

Ctrl-=: **indent-to-match** - Indent the current line or selected region to match indentation of preceding non-blank line. Set toggle=True to indent instead of one level higher if already at the matching position.

Ctrl-Alt-Down: **goto-next-bookmark(current_file_only=True)** - Go to the next bookmark, or the first one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-Alt-F6: **debug-failed-tests** - Re-run all the previously failed tests in the debugger.

Ctrl-Alt-F7: **debug-last-tests** - Debug the last group of tests that were run.

Ctrl-Alt-Left: **goto-previous-bookmark** - Go to the previous bookmark in the bookmark list, or the last one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-Alt-Right: **goto-next-bookmark** - Go to the next bookmark, or the first one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-Alt-Up: **goto-previous-bookmark(current_file_only=True)** - Go to the previous bookmark in the bookmark list, or the last one if no bookmark is selected. Stays within the file in the current editor when current_file_only is True.

Ctrl-Apostrophe: **enclose(start="", end="")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-BackSpace: **backward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word behind of the cursor ; *Toolbar Search Commands*: Delete word behind the cursor

Ctrl-Braceleft: **enclose(start="{", end="}")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-Bracketleft: **enclose(start="[", end="]")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-C: **center-cursor** - Scroll so cursor is centered on display

Ctrl-C: **copy** - Action varies according to focus: *Active Editor Commands*: Copy selected text ; *Document Viewer Commands*: Copy any selected text. ; *Exceptions Commands*: Copy the exception traceback to the clipboard ; *Search Manager Instance Commands*: Copy selected text ; *Toolbar Search Commands*: Cut selection

Ctrl-D: **scroll-text-down** - Scroll text down a line w/o moving cursor's relative position on screen. Repeat is number of lines or if >0 and <1.0 then percent of screen. Set move_cursor to False to leave cursor in current position within the source, otherwise it is moved so the cursor remains on same screen line.

Ctrl-Delete: **forward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word in front of the cursor ; *Toolbar Search Commands*: Delete word in front of the cursor

Ctrl-Down: **select-less** - Select less code; undoes the last select-more command

Ctrl-E: **scroll-text-up** - Scroll text up a line w/o moving cursor's relative position on screen. Repeat is number of lines or if >0 and <1.0 then percent of screen. Set move_cursor to False to leave cursor in current position within the source, otherwise it is moved so the cursor remains on same screen line.

Ctrl-End: **end-of-document** - Move cursor to end of document

Ctrl-F12: **command-by-name** - Execute given command by name, collecting any args as needed

Ctrl-F3: **search-sel-forward** - Search forward using current selection

Ctrl-F4: **close** - Close active document. Abandon any changes when ignore_changes is True. Close empty windows when close_window is true and quit if all document windows closed when can_quit is true.

Ctrl-F5: **debug-kill** - Stop debugging

Ctrl-F6: **step-over** - Step over current instruction

Ctrl-F8: **start-select-line** - Turn on auto-select mode line by line

Ctrl-F9: **break-clear-all** - Clear all breakpoints

Ctrl-Home: **start-of-document** - Move cursor to start of document

Ctrl-Insert: **copy** - Action varies according to focus: *Active Editor Commands*: Copy selected text ; *Document Viewer Commands*: Copy any selected text. ; *Exceptions Commands*: Copy the exception traceback to the clipboard ; *Search Manager Instance Commands*: Copy selected text ; *Toolbar Search Commands*: Cut selection

Ctrl-K: **forward-delete-word** - Action varies according to focus: *Active Editor Commands*: Delete one word in front of the cursor ; *Toolbar Search Commands*: Delete word in front of the cursor

Ctrl-Left: **backward-word** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move backward one word

Ctrl-Less: **enclose(start="<", end=">")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-Minus: **kill-buffer** - Close the current text file

Ctrl-Minus: **zoom-out** - Action varies according to focus: *Document Viewer Commands*: Decrease documentation font size; *General Editor Commands*: Zoom out, increasing the text display size temporarily by one font size

Ctrl-Next: **forward-page** - Move cursor forward one page

Ctrl-PageDown: **end-of-document** - Move cursor to end of document

Ctrl-PageUp: **beginning-of-document**

Ctrl-Page_Down: **next-document** - Move to the next document alphabetically in the list of documents open in the current window

Ctrl-Page_Up: **previous-document** - Move to the previous document alphabetically in the list of documents open in the current window

Ctrl-Parenleft: **enclose(start="(", end=")")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-Plus: **zoom-in** - Action varies according to focus: *Document Viewer Commands*: Increase documentation font size; *General Editor Commands*: Zoom in, increasing the text display size temporarily by one font size

Ctrl-Pointer_Button1: **goto-clicked-symbol-defn** - Goto the definition of the source symbol that was last clicked on, optionally showing the definition in another split if one is available and other_split is True.

Ctrl-Prior: **backward-page** - Move cursor backward one page

Ctrl-Quotedbl: **enclose(start='"', end='")** - Enclose the selection or the rest of the current line when there is no selection with the given start and end strings. The caret is moved to the end of the enclosed text.

Ctrl-Quoteleft: **begin-visited-document-cycle(move_back=True, back_key="Ctrl-Quoteleft", forward_key="Ctrl-AsciiTilde")** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-R: **initiate-repeat-4** - Enter a sequence of digits indicating number of times to repeat the subsequent command or keystroke.

Ctrl-Return: **new-line-after** - Place a new line after the current line

Ctrl-Right: **forward-word** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one word. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move forward one word

Ctrl-Shift-Delete: **delete-lines**

Ctrl-Shift-Down: **scroll-text-down** - Scroll text down a line w/o moving cursor's relative position on screen. Repeat is number of lines or if >0 and <1.0 then percent of screen. Set move_cursor to False to leave cursor in current position within the source, otherwise it is moved so the cursor remains on same screen line.

Ctrl-Shift-End: **end-of-document-extend** - Move cursor to end of document, adjusting the selection range to new position

Ctrl-Shift-F3: **search-sel-backward** - Search backward using current selection

Ctrl-Shift-F5: **debug-stop** - Pause free-running execution at current program counter

Ctrl-Shift-F6: **debug-all-tests** - Debug all the tests in testing panel.

Ctrl-Shift-F7: **debug-current-tests** - Runs the current test or tests, if possible. The current tests are determined by the current position in the active view.

Ctrl-Shift-F9: Multiple commands (first available is executed):

- **break-disable-all** - Disable all breakpoints
- **break-enable-all** - Enable all breakpoints

Ctrl-Shift-Home: **start-of-document-extend** - Move cursor to start of document, adjusting the selection range to new position

Ctrl-Shift-I: **add-current-file-to-project** - Add the frontmost currently open file to project

Ctrl-Shift-ISO_Left_Tab: **begin-visited-document-cycle(move_back=False)** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-Shift-Left: **backward-word-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one word, adjusting the selection range to new position. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move backward one word, extending the selection

Ctrl-Shift-Next: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Ctrl-Shift-Page_Down: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Ctrl-Shift-Page_Up: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Ctrl-Shift-Prior: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Ctrl-Shift-Right: **forward-word-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one word, adjusting the selection range to new position. Optionally, provide a string that contains the delimiters to define which characters are part of a word. Gravity may be "start" or "end" to indicate whether cursor is placed at start or end of the word.; *Toolbar Search Commands*: Move forward one word, extending the selection

Ctrl-Shift-Tab: **begin-visited-document-cycle(move_back=False)** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-Shift-Up: **scroll-text-up** - Scroll text up a line w/o moving cursor's relative position on screen. Repeat is number of lines or if >0 and <1.0 then percent of screen. Set move_cursor to False to leave cursor in current position within the source, otherwise it is moved so the cursor remains on same screen line.

Ctrl-T: **forward-tab** - Action varies according to focus: *Active Editor Commands*: Place a tab character at the current cursor position ; *Search Manager Instance Commands*: Place a forward tab at the current cursor position in search or replace string

Ctrl-Tab: **begin-visited-document-cycle(move_back=True)** - Start moving between documents in the order they were visited. Starts modal key interaction that ends when a key other than tab is seen or ctrl is released.

Ctrl-U: **redo** - Redo last action

Ctrl-Up: **select-more** - Select more code on either the current line or larger multi-line blocks.

Ctrl-V: **paste** - Action varies according to focus: *Active Editor Commands*: Paste text from clipboard ; *Search Manager Instance Commands*: Paste text from clipboard ; *Toolbar Search Commands*: Paste from clipboard

Ctrl-X: **cut** - Action varies according to focus: *Active Editor Commands*: Cut selected text ; *Search Manager Instance Commands*: Cut selected text ; *Toolbar Search Commands*: Cut selection

Ctrl-Z: **undo** - Undo last action

Delete: **forward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character in front of the cursor ; *Toolbar Search Commands*: Delete character in front of the cursor

Down: **next-line** - Move to screen next line, optionally repositioning character within line: 'same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

End: **cursor-end** - Bring cursor to end of line, to end of visible area, or to end of document each successive consecutive invocation of this command.

End: **end-of-document** - Move cursor to end of document

End: **end-of-line** - Action varies according to focus: *Active Editor Commands*: Move to end of current line; *Toolbar Search Commands*: Move to the end of the toolbar search entry

End End End: **end-of-document** - Move cursor to end of document

F1: Multiple commands (first available is executed):

- **show-horizontal-tools** - Show the horizontal tool area
- **minimize-horizontal-tools** - Minimize the horizontal tool area

F10: **command-by-name** - Execute given command by name, collecting any args as needed

F11: **frame-up** - Move up the current debug stack

F12: **frame-down** - Move down the current debug stack

F2: Multiple commands (first available is executed):

- **show-vertical-tools** - Show the vertical tool area
- **minimize-vertical-tools** - Minimize the vertical tool area

F3: **search-forward** - Search again using the search manager's current settings in forward direction

F3: **split-vertically** - Split current view vertically. Create new editor in new view when new==1.

F4: **goto-selected-symbol-defn** - Goto the definition of the selected source symbol, optionally showing the definition in another split if one is available and other_split is True.

F4: **unsplit** - Unsplit all editors so there's only one. Action specifies how to choose the remaining displayed editor. One of:

```
current -- Show current editor
close   -- Close current editor before unsplitting
recent  -- Change to recent buffer before unsplitting
recent-or-close -- Change to recent buffer before closing
split, or close the current buffer if there is only
one split left.
```

NOTE: The parameters for this command are subject to change in the future.

F5: **debug-continue** - Continue (or start) running, to next breakpoint

F5: **search** - Bring up the search manager in search mode.

F6: **replace** - Bring up the search manager in replace mode.

F6: **step-over-statement** - Step over current statement

F7: **start-kbd-macro** - Start definition of a keyboard macro. If register=None then the user is prompted to enter a letter a-z under which to file the macro. Otherwise, register 'a' is used by default.

F7: **step-into** - Step into current execution point, or start debugging at first line

F8: **execute-kbd-macro** - Execute most recently recorded keyboard macro. If register is None then the user is asked to enter a letter a-z for the register where the macro is filed. Otherwise, register 'a' is used by default.

F8: **step-out** - Step out of the current function or method

F9: Multiple commands (first available is executed):

- **break-set** - Set a new regular breakpoint on current line
- **break-clear** - Clear the breakpoint on the current line

Home: **beginning-of-line-text** - Move to end of the leading white space, if any, on the current line. If toggle is True, moves to the beginning of the line if already at the end of the leading white space (and vice versa).

Home: **cursor-home** - Bring cursor to start of line, to start of visible area, or to start of document each successive consecutive invocation of this command.

Home: **start-of-document** - Move cursor to start of document

Home Home Home: **start-of-document** - Move cursor to start of document

ISO_Left_Tab: **backward-tab** - Outdent line at current position

Insert: **paste** - Action varies according to focus: *Active Editor Commands*: Paste text from clipboard ; *Search Manager Instance Commands*: Paste text from clipboard ; *Toolbar Search Commands*: Paste from clipboard

Insert: **toggle-overtype** - Toggle status of overtyping mode

Left: **backward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one character ; *Toolbar Search Commands*: Move backward one character

Next: **forward-page** - Move cursor forward one page

Page_Down: **forward-page** - Move cursor forward one page

Page_Up: **backward-page** - Move cursor backward one page

Prior: **backward-page** - Move cursor backward one page

Return: **new-line** - Place a new line at the current cursor position

Right: **forward-char** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one character ; *Toolbar Search Commands*: Move forward one character

Shift-Alt-A: **diff-merge-a-b**

Shift-Alt-B: **diff-merge-b-a**

Shift-Alt-Down: **next-line-extend-rect** - Move to next screen line, adjusting the rectangular selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Shift-Alt-Left: **backward-char-extend-rect** - Move cursor backward one character, adjusting the rectangular selection range to new position

Shift-Alt-N: **diff-next**

Shift-Alt-P: **diff-previous**

Shift-Alt-Right: **forward-char-extend-rect** - Move cursor forward one character, adjusting the rectangular selection range to new position

Shift-Alt-Up: **previous-line-extend-rect** - Move to previous screen line, adjusting the rectangular selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Shift-BackSpace: **backward-delete-char** - Action varies according to focus: *Active Editor Commands*: Delete one character behind the cursor, or the current selection if not empty. ; *Toolbar Search Commands*: Delete character behind the cursor

Shift-Ctrl-F8: **start-select-block** - Turn on auto-select block mode

Shift-Delete: **cut** - Action varies according to focus: *Active Editor Commands*: Cut selected text ; *Search Manager Instance Commands*: Cut selected text ; *Toolbar Search Commands*: Cut selection

Shift-Down: **next-line-extend** - Move to next screen line, adjusting the selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, 'fnb' for first non-blank char, or 'xcode' to simulate XCode style Shift-Alt line selection.

Shift-End: **end-of-line** - Action varies according to focus: *Active Editor Commands*: Move to end of current line; *Toolbar Search Commands*: Move to the end of the toolbar search entry

Shift-End: **end-of-line-extend** - Action varies according to focus: *Active Editor Commands*: Move to end of current line, adjusting the selection range to new position ; *Toolbar Search Commands*: Move to the end of the toolbar search entry, extending the selection

Shift-F1: **move-focus** - Move the keyboard focus forward within the Window to the next editable area

Shift-F11: **frame-show** - Show the position (thread and stack frame) where the debugger originally stopped

Shift-F2: Multiple commands (first available is executed):

- **enter-fullscreen** - Hide both the vertical and horizontal tool areas and toolbar, saving previous state so it can be restored later with exit_fullscreen
- **exit-fullscreen** - Restore previous non-fullscreen state of all tools and tool bar

Shift-F3: **search-backward** - Search again using the search manager's current settings in backward direction

Shift-F4: **find-points-of-use** - Find points of use for a symbol. The symbol defaults to the active selection.

Shift-F5: **debug-file** - Start debugging the current file (rather than the main entry point)

Shift-F5: **search-forward** - Search again using the search manager's current settings in forward direction

Shift-F6: **replace-and-search** - Replace current selection and search again.

Shift-F6: **run-all-tests** - Runs all the tests in testing panel.

Shift-F7: **run-current-tests** - Runs the current test or tests, if possible. The current tests are determined by the current position in the active view. The tests are debugged when debug is True.

Shift-F7: **stop-kbd-macro** - Stop definition of a keyboard macro

Shift-F8: **start-select-char** - Turn on auto-select mode character by character

Shift-F9: Multiple commands (first available is executed):

- **break-enable** - Enable the breakpoint on the current line
- **break-disable** - Disable the breakpoint on current line

Shift-Home: **beginning-of-line** - Action varies according to focus: *Active Editor Commands*: Move to beginning of current line. When toggle is True, moves to the end of the leading white space if already at the beginning of the line (and vice versa).; *Toolbar Search Commands*: Move to the beginning of the toolbar search entry

Shift-Home: **beginning-of-line-text-extend** - Move to end of the leading white space, if any, on the current line, adjusting the selection range to the new position. If toggle is True, moves to the beginning of the line if already at the end of the leading white space (and vice versa).

Shift-Insert: **paste** - Action varies according to focus: *Active Editor Commands*: Paste text from clipboard ; *Search Manager Instance Commands*: Paste text from clipboard ; *Toolbar Search Commands*: Paste from clipboard

Shift-Left: **backward-char-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor backward one character, adjusting the selection range to new position ; *Toolbar Search Commands*: Move backward one character, extending the selection

Shift-Next: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Shift-Page_Down: **forward-page-extend** - Move cursor forward one page, adjusting the selection range to new position

Shift-Page_Up: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Shift-Prior: **backward-page-extend** - Move cursor backward one page, adjusting the selection range to new position

Shift-Return: **new-line-before** - Place a new line before the current line

Shift-Right: **forward-char-extend** - Action varies according to focus: *Active Editor Commands*: Move cursor forward one character, adjusting the selection range to new position ; *Toolbar Search Commands*: Move forward one character, extending the selection

Shift-Tab: **backward-tab** - Outdent line at current position

Shift-Up: **previous-line-extend** - Move to previous screen line, adjusting the selection range to new position, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, 'fnb' for first non-blank char, or 'xcode' to simulate XCode style Shift-Alt line selection.

Tab: **tab-key** - Implement the tab key, the action of which is configurable by preference

Up: **previous-line** - Move to previous screen line, optionally repositioning character within line: same' to leave in same horizontal position, 'start' at start, 'end' at end, or 'fnb' for first non-blank char.

Visual-Esc: **exit-visual-mode** - Exit visual mode and return back to default mode

License Information

Wing IDE is a commercial product that is based on a number of open source technologies. Although the product source code is available for Wing IDE Professional users (with signed non-disclosure agreement) the product is not itself open source.

The following sections describe the licensing of the product as a whole (the End User License Agreement) and provide required legal statements for the incorporated open source components.

22.1. Wing IDE Software License

This End User License Agreement (EULA) is a CONTRACT between you (either an individual or a single entity) and Wingware, which covers your use of "Wing IDE Professional" and related software components. All such software is referred to herein as the "Software Product." A software license and a license key or serial number ("Software Product License"), issued to a designated user, only by Wingware or its authorized agents, is required for each user of the Software Product. If you do not agree to the terms of this EULA, then do not install or use the Software Product or the Software Product License. By explicitly accepting this EULA you are acknowledging and agreeing to be bound by the following terms:

1. EVALUATION LICENSE WARNING

This Software Product can be used in conjunction with a free evaluation Software Product License. If you are using such an evaluation Software Product License, you may use the Software Product only to evaluate its suitability for purchase. Evaluation Software Product Licenses have an expiration date and most of the features of the software will be disabled after that date. WINGWARE BEARS NO LIABILITY FOR ANY DAMAGES RESULTING FROM USE (OR ATTEMPTED USE AFTER THE EXPIRATION DATE) OF THE SOFTWARE PRODUCT, AND HAS NO DUTY TO PROVIDE ANY SUPPORT BEFORE OR AFTER THE EXPIRATION DATE OF AN EVALUATION LICENSE.

2. GRANT OF NON-EXCLUSIVE LICENSE

Wingware grants the non-exclusive, non-transferable right for a single user to use this Software Product for each license purchased. Each additional user of the Software Product requires an additional Software Product License. This includes users working on operating systems where the Software Product is compiled from source code by the user or a third party.

Wingware grants you the right to modify, alter, improve, or enhance the Software Product without limitation, except as described in this EULA.

Although rights to modification of the Software Product are granted by this EULA, you may not tamper with, alter, or use the Software Product in a way that disables, circumvents, or otherwise defeats its built-in licensing verification and enforcement capabilities. The right to modification of the Software Product also does not include the right to remove or alter any trademark, logo, copyright or other proprietary notice, legend, symbol or label in the Software Product.

You may at your discretion distribute patch files containing any modifications or improvements made to the Software Product, other than those that are aimed at disabling or circumventing its built-in license verification capabilities, or that result in the removal or alteration of any trademark, logo, copyright, or other proprietary notice, legend, symbol or label in the Software Product. This right does not include the right to distribute substantial portions of the original source, where distribution rights are limited to contextual information normally existing in software patch files.

You may at your discretion designate license terms, open source or otherwise, for all modifications or improvements made by you. Wingware has no special rights to any such modifications or improvements.

You may make copies of the Software Product as reasonably necessary for its use. Each copy must reproduce all copyright and other proprietary rights notices on or in the Software Product.

You may install your Software Product License only on computer systems and user accounts that are used by you, the licensee. You may also make copies of the

Software Product License as necessary for backup and/or archival purposes. No other copies or installations may be made.

All rights not expressly granted to you are retained by Wingware.

2.1 NON-COMMERCIAL USE LICENSES

Wingware provides Non-Commercial Use licenses to the following types of users: (a) publicly funded charities, (b) universities, colleges, and other educational institutions (including, but not limited to elementary schools, middle schools, high schools, and community colleges), (c) students at any of these types of educational institutions, (d) individuals or entities who are under contract by the above-stated organizations and using the Software Product exclusively for such charitable or educational clients, and (d) other individual users who use the Software Product for unpaid personal use only (for example, unpaid hobby, learning, or entertainment).

Non-Commercial Use licenses purchased by companies; organizations other than publicly funded charities; government divisions, agencies, or offices; or any other individual or entity deriving income, directly or indirectly, from their use of the Software Product are invalid and may not be used until the license is upgraded by paying the price difference between the Non-Commercial Use and Commercial Use license for the Software Product.

Wingware, a Delaware corporation, reserves the right to further clarify the terms of Non-Commercial Use at its sole determination.

3. INTELLECTUAL PROPERTY RIGHTS RESERVED BY WINGWARE

The Software Product is owned by Wingware and is protected by United States and international copyright laws and treaties, as well as other intellectual property laws and treaties. You must not remove or alter any copyright notices on any copies of the Software Product. This Software Product copy is licensed, not sold. You may not use, copy, or distribute the Software Product, except as granted by this EULA, without written authorization from Wingware or its designated agents. Furthermore, this EULA does not grant you any rights in connection with any trademarks or service marks of Wingware. Wingware reserves all intellectual property rights, including copyrights, and trademark rights.

4. LIMITED RIGHTS TO TRANSFER

You may not rent, lease, lend, or in any way distribute or transfer any rights in this EULA or the Software Product to third parties without Wingware's written approval.

However, companies that purchase a Commercial Use license may from time to time, as employees come and go or roles change, transfer that license to another individual, provided that the prior user of the license ceases to use the license immediately after the transfer has been made.

All transfers of a license to another individual are subject to the recipient's acceptance of the terms of the EULA and are null and void in the event that the prior user continues to use the license or otherwise fails to relinquish their rights to the Software Product.

5. INDEMNIFICATION

You hereby agree to indemnify Wingware against and hold harmless Wingware from any claims, lawsuits or other losses that arise out of your breach of any provision of this EULA.

6. THIRD PARTY RIGHTS

Any software provided along with the Software Product that is associated with a separate license agreement is licensed to you under the terms of that license agreement. This license does not apply to those portions of the Software Product. Copies of these third party licenses are included in all copies of the Software Product.

7. SUPPORT SERVICES

Wingware may provide you with support services related to the Software Product. Use of any such support services is governed by Wingware policies and programs described in online documentation and/or other Wingware-provided materials.

As part of these support services, Wingware may make available bug lists, planned feature lists, and other supplemental informational materials. WINGWARE MAKES NO WARRANTY OF ANY KIND FOR THESE MATERIALS AND ASSUMES NO LIABILITY WHATSOEVER FOR DAMAGES RESULTING FROM ANY USE OF THESE MATERIALS. FURTHERMORE, YOU MAY NOT USE ANY MATERIALS PROVIDED IN THIS WAY TO SUPPORT ANY CLAIM MADE AGAINST WINGWARE.

Any supplemental software code or related materials that Wingware provides to you as part of the support services, in periodic updates to the Software Product or otherwise, is to be considered part of the Software Product and is subject to the terms and conditions of this EULA.

With respect to any technical information you provide to Wingware as part of the support services, Wingware may use such information for its business purposes without restriction, including for product support and development. Wingware will not use such technical information in a form that personally identifies you without first obtaining your permission.

9. TERMINATION WITHOUT PREJUDICE TO ANY OTHER RIGHTS

Wingware may terminate this EULA if you fail to comply with any term or condition of this EULA. In such event, you must destroy all copies of the Software Product and Software Product Licenses.

10 U.S. GOVERNMENT USE

.

If the Software Product is licensed under a U.S. Government contract, you acknowledge that the software and related documentation are "commercial items," as defined in 48 C.F.R 2.01, consisting of "commercial computer software" and "commercial computer software documentation," as such terms are used in 48 C.F.R. 12.212 and 48 C.F.R. 227.7202-1. You also acknowledge that the software is "commercial computer software" as defined in 48 C.F.R. 252.227-7014(a)(1). U.S. Government agencies and entities and others acquiring under a U.S. Government contract shall have only those rights, and shall be subject to all restrictions, set forth in this EULA. Contractor/manufacturer is Wingware, P.O. Box 400527, Cambridge, MA 02140-0006, USA.

11 EXPORT RESTRICTIONS

.

You will not download, export, or re-export the Software Product, any part thereof, or any software, tool, process, or service that is the direct product of the Software Product, to any country, person, or entity -- even to foreign units of your own company -- if such a transfer is in violation of U.S. export restrictions.

12 NO WARRANTIES

.

YOU ACCEPT THE SOFTWARE PRODUCT AND SOFTWARE PRODUCT LICENSE "AS IS," AND WINGWARE AND ITS THIRD PARTY SUPPLIERS AND LICENSORS MAKE NO WARRANTY AS TO ITS USE, PERFORMANCE, OR OTHERWISE. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, WINGWARE AND ITS THIRD PARTY SUPPLIERS AND LICENSORS DISCLAIM ALL OTHER REPRESENTATIONS, WARRANTIES, AND CONDITIONS, EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, SATISFACTORY QUALITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NON-INFRINGEMENT. THE ENTIRE RISK ARISING OUT OF USE OR PERFORMANCE OF THE SOFTWARE PRODUCT REMAINS WITH YOU.

13 LIMITATION OF LIABILITY

.

THIS LIMITATION OF LIABILITY IS TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW. IN NO EVENT SHALL WINGWARE OR ITS THIRD PARTY SUPPLIERS AND LICENSORS BE LIABLE FOR ANY COSTS OF SUBSTITUTE PRODUCTS OR SERVICES, OR FOR ANY SPECIAL, INCIDENTAL, INDIRECT, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, OR LOSS OF BUSINESS INFORMATION) ARISING OUT OF THIS EULA OR THE USE OF OR INABILITY TO USE THE SOFTWARE PRODUCT OR THE FAILURE TO PROVIDE SUPPORT SERVICES, EVEN IF WINGWARE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN ANY CASE, WINGWARE'S, AND ITS THIRD PARTY SUPPLIERS' AND LICENSORS', ENTIRE LIABILITY ARISING OUT OF THIS EULA SHALL BE LIMITED TO THE LESSER OF THE AMOUNT ACTUALLY PAID BY YOU FOR THE SOFTWARE PRODUCT OR THE PRODUCT LIST PRICE; PROVIDED, HOWEVER, THAT IF YOU HAVE ENTERED INTO AN WINGWARE SUPPORT SERVICES AGREEMENT, WINGWARE'S ENTIRE LIABILITY REGARDING SUPPORT SERVICES SHALL BE GOVERNED BY THE TERMS OF THAT AGREEMENT.

14 HIGH RISK ACTIVITIES

.

The Software Product is not fault-tolerant and is not designed, manufactured or intended for use or resale as on-line control equipment in hazardous environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines, or weapons systems, in which the failure of the Software Product, or any software, tool, process, or service that was developed using the Software Product, could lead directly to death, personal injury, or severe physical or environmental damage ("High Risk Activities"). Accordingly, Wingware and its suppliers and licensors specifically disclaim any express or implied warranty of fitness for High Risk Activities. You agree that Wingware and its suppliers and licensors will not be liable for any claims or damages arising from the use of the Software Product, or any software, tool, process, or service that was developed using the Software Product, in such applications.

15 GOVERNING LAW; ENTIRE AGREEMENT ; DISPUTE RESOLUTION

.

This EULA is governed by the laws of the Commonwealth of Massachusetts, U.S.A., excluding the application of any conflict of law rules. The United Nations Convention on Contracts for the International Sale of Goods shall not apply.

This EULA is the entire agreement between Wingware and you, and supersedes any other communications or advertising with respect to the Software Product; this

EULA may be modified only by written agreement signed by authorized representatives of you and Wingware.

Unless otherwise agreed in writing, all disputes relating to this EULA (excepting any dispute relating to intellectual property rights) shall be subject to final and binding arbitration in the State of Massachusetts, in accordance with the Licensing Agreement Arbitration Rules of the American Arbitration Association, with the losing party paying all costs of arbitration. Arbitration must be by a member of the American Arbitration Association. If any dispute arises under this EULA, the prevailing party shall be reimbursed by the other party for any and all legal fees and costs associated therewith.

16 GENERAL

.

If any provision of this EULA is held invalid, the remainder of this EULA shall continue in full force and effect.

A waiver by either party of any term or condition of this EULA or any breach thereof, in any one instance, shall not waive such term or condition or any subsequent breach thereof.

17 OUTSIDE THE U.S.

.

If you are located outside the U.S., then the provisions of this Section shall apply. Les parties aux présentes confirment leur volonté que cette convention de même que tous les documents y compris tout avis qui s'y rattache, soient redigés en langue anglaise. (translation: "The parties confirm that this EULA and all related documentation is and will be in the English language.") You are responsible for complying with any local laws in your jurisdiction which might impact your right to import, export or use the Software Product, and you represent that you have complied with any regulations or registration procedures required by applicable law to make this license enforceable.

18 TRADEMARKS

.

The following are trademarks or registered trademarks of Wingware: Wingware, the feather logo, Wing IDE, Wing IDE 101, Wing IDE Personal, Wing IDE Professional, Wing IDE Enterprise, Wing Debugger, and "The Intelligent Development Environment for Python Programmers"

19 CONTACT INFORMATION

.

If you have any questions about this EULA, or if you want to contact Wingware for any reason, please direct all correspondence to: Wingware, P.O. Box 400527, Cambridge, MA 02140-0006, United States of America or send email to info at wingware.com.

22.2. Open Source License Information

Wing IDE incorporates the following open source technologies, most of which are under OSI Certified Open Source licenses except as indicated in the footnotes:

- Crystal Clear -- An icon set by Everaldo -- LGPL v. 2.1 [1]
- docutils -- reStructuredText markup processing by David Goodger and contributors-- Public Domain [2]
- parsetools -- Python parse tree conversion tools by John Ehresman -- MIT License
- pexpect -- Sub-process control library by Noah Spurrier, Richard Holden, Marco Molteni, Kimberley Burchett, Robert Stone, Hartmut Goebel, Chad Schroeder, Erick Tryzelaar, Dave Kirby, Ids vander Molen, George Todd, Noel Taylor, Nicolas D. Cesar, Alexander Gattin, Geoffrey Marshall, Francisco Lourenco, Glen Mabey, Karthik Gurusamy, and Fernando Perez -- MIT License
- py2pdf -- Python source to PDF output converter by Dinu Gherman -- MIT License
- PySide -- Python bindings for Qt by Nokia and contributors -- LGPL v. 2.1 [1]
- pysqlite -- Python bindings for sqlite by Gerhard Haering -- BSD-like custom license [4]
- Python -- The Python programming language by Guido van Rossum, PythonLabs, and many contributors -- Python Software Foundation License version 2 [3]
- Python Imaging Library -- Library for image manipulation with Python, written by Secret Labs AB and Fredrik Lundh -- MIT License
- Qt -- Graphical user interface toolkit by many contributors and Digia -- LGPL v. 2.1 [1] [6]
- scintilla -- Source code editor component by Neil Hodgson and contributors -- MIT License
- sqlite -- A self-contained, serverless, zero-configuration, transactional SQL database engine -- Public domain [5]
- Tulliana-1.0 -- An icon set by M. Umut Pulat, based on Nuvola created by David Vignoni -- LGPL v. 2.1 [1]
- A few stock icons from the GTK GUI development framework -- LGPL v. 2.1 [1]

Notes

[1] The LGPL requires us to redistribute the source code for all libraries linked into Wing IDE. All of these modules are readily available on the internet. In some cases we may have modifications that have not yet been incorporated into the official versions; if you wish to obtain a copy of our version of the sources of any of these modules, please email us at info at wingware.com.

[2] Docutils contains a few parts under other licenses (BSD, Python 2.1, Python 2.2, Python 2.3, and GPL). See the COPYING.txt file in the source distribution for details.

[3] The Python Software Foundation License version 2 is an OSI Approved Open Source license. It consists of a stack of licenses that also include other licenses that apply to older parts of the Python code base. All of these are included in the OSI Approved license: PSF License, BeOpen Python License, CNRI Python License, and CWI Python License. The intellectual property rights for Python are managed by the Python Software Foundation.

[4] Not OSI Approved, but similar to other OSI approved licenses. The license grants anyone to use the software for any purpose, including commercial applications.

[5] The source code states the author has disclaimed copyright of the source code. The sqllite.org website states: "All of the deliverable code in SQLite has been dedicated to the public domain by the authors. All code authors, and representatives of the companies they work for, have signed affidavits dedicating their contributions to the public domain and originals of those signed affidavits are stored in a firesafe at the main offices of Hwaci. Anyone is free to copy, modify, publish, use, compile, sell, or distribute the original SQLite code, either in source code form or as a compiled binary, for any purpose, commercial or non-commercial, and by any means."

[6] Qt is available under several licenses. The LGPL v. 2.1 version of the software was used for Wing IDE.

Scintilla Copyright

We are required by the license terms for Scintilla to include the following copyright notice in this documentation:

```
Copyright 1998-2003 by Neil Hodgson <neilh@scintilla.org>

All Rights Reserved

Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted,
provided that the above copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in
supporting documentation.
```

```
NEIL HODGSON DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS
SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS, IN NO EVENT SHALL NEIL HODGSON BE LIABLE FOR ANY
SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER
TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE
OR PERFORMANCE OF THIS SOFTWARE.
```

Python Imaging Library Copyright

We are required by the license terms for Scintilla to include the following copyright notice in this documentation:

```
The Python Imaging Library (PIL) is

    Copyright ï¿½ 1997-2011 by Secret Labs AB
    Copyright ï¿½ 1995-2011 by Fredrik Lundh

By obtaining, using, and/or copying this software and/or its associated documentation,
that you have read, understood, and will comply with the following terms and condition

Permission to use, copy, modify, and distribute this software and its associated docum
any purpose and without fee is hereby granted, provided that the above copyright notic
all copies, and that both that copyright notice and this permission notice appear in s
documentation, and that the name of Secret Labs AB or the author not be used in advert
publicity pertaining to distribution of the software without specific, written prior p

SECRET LABS AB AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, I
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL SECRET LABS AB OR
BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER
FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTH
ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
```